Online Evolution in Dynamic Environments using Neural Networks in Autonomous Robots

Christopher Schwarzer
Nico K. Michiels
Institute for Evolution and Ecology
University of Tuebingen
Tuebingen, Germany
Email: Christopher.Schwarzer@uni-tuebingen.de

Florian Schlachter
Institute for Parallel and Distributed Systems
University of Stuttgart
Stuttgart, Germany
Email: Florian.Schlachter@ipvs.uni-stuttgart.de

***Abstract*—Online evolution is adaptation of agents while they are deployed in their task. The agents adapt autonomously and continuously to changing environmental conditions and new challenges. Such changes are also a topic in incremental evolution, where the difficulty of a task is gradually increased in an attempt to increase adaptation success. Here we investigate an online evolutionary process in simulated swarm robots using recurrent neural networks as controllers. In order to cope with dynamic environments, we present a distributed online evolutionary algorithm that uses structural evolution and adaptive fitness. Using an experiment about incremental evolution as a test case, we show that our approach is capable of adapting to a change that requires new recurrent connections.**

Keywords-online evolution; incremental evolution; recurrent neural networks; swarm robotics; evolutionary robotics.

I. Introduction

The design of adaptive robotic systems is a big challenge. Much research is being done to increase the flexibility of robot behaviour so they are able to adapt to changes in the environment. One big approach to solve this problem is evolutionary robotics, where the design of the robot controllers is driven by bio-inspired approaches [2], [3], [4]. Many of them are evolved offline on an external computer where the controller candidates are tested repeatedly with the same problem and the best solutions advance. After the controllers are optimized, the best ones are deployed to the robots in their actual task but the adaptive process is stopped. However in many environments, the conditions can change continuously and chaotically and it would be too inefficient or impossible to manually update the robots' behaviour. Such changes can be random or hard to predict and can make control structures obsolete or inefficient for the task. To deal with a dynamic environment, where the conditions of the task or even the task itself can change after the robotic system has been deployed, a process of continuous adaptation is needed that is running on the robots. Online evolution is such a process where the robots continuously evaluate their behaviour and change it to find improvements. A lot of research has been done in recent years about online evolution; it has been used successfully with neural networks [5] and it is often applied to robots [6], [7], [8].

One major aspect of evolution is that the evolutionary engine is flexible enough to adapt to a wide variety of changes. This is particularly important for online evolution because it would be ideal if our evolutionary algorithm and genome can adapt to a wide variety of a priori unknown situations. In offline evolution this is less of an issue as the entire evolutionary system can be tailored towards the known problem. Our main motivation is to come closer to the example given by natural evolution, by being able to create an evolutionary process in artificial agents that continuously evolve into increasingly sophisticated solutions. This can be on the organism level by evolving more complex organisms, for example multicellular robots, and also on species level by evolving different, interacting and coevolving robotic species [9].

In this paper, we investigate how online evolution can deal with a dynamic, changing environment. Our model system is swarm robots that are simulated in a 2D environment. We use artificial neural networks as robot controllers, which is a state-of-the-art approach for evolving robot behaviour [4]. In Section II, we give an overview of the state of the art in the evolution of artificial neural networks and incremental evolution. In Section III, we describe our proposal of an evolutionary algorithm that allows evolution of the genome structure. While having state-of-the-art capabilities, it has the novelty that it also runs online and distributed. Because there is no other comparable online and distributed approach, we use an experiment about incremental evolution to test our algorithm in Section IV. We show that our approach can effectively deal with a strong environmental change that requires structural evolution to achieve best performance. In an experiment that ends in a certain difficult environment, we compare treatments that have different intermediate steps and find no differences in the end performance. Section V provides a conclusion to our findings.

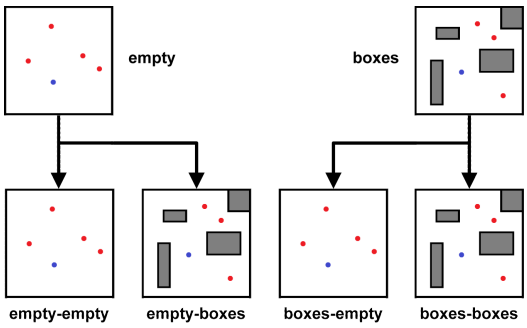
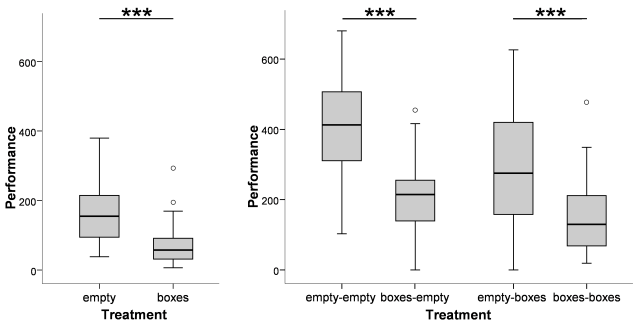


Figure 1. Experimental setup of our previous work [1] about incremental evolution with island evolution on a single robot. First populations evolved for 100 evaluations in two types of arenas, *empty* and *boxes*. Then arenas were changed (treatments *empty-boxes* and *boxes-empty*) or kept the same (*empty-empty* and *boxes-boxes*) for a second period of evolutionary adaptation with 100 evaluations.



(a) (b)

Figure 2. The collection performance at the end of each evolutionary phase of our previous experiment [1]. Shown is the summed performance of the last 10 evaluations ($n = 40$). **(a)** After the first phase, the performance is lower in the arena with boxes (*Wilcoxon test* $z = -5.2$ $p < 0.0001$). **(b)** After the second phase, the treatments that evolved first in the empty arena have a better final performance in the empty arena (*Wilcoxon test* $z = 5.6$ $p < 0.0001$) as well as in the boxes arena (*Wilcoxon test* $z = -3.9$ $p < 0.0001$).

II. Related Work

In several approaches, it has been shown that the evolution of neural networks can be improved by structural evolution of the networks. One of the early works in this field is the Generalized Acquisition of Recurrent Links (GNARL) [10]. In this work, they developed algorithms for the evolution of neural networks with recurrent links. The networks are randomly initialized (random hidden neurons and links) and evaluated. Afterwards, fifty percent of the population are allowed to create offspring (two children) for the next generation and so on. In the NeuroEvolution of Augmenting Topologies (NEAT) [11] the structural evolution starts with empty neural networks and develops over time. They also introduced a cross-over mechanism based on historic information and showed mechanisms for innovation protection (speciation). The improvements to the Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT) [12] extend the algorithms with a generative encoding and inclusion of sensors and output geometries [13].

