On-line, On-board Evolution of Robot Controllers

N. Bredeche[1], E. Haasdijk[2], and A.E. Eiben[2]

[1]TAO/LRI, INRIA Saclay, Univ. Paris-Sud, CNRS (Orsay, France)
[2]Free University, Amsterdam, The Netherlands
nicolas.bredeche@lri.fr, {e.haasdijk,gusz}@few.vu.nl

Abstract. This paper reports on a feasibility study into the evolution of robot controllers *during* the actual operation of robots (on-line), using only the computational resources within the robots themselves (on-board). We identify the main challenges that these restrictions imply and propose mechanisms to handle them. The resulting algorithm is evaluated in a hybrid system, using the actual robots' processors interfaced with a simulator that represents the environment. The results show that the proposed algorithm is indeed feasible and the particular problems we encountered during this study give hints for further research.

1 Background and Introduction

Evolutionary Computing (EC) has proved a powerful technology for developing robot controllers [4] and has resulted in the establishment of Evolutionary Robotics (ER). The overwhelming majority of ER applications use an off-line flavour of evolution. In these cases an evolutionary algorithm (EA) is used to optimise the robot controllers *before* the robots start their actual operation. This process may rely on real-life fitness evaluations or on a simulation-based assessment of controller quality, but in all cases the EA is executed on one or more computer(s) distinct from the robots. Once the development process has terminated, the controllers are deployed on real robots and remain fixed while the robots go about their given tasks. Thus, during the operational period of the robots, the controllers do not adapt anymore (or at least, not by evolutionary operators [8,9,14]).

The present study was undertaken as part of the Symbrion project [1] that explicitly aims at using evolution on-line. That is, the evolutionary algorithm is required to adapt the robot controllers *during* the actual operation period of the robots. Such a switch from (off-line) optimisation to pervasive adaptation offers advantages in cases where the environment is changing and/or it is impossible to optimize the robots for circumstances in which they will operate (for instance, because they are not known well enough in advance). One of the premises of the Symbrion project is the presence of a large group of robots that form a changing "social environment" for each other, which in turn, necessitates on-line adaptation again. All in all, we aim at a system that is decentralised, on-board, without any master computer that executes the evolutionary operators, and fully autonomous, with no human intervention. These requirements imply two major restrictions:

[1] EU FP7, FET project, Grant No. 216342, http://symbrion.eu/

1. Fitness must be evaluated *in vivo*, i.e., the quality of any given controller is determined by actually using that controller in a robot as it goes about its tasks.
2. All necessary computation must be performed by the robots themselves, implying limited processing power and storage capacity.

The real-life, real-time fitness evaluations are inevitably very noisy because the initial conditions for the genomes under evaluation vary considerably. Whatever the details of the evolutionary mechanism, different controllers will be evaluated under different circumstances; for instance, the nth controller will start at the final location of the $(n-1)$th one. This leads to very dissimilar evaluation conditions and ultimately to very noisy fitness evaluations. The limited processor power and storage capacity implies that we must use a "lightweight" evolutionary algorithm, with a small population per robot. Obviously, this could limit the exploratory power of the EA, with a high risk of premature convergence at a local optimum. Taking these considerations into account, we formulate the following research objectives:

1. Provide an evolutionary mechanism that can cope with noisy fitness evaluations.
2. Provide an evolutionary mechanism that can perform balanced local and global search even with very small populations.

Related work on the on-line, on-board evolution of robot controllers can be roughly divided into two categories:

The distributed online onboard ER approach. Each robot carries one genotype and is controlled by the corresponding phenotype. Robots can reproduce autonomously and asynchronously and create offspring controllers by recombination and/or mutation. Here, the iterative improvement (optimisation) of controllers is the result of the evolutionary process that emerges from the exchange of genetic information among the robots. [17]

The encapsulated online onboard ER approach. A robot has an EA implemented on-board, maintaining a population of controllers inside itself. The EA is running on a local basis and perform the fitness evaluations autonomously. This is typically done in a time-sharing system, where one member of the inner population is activated (i.e., decoded into a controller) at a time and is used for a while to gather feedback on its quality. Here, the iterative improvement (optimisation) of controllers is the result of the EA running on a single robot [10,16]. This can be extended to multiple robots setup, where each robot is completely independant from others.

Note, that both approaches inherently work with a heterogeneous population of robot controllers. The two approaches can also be combined, and often are, resulting in a setup akin to an island model as used in parallel genetic algorithms. In such a combined system, there are two ways of mixing genetic information: intra-island variation (i.e., within the "population" of the encapsulated EA in one robot) and inter-island migration (between two, or more, robots). [6,15,18,11,3]

The work presented here falls in the second category, i.e. the encapsulated approach, explicitly aiming at online adaptation for a single robot.

2 The (1+1)-ONLINE Evolutionary Algorithm

We propose an EA based on the classical (1+1) Evolution Strategy (ES)[13]. In our experiments, the genome consists of the weights in an artificial neural networks (NN) that controls the robot, formally a real-valued vector $\bar{x} = \langle x_1, \ldots, x_n \rangle$. The controlling NN is a perceptron with 9 input nodes (8 sensor inputs and a bias node), no hidden nodes and 2 output nodes (the left and right motor values) –18 weights in total. Thus, the genome is a vector of 18 real values. The perceptron uses a hyperbolic tangent activation function. Variation in a (1+1) ES is necessarily restricted to mutation. This is implemented as straightforward Gaussian mutation, adding values from a distribution $\mathcal{N}(0, \sigma)$ to each x_i in the genotype $\bar{x}$. Parent selection in a singleton population is trivial and for survival selection we rely on the so-called + *strategy*: the child (challenger) replaces the parent (champion) if its fitness is higher. This simple scheme defines the core of our EA, but it is not sufficient to cope with a number of issues in our particular application. Therefore, we extend this basic scheme with a number of advanced features, described below.

Adapting σ values A singleton population is very sensitive to premature convergence to a local optimum. To overcome this problem, we augment the EA with a mechanism that varies the mutation stepsize σ on the fly, switching from local to global search and back, depending on the course of the search. In particular, σ is set to a pre-defined minimum to promote local search whenever a new genome is stored (that is, when the challenger outperforms the champion). Then, σ gradually increases up to a maximum value (i.e., the search shifts towards global search) while the champion outperforms its children. If local search leads to improvements, σ remains low, thus favouring local search. If no improvement is made on a local basis, either because of a neutral landscape or a local optimum, the increasing σ values ensure that the search will move to new regions in the search space.

Recovery period Because we use *in vivo* fitness evaluation, a new genome $\bar{x}$ needs to be "activated" to be evaluated: it has to be decoded into a NN and take over the control of the robot for a while. One of the essential design decisions is to avoid any human intervention during evolution, such as repositioning the robot befor evaluating a new genome. Consequently, a new controller will start where the previous one finished, implying the danger of being penalised for bad behaviour of its predecessor that, for instance, may have manoeuvred itself into an impossibly tight corner. To reduce this effect, we introduce a $recoveryTime$, during which robot behaviour is not taken into account for the fitness value computation. This favours genomes that are efficient at both getting out of trouble during the recovery phase and displaying efficient behavior during the evaluation phase.

Re-evaluation The evaluation of a genome is very noisy because the initial conditions for the genomes vary considerably: an evaluation must start at the final location of the previous evaluation, leading to very dissimilar evaluation conditions from one genome to another. For any given genome this implies that the measurenemt of its fitness, during the evaluation period, may be misleading, simply because of the lucky/unlucky starting conditions. To cope with such noise, we re-evaluate the champion (i.e., current best) genome with a probability $P_{reeavulate}$. This is, in effect, resampling as advocated by Beyer to deal with noisy fitness evaluations [1] and

it implies sharing the robot's time between producing and evaluating new genomes and re-evaluating old ones.
The fitness value that results from this re-evaluation could be used to refine a calculation of the average fitness of the given genome. However, we choose to overwrite the previous value instead. This may seem counterintuitive, but we argue that this works as a bias towards genomes with low variance in their performance. This makes sense as we prefer controllers with robust behaviour. It does, however, entail an intrinsic drawback as good genomes may be replaced by inferior, but lucky genomes in favourable but specific conditions. Then again, a lucky genome which is not good on average will not survive re-evaluation, avoiding the adaptive process getting stuck with a bad genome.