The alternative to structural evolution is to use a fixed amount of structural genetic elements and just evolving connections between those elements. We call this here parameter evolution because the entire genome is a fixed set of parameters whose values are evolved. For example in a neural network, the neurons can be considered structural elements and the adjacency matrix of the network as the parameters. In this conceptual model, structural evolution does not only evolve the values of the parameters, but also the number of parameters. Consider that for a given problem a certain amount of structure is optimal: one that is large enough to be able to solve the problem but as small as possible to minimize search space. Thus for a known and static problem, a fixed approach will likely outperform structural evolution if the starting structure is optimal for this problem. However in dynamic environments, the optimal amount of structure is dynamic as well and possibly unpredictable. Structural evolution can then adaptively increase structural complexity to increase computational capabilities or reduce structure to reduce search space.

Structural evolution has one additional problem compared to parameter evolution; it complicates recombination. Structural, and functional, elements of two genomes must roughly match for recombination to be efficient, otherwise similar elements can be duplicated or omitted completely in the resulting recombinant genome. This can make recombination very disruptive and reduce overall offspring performance. An outstanding feature of NEAT is that it uses structural evolution and tackles this problem by tracking structural elements with innovation numbers. By comparing the innovation numbers of two networks, similar and dissimilar structural elements can be recognized and recombined accordingly. Furthermore, the recognition of similarity is used in forming speciation by only recombining individuals of certain similarity [11]. However, NEAT uses a central database that contains all known innovations and this database is needed for the matching algorithm. Thus NEAT cannot run truly distributed with a separate instance of the evolutionary algorithm running in each robot and with an exchange of genomes between instances. Because each instance would have its own innovation database, foreign genomes would contain innovations that are not known to this instance.

In this work, we propose an evolutionary algorithm that is comparable in features to NEAT, using structural evolution of neural networks and recombination based on network similarity, but it has the notable extension that it can run fully distributed and it is especially tailored for online evolution. Because there is no other comparable framework present that runs both online and distributed, we showcase the capabilities of our approach in several experiments and

especially in the context of incremental evolution.

With incremental evolution, the difficulty of an evolutionary challenge is gradually increased by introducing intermediate steps of relaxed difficulty. The theory is that evolution works better in smooth fitness landscapes [14]. For a population to evolve towards a challenging task, an increase in fitness must be possible within the neighbourhood of solutions that can be reached with the evolutionary operators like mutation and recombination. The more likely it is to reach a fitness increase, the quicker the evolutionary process can proceed. This is important for artificial evolution in robotics, where fitness evaluations are particularly costly on real hardware. An early work that used incremental evolution on a real robot was in 1994 by Harvey et al. [15].

There has been some work with different kinds of experiments that tried to show advantages of incremental evolution compared to direct evolution that has no intermediate steps but results have been mixed. For example, Gomez and Miikkulainen [16] evolved robots for a capture-prey scenario, where an agent has to capture a prey in a grid world and the evasiveness of the prey is gradually increased. They show that direct evolution could not solve the problem in the same time as incremental evolution. Similarly, Barlow et al. [17] found that it increases the chance to find a successful controller.

In our previous work [1], we have also found a positive effect of incremental evolution. In the experiment that was a precursor to the one described later in this work, we compared a plain empty environment with one with several large obstacles in a search and collection task. Populations were first adapted on one arena type and then arenas were changed as illustrated in Figure 1. In Figure 2, the collection performance at the end of each evolutionary phase is shown for 40 replicates. Notable is that the performance of the treatment *empty-boxes* is significantly higher than *boxes-boxes*. The incremental evolution treatment that first evolved in the empty arena adapted better to the boxes arena than the direct evolution treatment that spent the same total time in the boxes arena.

But there are also contradicting results like the one from Christensen and Dorigo [18], who experimented with the task of finding a light source while avoiding holes in the ground. They gradually increased the challenge by increasing the complexity of the fitness function and by adding more holes to the arena. They conclude that the incremental strategies do not perform better than direct evolution when given equal computational time.

Based on these mixed results, we want to provide more insight if incremental evolution is beneficial for evolving robot controllers. Because incremental evolution provides an environment with explicit changes, it also provides a dynamic environment to test our framework for distributed online evolution, which we present in the following section.

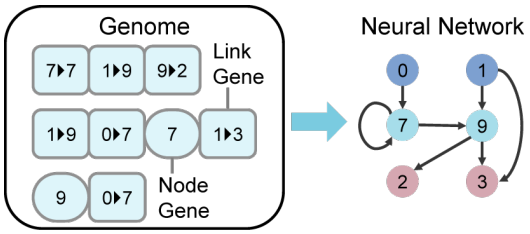


Figure 3. Example of a genome and its neural network. The genome is a set of link genes and node genes that produce the respective elements in the neural network. Input and output neurons are fixed and not part of the genome.

III. Distributed Online Evolution Framework

The genome of our evolutionary framework encodes a neural network as a set of genes with two types of genes: node genes and link genes. The neural network model is similar to NEAT: there are no layers and recurrent connections are allowed. We use a piecewise linear activation function with a variable bias value (Formula 1). The use of bias values replaces the bias neuron, common to many other neural network models. No learning mechanisms are employed.

$$\varphi^{pwl}(v) = \begin{cases} 1 & \text{if } v \geq 1 + b \\ v - b & \text{if } b < v < 1 + b \\ 0 & \text{if } v \leq b \end{cases} \quad (1)$$

A node gene contains an id and a bias value for a neuron. A link gene contains a source and destination neuron id and a link weight value. The first step of producing a neural network from the genome is creating the input and output neurons. These have fixed ids and parameters and are not part of the genome. Then the hidden neurons are created and finally the neural links between neurons. Each node gene produces one hidden neuron, using the id and bias values stored in each gene. In the same fashion, each link gene produces one neural link, making a connection between the source neuron to the destination neuron with the weight value of the gene. An example of a genome with the corresponding neural network is displayed in Figure 3.

The template of our online evolutionary algorithm is an $(\mu + 1)$ algorithm [19]. On each robot, there is a population of μ genomes and one extra genome is active, controlling the robot. One robot is considered an island population and one such algorithm instance is independent and unaffected by the other islands except of genome exchange between islands, called genome migration. An overview of the concepts of our approach is shown in Figure 4.