Algorithm 1 The (1+1)-ONLINE evolutionary algorithm.

for $evaluation = 0$ to N **do**
  if random() $< P_{reevaluate}$ **then**
    Recover(Champion)
    $Fitness_{Champion}$ = RunAndEvaluate(Champion)
  else
    Challenger = Champion + $N(0, \sigma)$ {Gaussian mutation}
    Recover(Challenger)
    $Fitness_{Challenger}$ = RunAndEvaluate(Challenger)
    if $Fitness_{Challenger} > Fitness_{Champion}$ **then**
      Champion = Challenger
      $Fitness_{Champion} = Fitness_{Challenger}$
      $\sigma = \sigma_{min}$
    else
      $\sigma = \sigma \cdot 2$
    end if
  end if
end for

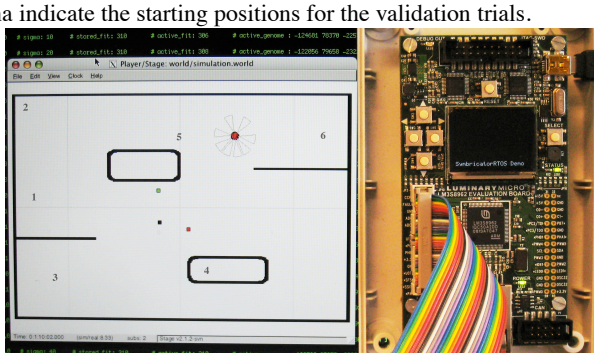
3 Experimental Setup

We evaluate the (1+1)-ONLINE algorithm in a set-up that features actual robotic hardware, a Cortex M3 board with 256kb memory. This controls a simulated autonomous robot in a Player/Stage[2] environment. Using the Cortex board instead of a fully simulated setup is due to administrative constraint in the project within which this research takes place: the Cortex board is the same hardware that is currently being integrated in the Symbrion robot prototypes and there is a strong emphasis on validation with similar hardware constraints. After N time-steps, the evaluation of the current controller is complete and the controller parameters are replaced with values from a new genome, which is evaluated *from the location the previous controller left it in*. This means that **no** human intervention is **ever** needed. We run the experiment 12 times.

[2] http://playerstage.sourceforge.net

Figure 1 illustrates the experimental set-up, with a Cortex board connected to the computer running Player/Stage. The simulated robot is modelled after an ePuck mobile robot with two wheels and eight proximity sensors. The maze environment used in our experiment is exactly as shown in this figure.

Fig. 1. The experimental setup: the Cortex board connected to Player/Stage. The numbers in the player-stage arena indicate the starting positions for the validation trials.



For each run of the experiment, the robot starts with a random genome and a random seed. The fitness function is inspired by a classic one, described in [7] which favours robots that are fast and go straight-ahead, which is of course in contradiction with a constrained environment, implying a trade-off between translational speed and obstacle avoidance. The following equation describes the fitness calculation:

$$fitness = \sum_{t=0}^{evalTime} (speed_{translational} * (1 - speed_{rotational}) * (1 - minSensorValue))$$

All values are normalised between 0 and 1. $minSensorValue$ is the value of the proximity sensor closest to any obstacle, normalised to $[0, 1]$ (i.e., the value decreases as an obstacle gets closer). We used the following settings during our experiments: both $recoveryTime$ and $evaluationTime$ are set to 30 time-steps, $P_{reeavulate}$ is set to 0.2, the σ initial value is set to 1 and may range from 0.01 up to a maximum of 4 and the gene values are defined to be in $[-4, +4]$.

It is important to note that this fitness function is used as a test function. Indeed, the current algorithm is by no mean limited to optimize collision avoidance. Relying on such a fitness function makes it possible to focus on the dynamics of the evolutionary algorithm with regards to desired properties.