The island population of genomes serves as parental gene pool from which offspring genomes are created. First, one genome of the population is selected to be a parent (parent selection). Then, the parent may choose another genome from the population for recombination (mate selection). This

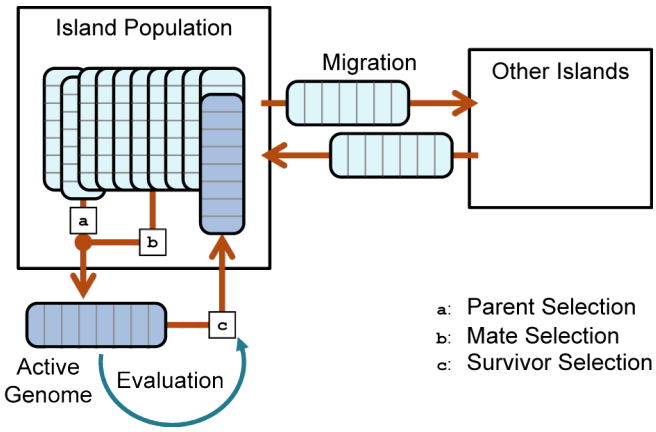


Figure 4. Overview of the distributed online evolution algorithm. It is based on a $(\mu + 1)$ algorithm with an island population of genomes in each robot and one active genome serving as controller. The active genome is an offspring genome of members of the island population.

Table I
KEY TERMS OF THE EVOLUTIONARY ALGORITHM

Term	Description
Island Population	Set of genomes within one robot. A population is initialized fully at start and the size is always constant.
Parent Selection	Uniform random selection of one genome of the population.
Mate Selection	Parent genome compares similarity with the other genomes of the population with its desired mate similarity. If there are genomes within a desired similarity window, one of those is randomly picked as mate.
Survivor Selection	Evaluated genome replaces the genome of the population with the lowest fitness score, if the performance score is better.
Migration	If two robots are in close proximity and they have not had a migration in a delay period, one random genome of each population is exchanged with the other.
Evaluation	Offspring genome is active and controls the robot for a fixed period of time. During this time it accumulates a performance score.
Fitness Score	A combination of a genome's performance score and the performance scores of its offspring.

selection is based on the similarity of the parent genome and the potential mates. If a suitable mate is found, a recombinant genome is produced from the two parents and the recombinant is mutated. Otherwise, a clone of the single parent undergoes mutation instead. The parent genomes are not modified. The resulting offspring genome is set as active genome and a neural network is produced from the genome, acting as controller of the robot for a fixed amount of time. This time is the evaluation period of the offspring genome and a performance score is accumulated. After the evaluation period has elapsed, a decision is made whether a member of the population is replaced or if the evaluatee is discarded (survivor selection). At this point, the cycle of online evolution starts anew by picking a parent and producing an offspring genome to control the robot.

Asynchronously to this cycle, the population of a robot can change by migration with other robot populations. For this process, we assume range limited communication mechanisms of real robots like infrared communication. When two robots are in close proximity and no migration has happened within a grace period, one random genome of each population is exchanged with the other. An overview of terms of our evolutionary algorithm and their implementation details is given in Table I.

The fitness score f of a population member x is calculated as the weighted average between its original performance score s_x and the performance scores of all its offspring $O(x)$ according to Formula 2. The weight w of an offspring is 0.5 for recombinant offspring and 1.0 otherwise.

$$f(x) = \frac{2\ s_x + \sum^{i \in O(x)} w_i\ s_i}{2 + \sum^{i \in O(x)} w_i} \qquad (2)$$

The combination of offspring performance with original performance of an individual is a major source of the adaptiveness of our approach. It can be considered as an adaptive fitness function because the comparative fitness of an individual changes as more offspring is produced and evaluated. If the environment changes and the individual becomes maladapted, its average offspring performance drops and so does the individual's fitness score. In this way, once dominating individuals can be purged from the system if they become maladapted in a changing environment. A second effect of this approach deals with the inherent error of performance evaluations in online evolution. A single evaluation can be affected heavily by chance as the current situation of the dynamic environment can vary heavily in difficulty. One way to tackle this problem is by performing repeated evaluations of the same individual to reduce the error [8]. However, these re-evaluations cost time and in our approach every genome is only evaluated a single time. We argue that offspring performance is highly correlated to parent performance and thus offspring evaluations are in fact partial re-evaluations of the parent.

The mutation operator is implemented for each gene type. There is a probability of 0.2 per gene for a point mutation changing the gene itself and a separate probability of 0.2 for making a structural mutation. The point mutation of a link gene changes the link weight by applying a uniform random change in the range from -0.2 to 0.2. In the same way, the bias value of a node gene is mutated. Structural mutation of a link gene cane either delete the gene or produce a new link gene with random link weight, source and destination neurons. Structural mutation of a node gene can also either delete the gene or produce a new node gene with a random identifier and random bias value. Deleting a node does not remove link genes that connect to this neuron. Such dangling

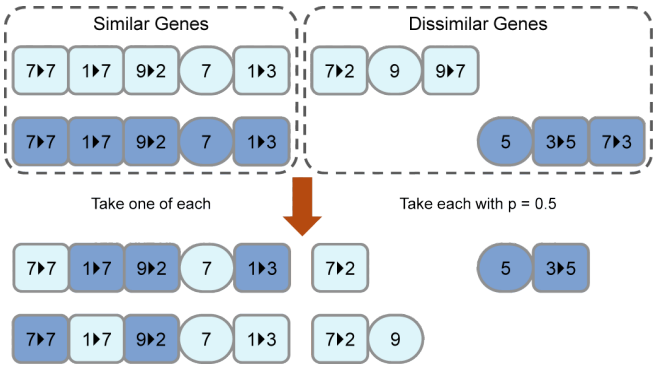


Figure 5. Example of genome recombination. The genes of two genomes are grouped in a set of similar genes, that are paired up, and a set of dissimilar genes. Two exemplary results of recombination are shown.

links simply have no effect in the resulting neural network. Structural mutation for dangling link genes reconnects them with a random neuron.