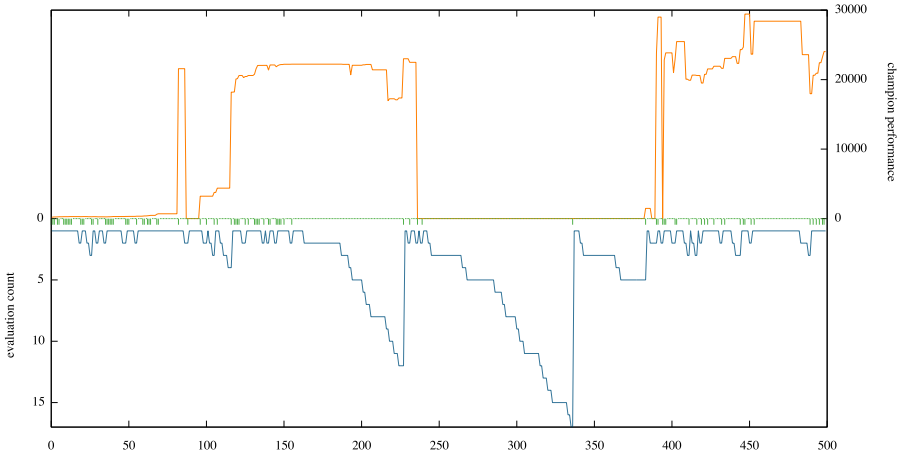
To provide an indication of the true performance and reusability of the best individuals found by (1+1)-ONLINE evolution, a hall-of-fame is computed during the course of evolution from the champions of all runs. The 10 best genomes from the hall-of-fame are validated by running each from six initial positions in the environment, indicated in figure 1. Starting from each of these positions, the genomes are evaluated for ten times the number of steps used for evaluation during evolution. Note, that one of the validation starting positions has certainly never been visited during development (test no.4, within a small enclosed area) and provides an extreme test case in a very constrained

environment. This decomposition into an evolution (development) phase and a post-experiment testing phase is similar to the learning and testing phases commonly seen in Machine Learning and does not imply a deployment phase as in traditional, off-line ER approaches.

4 Results

Evolution dynamics. We conducted a series of twelve independent experiments (1+1)-ONLINE evolution, with parameters set as stated above. Each experiment started with a different random controller (with very poor behaviour indeed) and a different random seed. The experiments ran for 500 evaluations and displayed different overall fitness dynamics with very similar patterns. Figure 2 shows typical statistics from one of those runs. Evaluations are denoted on the x-axis. The y-axis consists of two parts: the top half shows the fitness of the current champion genome. When a champion is re-evaluated very poorly or is replaced by an individual that upon re-evaluation turns out to be very bad, the fitness drops dramatically, as happens in this case after about 250 evaluations. The bottom half of the y-axis shows the number of (re-)evaluations of the current champion (downwards; the lower the line, the higher the number of re-evaluations). Every time the champion is re-evaluated, the line drops down a notch, until a new champion is found; then, the number of re-evalations is reset and the line jumps back to the top. The small vertical markers near the x-axis indicate whenever a new champion is adopted, i.e., when the challenger outperforms the current champion.

Fig. 2. Evolution dynamics of a typical run



By analysing the course of the evolutionary process for the experiments, we can observe important mechanisms such as local search (small continuous improvements in the fitness values due to nearby genomes), global search (the ability to get out of a neutral landscape or to jump from a local optimum to a different region), performance and robustness (the ability of certain genomes to display good performance *and* to remain champion even through re-evaluation).

Initially, performance is quite low, as is the number of re-evaluations; in effect, we are waiting for random search (σ is very high at this point) to bootstrap the adaptation. Then, after about 90 evaluations, an interesting individual is selected as champion that produces children that are even better. We observe a quick sequence of new, better performing individuals and an increasing fitness. After about 160 evaluations, we find that the champion has good fitness and is very robust: the individual survives many re-evaluations (the bottom line goes down) while displaying similar fitness values after successive re-evaluations.

During this period, σ steadily increases (as prescribed by the algorithm), causing mutation to become more and more aggressive, approaching random search. Eventually, this results in a challenger that beats the champion –either because this newcomer is actually very good or because it was lucky (favourable environmental conditions or taking advantage of an unfortunate re-evaluation of the champion). Observation during experiments showed that the latter option is more likely: at some point, the champion encounters a very difficult set-up and is re-evaluated as performing badly so that almost any challenger has a good chance of beating it. In the plot, this is exactly what happens at the precipitous drop in performance after 250 evaluations.

In all our experiments, we saw a similar pattern of initial random search characterised by many different genomes with poor fitness; then, local search characterised by subsequent genomes with increasing fitness until a robust genome is found that survives re-evaluation for some time and then a switch to another region that yields good results or towards an inferior genome that got lucky (almost a restart, in effect).