An important feature of our genome is that the similarity can be calculated between any two genomes without the need for a common gene database. A link gene is similar to another link gene if it has the same source and destination neuron ids. A neuron gene is similar to another neuron gene if it has the same identifier. The similarity s between two genomes A and B is calculated using the number of similar genes n_s divided by the sum of the number of genes n of the two genomes as shown in Formula 3. The resulting value is between 0.0 for no similarity and 1.0 for high similarity.

$$s(A,B) = \frac{2\, n_s(A,B)}{n(A) + n(B)} \tag{3}$$

Recombination relies on the similarity mechanism. When two genomes are recombined, the sets of genes can be split into similar genes on both genomes and extra genes that are unique to either one genome. Figure 5 illustrates how recombination proceeds with the similar and dissimilar genes of each genome. The similar genes are paired up and of each pair one random gene is picked for the recombinant. Of the dissimilar genes, each gene has a probability of 0.5 of being picked. This is a balanced recombination operator because on average it does not increase or decrease the amount of genetic material in contrast to other strategies, for example taking all dissimilar genes. The similar genes can be seen as homologous genes that are matched and recombined. Although similar, they can still differ in their values, link weight or neuron bias, and those differences are recombined between the parents. The extra genes are structural differences between the genomes that cannot be matched and thus an offspring can have any subset of those genes.

The key to this recombination operator is the use of random identifiers for the neurons of the neural network. A detection of homologous structures of the network is actually not performed but the random neuron identifiers are used as a heuristic. Identifiers for hidden neurons are random upon creation of a new hidden neuron in structural mutation of a neural network. Offspring inherit these identifiers from the parents and thus, the identifiers are an indication of common ancestry: Two genomes that share a lot of identifiers are very likely to have a common ancestor and thus structures with the same identifiers are likely to have similar functions. Although it is possible that the same identifier is created in another context in another genome the probability of such a collision is so low that the system is overall not disturbed and colliding identifiers will be sorted out by selection. In fact, our experiments are performed with a much elevated identifier collision probability by using only 1000 identifiers. The problem of false positive matching is further reduced by restricting recombination to genomes that have a certain minimal similarity during mate selection.

IV. Experiments

To show the capabilities of our online evolutionary algorithm, we tested it in two related experiments. The first focuses on comparing the features of the algorithm itself and to understand the complexity of the scenario. The second experiment uses the results from the first one and a similar setup to make an experimental comparison of incremental evolution.

The experiments are run in a simulation environment based on an open source 2D physics engine (Farseer Physics [20]). The robot model approximates the capabilities of a small swarm robot like Jasmine [21] or Wanda [22] and 50 time steps (ticks) in the simulation approximates one real time second.

Both experiments use an exploration and foraging scenario, using a small group of four robots in an arena with ten power stations that can supply energy to the robots. When a robot is in close proximity to a power station it is charged and gains one performance score point every 25 ticks. However, a power station has only limited supply of 20 power units and runs dry while charging a robot. It does recharge its power supply slowly at one power unit every 125 ticks but only when no robot is nearby. This prevents a sessile behaviour that robots just stay close to one power station. When multiple robots are near the same power station only the one that approached it first is charged.

The robots are equipped with a virtual vision sensor that lets them detect colours and distance sensors to detect obstacles in a forward facing arc. Robots appear blue while charged power stations blink between red and black, depleted power stations are constant black. Walls of the arena have black and green colours to make navigation potentially possible for the robots. Figure 6 illustrates a snapshot of the arena of this experiment.

For statistical analysis we use JMP [23], Version 9.0.0, and for model fitting we use R [24], Version 2.13.1.

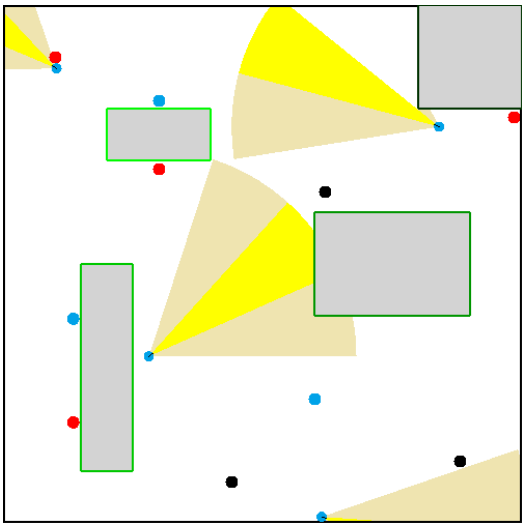


Figure 6. Snapshot of the simulation experiment. In the arena are four blue coloured robots with the view area of their sensors shown and ten power stations. The colour of the power stations is alternating between black and either red or blue. At this point, half the power stations blink in red, the other half in blue. The arena is also occupied by some larger obstacles with walls in different shades of green.

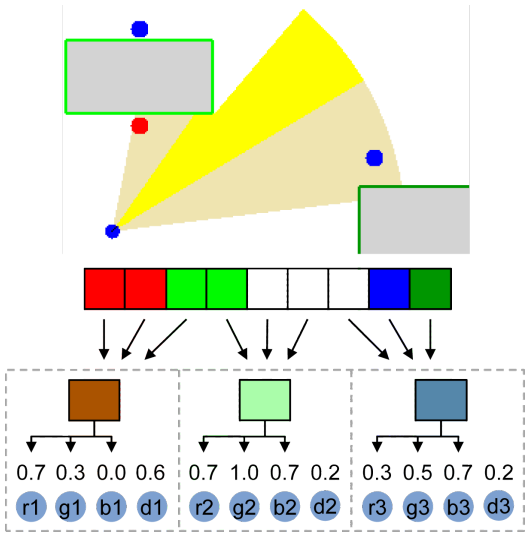


Figure 7. Sensor model of the simulated robot. In the 2D simulation environment, a 2D-to-1D perspective projection is used to create an array of nine RGB colour values. These values are combined into three averaged values for left, middle and right view area. Each RGB channel of those three colour values is fed into one input neuron of the neural network in addition to three proximity values of simulated distance sensors.

A. *Structural Evolution*

In this experiment, there is one large change in the environment that requires a complex adaptation and we compare different mechanisms of the evolutionary algorithm. Our hypothesis is that in face of a complex adaptation challenge, structural evolution and recombination are beneficial. We define here a complex challenge as one where hidden neurons and recurrent connections are needed and, thus, a perceptron without hidden neurons should perform significantly worse.