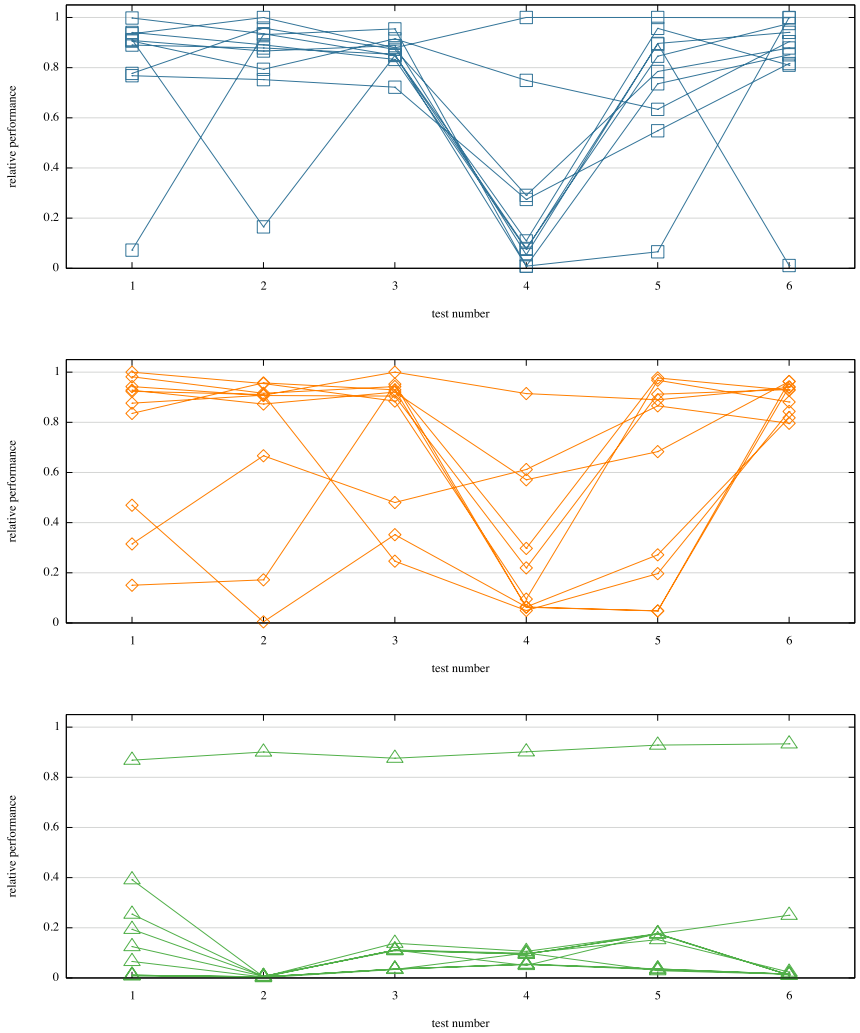
From the point of view of operational robot control such performance degradation may seem undesirable, but bear in mind that the (1+1)-ONLINE algorithm is meant as a *global* search algorithm. Therefore, such regular fitness reversals are a desired property as long as the search is slightly conservative around good individuals (as is evident from the lengthy episodes of re-evaluation in Figure 2). The regular re-evaluation of the champion promotes (in addition to the varying σ discussed below) global search; because of the noisy fitness calculation, if nothing else, it will occasionally be assessed as performing very poorly indeed. Such an occurrence provides an opportunity for a lucky new and possibly quite different genome to overthrow the champion.

Validation of the hall-of-fame. As described in section 3, a hall-of-fame was maintained during the course of the experiments for further validation of the -apparently- best genomes. Figure 3 shows the results of the validation of the hall-of-fame for the selected re-evaluation scheme (the champion's fitness is overwritten after every re-evaluation) and for two alternatives: one where the fitness is the average of all re-evaluations and one where there is no re-evaluation at all. This allows us to assess two things: whether high ranking genomes in the hall-of-fame are also efficient in a new set-up and whether the "overwrite fitness" re-evaluation scheme is relevant.

The y-axis shows the normalised performance: the best of all individuals for a scenario is set to 1.0, the performance of the other individuals is scaled accordingly. For each scenario (arranged along the x-axis), the graphs show a mark for each individual from the hall-of-fame. All results for a given genotype are linked together with a line.

The graph clearly shows that re-evaluation improves performance substantially; from the ten best solutions without re-evaluation, only a single one performs at a level

Fig. 3. Performance on validation scenarios for various re-evaluation schemes. Top: overwrite-last-fitness scheme ; Middle: average-fitness scheme ; Down: no re-evaluation scheme. X-axis shows the results on the six different validation setups (see fig.1), y-axis shows normalized fitness performance for each run. For a given genome, results in the six validation setups are joined together with a line.

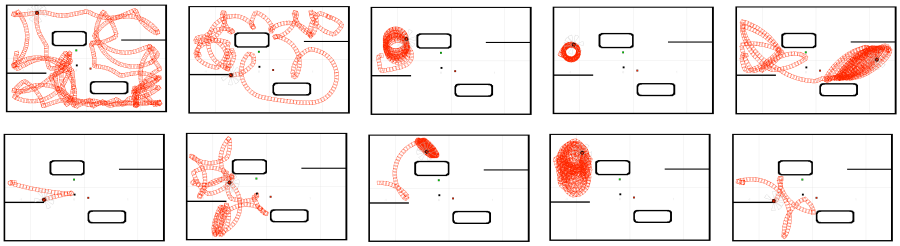


comparable to that of the ones with re-evaluation. The best individuals in the hall-of-fame for both re-evaluation variants are, on the whole, quite efficient; some come quite close to the maximum possible performance for these test cases (30,000). It is harder to distinguish between the performance of either variants: On the one hand, the spread of performance *seems* greater for the case with averaging fitness than it does for over-writing fitness, which would endorse the reasoning that overwriting after re-evaluation promotes individuals with high average fitness and low standard deviation. On the other hand, however, the nature of real world experiments have a negative impact on the amount of data available for statistically sound comparison of re-evaluation strategies,

as is often the case with real hardware, and keep from formulating a statistically sound comparaison. In particular, hardware contingencies implies strong constraints regarding time and human intervention, as robots should be re-located for each experiment and the Cortex board should be reloaded with the genome to be tested, as opposed to the completely autonomous setup during the evolution phase. Overall, the testing of ten genomes took approx. a full day of work with full investment from the experimenter [3].

Behavioural diversity. Further analysis of the ten best individuals with the overwrite-fitness re-evaluation scheme shows that the controllers actually display different kinds of behaviour –all good, robust, but different wall avoidance and/or open environment exploration strategies, ranging from cautious long turns (reducing the probability of encountering walls) to exploratory straight lines (improved fitness but more walls to deal with). Figure 4 illustrates this by showing the pathways of these individuals, starting from an initial position on the left of the environment. This reflects the genotypic diversity observed in the hall-of-fame and hints at the algorithm's capability to produce very different strategies with similar fitness.

Fig. 4. Traces for the ten best controllers (using fitness replacement after re-evaluation)