The evolutionary challenge for the robots is a change in the appearance of the power stations. The experiment starts with a genome population that is adapted to collecting power stations that blink in red (the colour alternates between black for 5 ticks and red for 5 ticks). The ten power stations start with red blinking and every 100.000 ticks, one power station changes its appearance to blinking in blue (same frequency). During the blue phase such power stations appear identical to other robots to the colour vision sensors of a robot and during the black phase they appear like walls and depleted power stations. Neural networks must perform temporal sensor fusion to detect the alternation between black and blue. They must infer the blinking to recognize a blue blinking power stations and distinguish them from the robots and walls. After one million ticks, all power stations blink in blue. The experiment continues for another one million ticks without further changes for a total of two million ticks.

Each robot is controlled by an artificial neural network produced by our evolutionary framework described in Section III. The neural network performs five update steps at each simulation time tick. There are twelve input neurons (three colour sensors each with a red green and blue channel and three proximity sensors) and two output neurons to control the differential drive of the robot. All inputs are mapped to values from 0 to 1, the output neurons provide values from -1 to 1.

The colour vision sensors are simulated using a 2D-to-1D perspective projection in the 2D simulation environment with an opening angle of 72° which returns an array of nine colour pixels. Every three of these nine pixels are averaged into three colour values. Each RGB colour channel of these values is fed into one input neuron of the neural network. A visual example of this procedure is given in Figure 7. The result is a simple colour vision input for the neural network with three virtual colour sensors, fanned out in a forward facing arc. A comparable sensor is feasible on actual robots using RGB sensors or downsampling a camera image. The simulated robot is also equipped with three proximity sensors, one facing ahead, one 24° to the right and one 24° to the left. These sensors behave similar to infrared proximity sensors.

Three factors of the evolutionary process are investigated:

- **Network type:** A perceptron with no hidden layer (*P*) and a recurrent network with hidden neurons (*H*).
- **Structural evolution:** Enables mutation to delete and create genes for both links and neurons of the network. Note that here, the perceptrons never evolve hidden neurons but can still add an remove links with structural evolution. Without structural evolution, only link weights and bias values of existing genes are mutated.

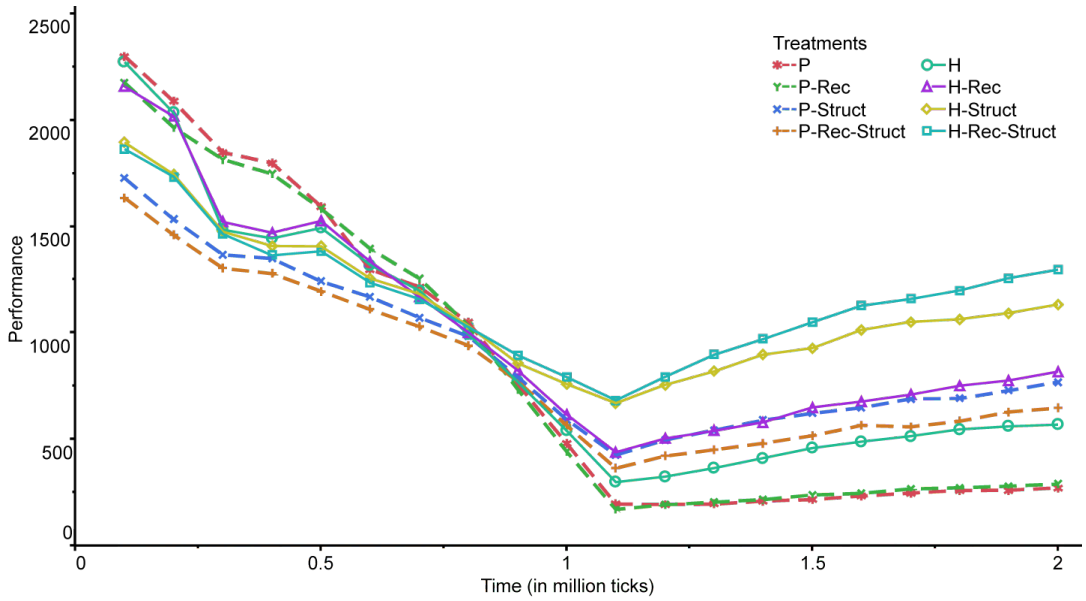


Figure 8. The development of foraging performance over time for each treatment. Shown is the average of 50 replicates, standard error is omitted for clarity. Dashed lines belong to the perceptron treatments (*P*), solid lines to the treatments with hidden neurons (*H*). The performance overall drops until 1 million ticks as the power stations change to blue blinking appearance. Some treatments are better able to adapt and recover from this change than others.

- **Recombination:** Enables mate selection and recombination in the evolutionary algorithm. Genomes try to find another genome that falls within a certain similarity window for producing a recombinant offspring. Without recombination, all offspring is mutated clones.

With these three factors, each with two levels, we did a full factorial setup for a total of eight treatments with 50 replicates. For preparation, initial populations of random perceptrons and random recurrent networks with five hidden neurons were evolved for three million ticks to adapt to the red blinking power stations. Of those runs, one population of recurrent networks and one population of perceptrons were selected for the experiment proper. Both populations have the same performance and the highest performing pair was selected out of 10 runs.

The response variable of the experiment is the total foraging performance of all four robots over a time period of 100.000 ticks. This foraging performance is the sum of all performance score points dispensed by all power stations in that period. With an evaluation time of the evolutionary algorithm of 2.500 ticks per individual, 160 individual evaluations contribute to this value. This measure represents the effective system performance and underlines the online nature of the scenario because every single evaluation contributes to the system performance rather than few peak performing individuals.

B. Results

The development over time of the average performance of the replicates is shown in Figure 8. Every treatment is affected by the environmental change and the performance drops. The perceptron treatments without structural evolution drop the lowest and are unable adapt much to the change. The treatments with hidden neurons and structural evolution do not drop as low, show a clear recovery and adapt better to the new situation.

Figure 9 presents the performance of each treatment at the end of the experiment. Confirming our expecetations, the treatments with perceptrons (*P*) perform worse than the ones with hidden neurons at every treatment. The treatment with the highest final performance uses hidden neurons, recombination and structural mutation (*H-Rec-Struct*), though the difference to the next best treatment *H-Rec* is slim and the statistical difference only borderline significant. Though, these two treatments are significantly different and superior to all others.

We fitted a general linear model to the endpoint performance results. After data exploration, we used log-transformed performance values to equalize distributions of residuals and two outliers with a value of 0.0 were excluded. Starting with a full-factorial model, the three-way interaction was removed, lowering the AIC value. Our final model can be seen in Table II. The residuals of this model appear linear in a Q-Q plot and can be considered normal distributed. The biggest influencing factors are network type and structural evolution (estimates of 0.754 and 1.07 respectively) with smaller interactions between all factors. Recombination as primary factor has no influence but in interaction with the hidden network type it acts positively (0.317) and slightly negatively together with structural evolution (-0.186).