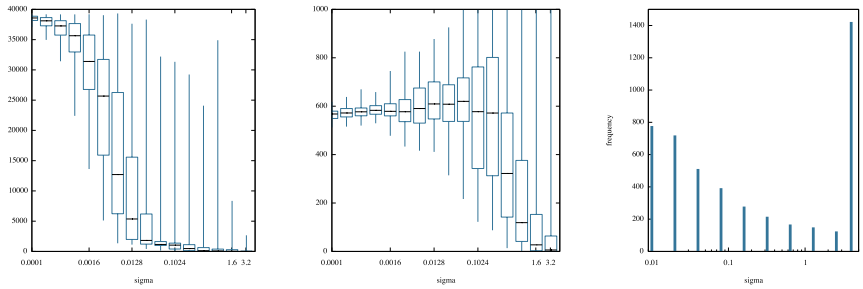


Strong causality and mutation. The reasoning behind the scheme to update σ relies on Strong Causality[12]: it only holds if small changes in the genome lead to small changes in behaviour and big changes in the genome to big changes in behaviour. To investigate if this property holds, a separate set of experiments was performed. For a range of σ values, 200 mutants were created from some fixed initial genome; every one of these mutants was then tested in our arena, from 193 different starting locations (homogeneously distributed over the environment), each with four orientations (i.e., a total of 772 tries per genome); each evaluation lasted 30 time-steps. Because such experiments using the Player/Stage and Cortex set-up as described above would require approx. 3.5 years to run, we used a simplified autonomous robot simulator.Each experiment started from one specific genome, the first experiment from the best genome in the hall-of-fame (Fig. 5.(a)) and the second experiment from a randomly generated genome (Fig. 5.(b)). In both figures, The x-axis shows σ. The y-axis shows the range of fitness values: the sum of fitnesses for each mutant over all 772 trials. For every value of σ, the candle bars show the minimum, maximum, median and lower and upper quartile. Figure (c) shows a histogram of the frequency of σ values over 12 runs (approximately

[3] To some extent, an illustrative metaphor is that of a biologist performing an experiment with mice.

4,700 evaluations) of the original experiment. The (logarithmic) x-axis shows occurring values for σ, ranging from 0.01 to 4. The count of occurrences is displayed along the y-axis.

Fig. 5. Strong causality experiments



(a) starting from one of the best genomes (b) from a random start (c) Incidence of σ values

Graphs (a) and (b) show that, as σ increases, the performance of mutated individuals becomes increasingly different; it actually covers the whole domain for medium to large values of σ. When starting from the 'best' genome, the average performance decreases as the mutations move further and further away from the original genome. From a randomly generated start point, performance changes either way as we move away from the original genome. This shows that there is strong causality: small changes in the genome (low σ) lead to small variations in fitness, and big changes lead to large variations. Finally, figure 5.(c) shows the density of σ values over the 12 original runs of the original experiment. The (logarithmic) x-axis shows occurring values for σ, ranging from 0.01 to 4, and the count of occurrences is displayed along the y-axis. As shown in the graph, all possible values of σ from very small (entailing local search) to large (global search) frequently occurred in the course of our experiments, with more occurences of both the minimum value (ie. local search) and maximum value (ie. global search). This provides a sound validation of the (1+1)-ONLINE algorithm ability to conduct both local and global search thanks to the self-updating σ.

5 Conclusions and Further Work

This paper provides a proof-of-concept for the viability of on-line, on-board evolution in autonomous robots. We have presented the (1+1)-ONLINE evolutionary algorithm to provide continuous adaptation in autonomous robots in unknown and/or changing environments, without help from any supervisor (human or otherwise). The (1+1)-ONLINE evolutionary algorithm is based on an "encapsulated" online onboard ER approach, as explained in the introduction. It was tested on very constrained, embedded hardware – the Cortex board we used in the experiments is limited in terms of performance as well as memory (256kb, including the operating system, complying with the robot prototype actually under construction). This requires a light-weight, low-complexity algorithm such as the one presented here, which is derived from the well known and well established $(1 + 1)$ evolution strategies.

One of the main contributions of the (1+1)-ONLINE evolutionary algorithm is that, by using re-evaluation, it specifically deals with intrinsically noisy performance evaluation in real world environments. This greatly increases the real-life applicability of this method and constitutes an original contribution compared to previous research into similar on-line setups. The second contribution is that of balancing local and global search through the σ update. Walker et al. described a similar $(1 + 1)-$ES inspired scheme with self-tuning σ [16] –however, the proposed approach updates σ through a heuristic that explicitly tunes the local and global search.

An on-line approach such as presented here tackles problems beyond the scope of traditional off-line ER, such as dealing with dynamic or unknown environments for which the robots could not be optimised before their deployment and it also addresses issues such as the reality gap and the lack of fidelity in simulation [2, 17]. In the experiments shown here, the (1+1)-ONLINE evolutionary algorithm yielded a great variety of behaviours in a limited amount of time; good performance was typically reached in under an hour. While the task at hand is relatively simple (obstacle avoidance and maximisation of translation speed), it should be noted once again that it requires no background knowledge whatsoever about the task and that the current algorithm can be applied in different contexts, simply by rewriting the fitness function.

The dynamics of evolution often result in the loss of a very good genome –this is actually desired behaviour of the algorithm as it ensures continued exploration of new or changed regions in the search space. It could, however, be interpreted as a complication from the engineer's viewpoint in a production environment where one wants to retain good genomes. This is in fact an instance of the well-known issue of exploration vs. exploitation in reinforcement learning; in this context, the algorithm proposed here provides the *exploration*. A reservoir such as the Hall-of-Fame introduced above can keep track of the best genomes and allow them to be re-used for exploitation.

Further research focusses on the following issues. Firstly, we consider alternative schemes to update the champion's fitness value after re-evaluation to combine the benefits of the "last fitness" and the "average fitness" approaches. This could for instance be achieved by weighting the influence of the latest fitness estimate and previous ones. Secondly, we intend to extend the algorithm towards a multi-robot set-up combining the distributed and encapsulated approaches (cf. Section1): adding a genome migration feature would make it possible to spread good genomes through a population of robots – similar to an island-based parallel EA. Thirdly, we are to test on-line, on-board evolution in a group of robots within dynamic environments - as a matter of fact, preliminary works with a real robot has already been done as an extension of this current work [5].

Acknowledgements

This work was made possible by the European Union FET Proactive Intiative: Pervasive Adaptation funding the Symbrion project under grant agreement 216342. We also thank Selmar Smit for his enthusiastic contribution to our discussions and Olivier Teytaud for discussions regarding the quasi-random number generator used on the Cortex board.

References

1. H.-G. Beyer. Evolutionary algorithms in noisy environments: theoretical issues and guidelines for practice. *Computer Methods in Applied Mechanics and Engineering*, 186(2-4):239 – 267, 2000.
2. R. A. Brooks. Intelligence without reason. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 569–595, Sydney, Australia, 1991. Morgan Kaufmann.
3. S. Elfwing, E. Uchibe, K. Doya, and H. Christensen. Biologically inspired embodied evolution of survival. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation IEEE Congress on Evolutionary Computation*, volume 3, pages 2210–2216, Edinburgh, UK, 2-5 September 2005. IEEE Press.
4. D. Floreano, P. Husbands, and S. Nolfi. Evolutionary robotics. In B. Siciliano and O. Khatib, editors, *Handbook of Robotics*, pages 1423–1451. Springer, 2008.
5. J.-M. Montanier and N. Bredeche. Embedde evolutionary robotics: The (1+1)-restart-online adaptation algorithm. In *IEEE IROS Workshop on Exploring new horizons in Evolutionary Design of Robots (Evoderob09)*, 2009.
6. U. Nehmzow. Physically embedded genetic algorithm learning in multi-robot scenarios: The pega algorithm. In *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.
7. S. Nolfi and D. Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. Cambridge, MA: MIT Press/Bradford Books, 2000.
8. S. Nolfi and D. Parisi. Auto-teaching: networks that develop their own teaching input. In *Free University of Brussels*. MIT Press, 1993.
9. S. Nolfi, D. Parisi, and J. L. Elman. Learning and evolution in neural networks. *Adapt. Behav.*, 3(1):5–28, 1994.
10. P. Nordin and W. Banzhaf. An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. *Adaptive Behavior*, 5(2):107–140, 1997.
11. A. L. F. Perez, G. Bittencourt, and M. Roisenberg. Embodied evolution with a new genetic programming variation algorithm. *icas*, 0:118–123, 2008.
12. I. Rechenberg. *Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien des Biologischen Evolution*. Fromman-Hozlboog Verlag, Stuttgart, 1973.
13. H.-P. Schwefel. *Numerical Optimisation of Computer Models*. Wiley, New York, 1981.
14. J. Urzelai and D. Floreano. Evolution of adaptive synapses: Robots with fast adaptive behavior in new environments. *Evol. Comput.*, 9(4):495–524, 2001.
15. 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.
16. J. H. Walker, S. M. Garrett, and M. 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.
17. R. A. Watson, S. G. Ficici, and J. B. Pollack. Embodied evolution: Distributing an evolutionary algorithm in a population of robots. *Robotics and Autonomous Systems*, 39(1):1–18, April 2002.
18. S. Wischmann, K. Stamm, and F. Wörgötter. Embodied evolution and learning: The neglected timing of maturation. In F. Almeida e Costa, editor, *Advances in Artificial Life: 9th European Conference on ArtificialLife*, volume 4648 of *Lecture Notes in Artificial Intelligence*, pages 284–293. Springer-Verlag, Lisbon, Portugal, September 10–14 2007.