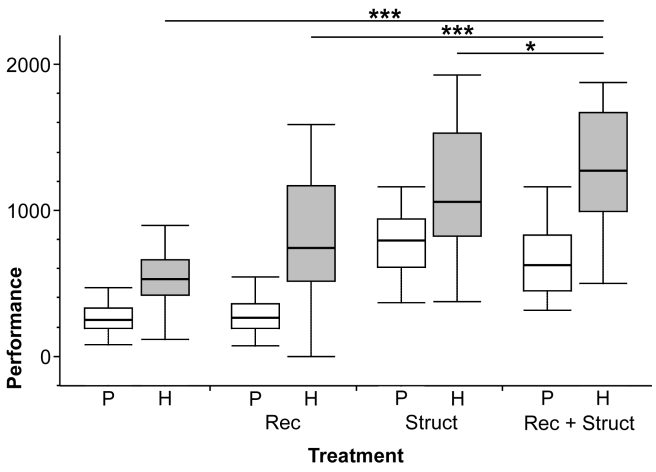


Figure 9. The performance endpoints for each treatment shown as boxplots with the median as centreline, the box ranging from the 25% to the 75% percentile and the whiskers marking minimum and maximum values of the replicates ($n = 50$). Treatment *H-Rec-Struct* has the highest performance, being slightly better than *H-Struct* (*Wilcoxon test* $z = 2.0$ $p = 0.045$) and significantly better than *H-Rec* (*Wilcoxon test* $z = 5.0$ $p < 0.0001$) and *H* (*Wilcoxon test* $z = 7.6$ $p < 0.0001$).

Table II
GENERAL LINEAR MODEL

	Estimate	SE	t	p
(Intercept)	5.521	0.056	97.82	$<$ **0.001**
Network	0.754	0.074	10.16	$<$ **0.001**
Struct	1.07	0.074	14.47	$<$ **0.001**
Rec	0.012	0.074	0.872	0.16
Network:Struct	-0.374	0.086	-4.37	$<$ **0.001**
Network:Rec	0.317	0.086	3.70	$<$ **0.001**
Struct:Rec	-0.186	0.086	-2.17	**0.03**
Adj. R^2	0.627			
F-statistic	112.2 on 6 and 391 DF			
p	$<$ **0.001**			
AIC	460.19			

These results show that our distributed online evolutionary algorithm is capable of adapting to a different environment. The selection mechanisms tolerated a general drop in population performance induced by the change. When offspring was produced that pioneered in dealing with the new environment, it could spread in the population and replace former champions although the performance level was lower than earlier in the evolutionary run.

Furthermore, we can deduce that the evolutionary challenge of the experiment is complex as indicated by the worse performance of the perceptrons. They had the same performance in the starting environment, foraging red blinking power stations, but are worse than the other treatments after the power stations changed to blue blinking. Thus, new neural interconnections and neural structures are required to adapt. This is also confirmed by our general linear model where structural evolution is the biggest factor to achieve high performance.

Regarding our recombination mechanism, our expectations were not fully met though we have weak empirical evidence for a small benefit in the best performing treatment. Some observations can be drawn from our results and our general linear model. The perceptrons did not benefit from recombination unlike the recurrent networks. *P-Rec* has the same performance as *P* and *P-Rec-Struct* is even worse than *P-Struct* while *H-Rec* is much better than *H* and *H-Rec-Struct* is slightly better than *H-Struct*. We think that the recurrent networks can use recombination as a makeshift structural mutation — by recombining their structural diversities into new solutions. This structural diversity can come from junk genes: genes for neural structures that were disconnected and then later reintegrated by recombination. The perceptrons do not have such junk genes because without hidden neurons all link genes are connected to output neurons and thus active.

C. Incremental Evolution

The previous experiment showed that the scenario has a certain complexity and our evolutionary framework can solve it using its features of structural evolution. With those findings, we performed a second experiment with the same scenario where the evolutionary algorithm is fixed and the change of the environment is varied instead. This serves to illustrate how our framework deals with different changes and is also an experiment about incremental evolution. The hypothesis is that intermediate steps towards an adaptive challenge lead to faster evolution than direct evolution.

In this experiment, the algorithm uses hidden neurons, structural evolution and recombination. Each run begins with an initial population of random networks with five hidden neurons and every possible neural connection is present. Neural connections have a random weight, hidden neurons a random bias value. The end conditions of the scenario are the same as in the previous experiment, blue blinking power stations. However, the conditions during the run are different among the treatments.

We decomposed the task of detecting a blue blinking signal into intermediate steps with different appearances of the power stations.

1) Red shining: A constant red colour. This is a unique sensor signal in the arena because nothing else gives a signal on the red colour channel. Simple neural networks can detect this.
2) Red blinking: An alternation of colour between five ticks of black and five ticks of red. Still a unique signal but the blinking requires some compensation in the network.
3) Blue blinking: Same as red blinking but with the colour blue instead of red. Blue is not a unique colour signal. The robots have the same blue colour appearance but do not blink.

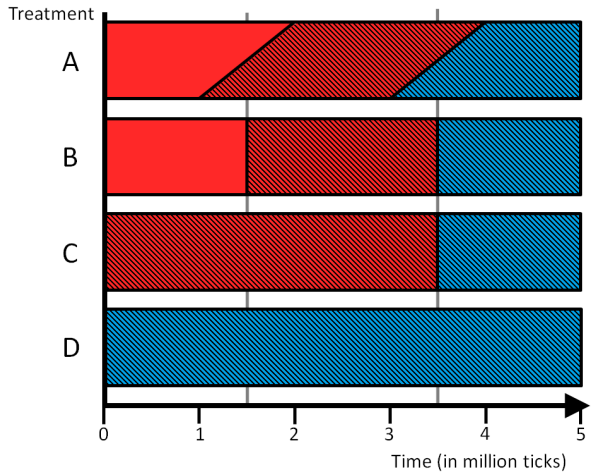


Figure 10. The four treatments of the incremental evolution experiment. *A:* Starting with red shining, gradual transition to red blinking, followed by another gradual transition to blue blinking. *B:* Same as *A* but the transitions happen instantly. *C:* Starting with red blinking and only one instant transition to blue blinking. *D:* Starting with blue blinking directly without any transitions.

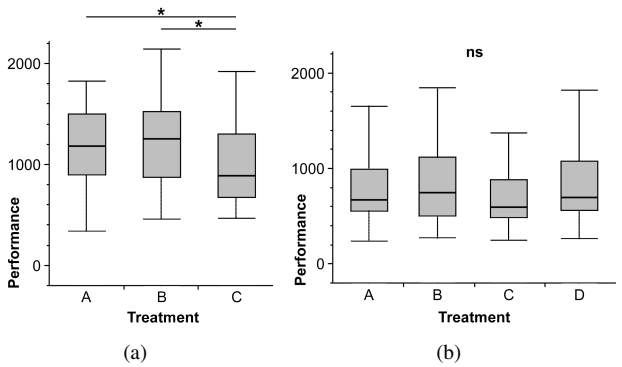


Figure 11. The performance of the incremental evolution experiment at 3 million ticks and 5 million ticks ($n = 50$). Treatment D is omitted at 3 million ticks because it is not comparable. **(a)** At 3 million ticks, treatments A and B have the same performance (*Wilcoxon test* $z = 0.6$ $p = 0.556$) but C is significantly lower than A (*Wilcoxon test* $z = -2.1$ $p = 0.036$) and than B (*Wilcoxon test* $z = -2.5$ $p = 0.011$). **(b)** At 5 million ticks there are not differences between the treatments (*Kruskal-Wallis test* $\chi^2 = 3.1$ $p = 0.380$).

This experiment has four treatments that are illustrated in Figure 10. Each treatment has a total of five million ticks to adapt from a random starting population to harvest the blue blinking power stations at the end.

- **A:** Starting with red shining, after 1 million ticks the power stations change one after the other to red blinking. The transition is complete after 2 million ticks. At 3 million ticks, there is another gradual transition from red blinking to blue blinking, which is finished after 4 million ticks. This is the treatment with the most increments.
- **B:** This starts similar to *A* with red shining but all power stations change to red blinking at 1.5 million ticks simultaneously. At 3.5 million ticks happens the simultaneous change to blue blinking.
- **C:** Here the phase of red shining is omitted and it starts with red blinking right away. The change to blue blinking happens at 3.5 million ticks like in *B*. This treatment serves also as comparison to direct evolution to the red blinking environment.
- **D:** The appearance of the power stations is blue blinking from the start with no changes in the environment. This is the direct evolution treatment for the blue blinking environment.

It is expected that the final performance is higher in the treatments with more increments: treatment *A* having highest performance and *D* lowest. For the treatments *A*, *B* and *C* we can make a similar comparison at 3 million ticks for adaptation to the red blinking environment.

D. Results

Surprisingly, the performance at the end of the 5 million ticks is the same across all treatments (Figure 11(b)). There are no significant differences between the direct evolution treatment *D* and the the incremental evolution treatments. However, a slightly different picture can be seen for the red blinking environment at 3 million ticks (Figure 11(a)). Here, treatment *C* is the direct evolution treatment with significantly lower performance than treatments *A* and *B*, though there is no difference between them.

An overview of the development of mean performance over time of all four treatments is displayed in Figure 12. The treatments with intermediate steps climb to high values during their relaxed conditions but then drop back when the environment changes. When comparing treatment *A* and *B*, we see that the gradual transitions of *A* carry no benefit and the instant transitions of *B* result in the same performance at 2 and 4 million ticks respectively. However, we see in comparison with *C* that they are able to leverage some advantage of first evolving with red shining power stations to the next phase of red blinking power stations, resulting in an increased performance at 3 million ticks. The transition around 4 million ticks to blue blinking affects treatments *A*, *B* and *C* equally hard. They all drop below the baseline of the direct evolution treatment *D* but recover quickly and reach the same performance level of *D* by the end of the experiment.

These results show that incremental evolution is not universally beneficial and give support to the conclusion of Christensen and Dorigo [18]. Each previous works on incremental evolution uses different experimental scenarios with a different approach to realizing the increments, which is a likely explanation for the mixed results. In our own previous experiment [1], we used different arenas and found

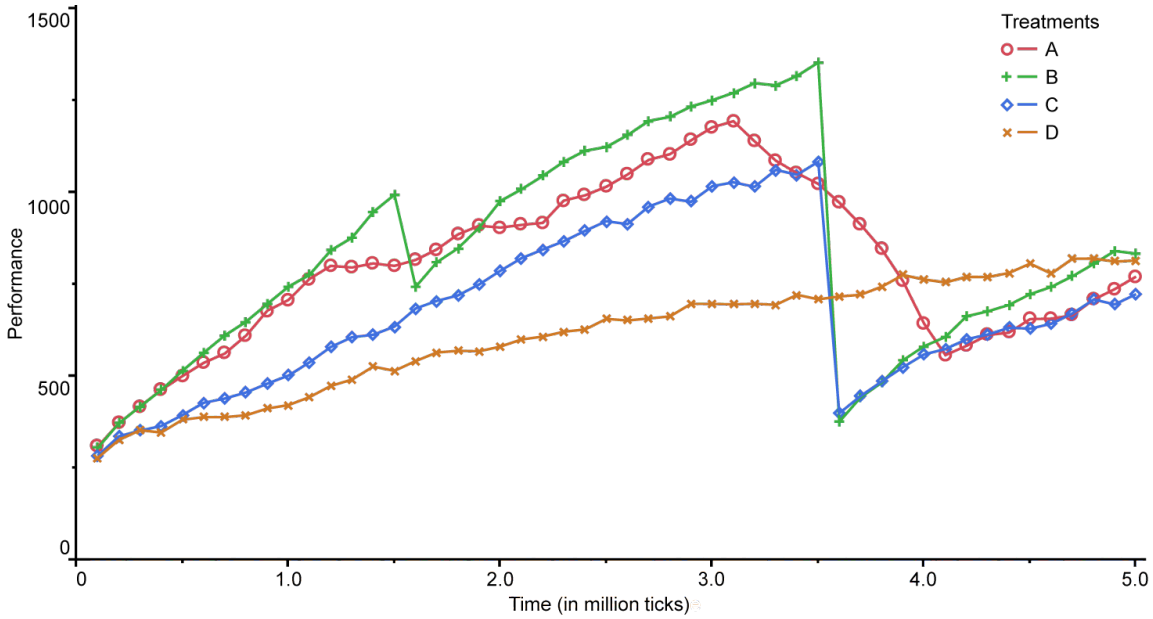


Figure 12. Development over time of the foraging performance of each treatment. Shown is the mean performance of 50 replicates. Treatments with relaxed intermediate steps reach temporarily higher values but are mostly unable to carry this advantage through when the environment changes. The transitions cause large drops in performance but it catches up quickly with the baseline of direct evolution.

benefits. Whereas here, we changed the appearance of the targets instead and could not see benefits. From this, we infer that incremental evolution is only beneficial in certain cases, depending on the design of the evolutionary increments.

V. Conclusion

We have presented a new framework for artificial evolution which has an unprecedented combination of features. It does state-of-the-art structural evolution of neural networks, including recombination of related neural structures and it adds the capability for online and distributed operation. One key element to achieve this is to use random, inheritable identifiers in the genome structure, making it possible to match structures of common ancestry.

The described experiments illustrate these features. Using our framework, we have shown that a simulated swarm of robots evolved online to solve a complex task. The evolved networks are able to distinguish an alternating signal from a constant signal, which is a form of sensor fusion over time that requires recurrent connections. It was also demonstrated that our approach can effectively evolve online in dynamic environments. In particular, drops in overall fitness levels due to changed conditions are tolerated by updating the fitness score of former champions with new evaluations of their offspring.

Our work has provided an effective solution for enabling artificial agents to adapt in a dynamic environment. While our approach deals well with differences in the environment over time, more work can be done with a varied environment where different solutions are possible at the same time. In a complex environment, there are multiple ways to approach a problem and there are even several different problems at the same time. Future work in artificial evolution needs to better support niching of the population so solutions can branch out to adapt to different problems in different ways. This brings us closer to our vision of developing artificial evolutionary processes that can produce a diversity, complexity and flexibility like natural evolution.

Acknowledgment

The “SYMBRION” project is funded by the European Commission within the work programme “Future and Emergent Technologies Proactive” under the grant agreement no. 216342. The “REPLICATOR” project is funded within the work programme “Cognitive Systems, Interaction, Robotics” under the grant agreement no. 216240. Additionally, we want to thank Ralph Dobler and Dennis Sprenger for fruitful discussions.

References

[1] F. Schlachter, C. Schwarzer, S. Kernbach, N. K. Michiels, and P. Levi, “Incremental online evolution and adaptation of neural networks for robot control in dynamic environments,” in *ADAPTIVE: Conference on Adaptive and Self-Adaptive Systems and Applications*, 2010, pp. 111–116.

[2] S. Nolfi and D. Floreano, *Evolutionary Robotics: The Biology, Intelligence, and Technology*. MIT Press, 2000.

[3] D. Floreano and C. Mattiussi, *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies*. MIT Press, 2008.

[4] D. Floreano and L. Keller, "Evolution of adaptive behaviour in robots by means of darwinian selection," *PLoS Biology*, vol. 8, no. 1, January 2010.

[5] A. Agogino, K. Stanley, and R. Miikkulainen, "Online interactive neuro-evolution," *Neural Process. Lett.*, vol. 11, no. 1, pp. 29–38, 2000.

[6] J. Walker, S. Garrett, and M. Wilson, "Evolving controllers for real robots: A survey of the literature," *Adaptive Behavior*, vol. 11, no. 3, pp. 179–203, September 2003.

[7] J.-M. Montanier and N. Bredeche, "Embedded Evolutionary Robotics: The (1+1)-Restart-Online Adaptation Algorithm," in *Workshop on Exploring new horizons in Evolutionary Design of Robots at IROS 2009*, 2009, pp. 37–43.

[8] N. Bredeche, E. Haasdijk, and A. Eiben, "On-line, on-board evolution of robot controllers," in *Artifical Evolution*, ser. Lecture Notes in Computer Science. Springer, 2010, vol. 5975, pp. 110–121.

[9] C. Schwarzer, C. Hösler, and N. Michiels, "Artificial sexuality and reproduction of robot organisms," in *Symbiotic Multi-Robot Organisms: Reliability, Adaptability, Evolution*, P. Levi and S. Kernbach, Eds. Springer-Verlag, 2010, pp. 389–408.

[10] P. J. Angeline, G. M. Saunders, and J. P. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 54–65, January 1994.

[11] K. O. Stanley and R. Miikkulainen, "Evolving neural network through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[12] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, "A hypercube-based encoding for evolving large-scale neural networks," *Artificial Life*, vol. 15, no. 2, pp. 185–212, 2009.

[13] D. B. D'Ambrosio and K. O. Stanley, "A novel generative encoding for exploiting neural network sensor and output geometry," in *GECCO: Conference on Genetic and Evolutionary Computation*. ACM, 2007, pp. 974–981.

[14] P. Stadler and S. Institute, "Towards a theory of landscapes," in *Complex Systems and Binary Networks*, ser. Lecture Notes in Physics. Springer, 1995, vol. 461-461, pp. 78–163.

[15] I. Harvey, P. Husbands, and D. Cliff, "Seeing the light: artificial evolution, real vision," in *Conference on Simulation of adaptive behavior: From Animals to Animats*. MIT Press, 1994, pp. 392–401.

[16] F. Gomez and R. Miikkulainen, "Incremental evolution of complex general behavior," *Adaptive Behavior*, vol. 5, pp. 5–317, 1996.

[17] G. J. Barlow, C. K. Oh, and E. Grant, "Incremental evolution of autonomous controllers for unmanned aerial vehicles using multi-objective genetic programming," in *CIS: Conference on Cybernetics and Intelligent Systems*, December 2004, pp. 688–693.

[18] A. L. Christensen and M. Dorigo, "Incremental evolution of robot controllers for a highly integrated task," in *SAB: Conference on the Simulation of Adaptive Behavior*, 2006, pp. 473–484.

[19] A. Eiben, E. Haasdijk, and N. Bredeche, "Embodied, on-line, on-board evolution for autonmous robotics," in *Symbiotic Multi-Robot Organisms: Reliability, Adaptability, Evolution*, P. Levi and S. Kernbach, Eds. Springer-Verlag, 2010, pp. 367–388.

[20] (2011, Aug.) Farseer physics engine. [Online]. Available: http://farseerphysics.codeplex.com/

[21] (2011, Aug.) Open-source micro-robotic project. [Online]. Available: http://www.swarmrobot.org/

[22] A. Kettler, M. Szymanski, J. Liedke, and H. Wörn, "Introducing wanda - a new robot for research, education, and arts," in *IROS: Conference on Intelligent Robots and Systems*, 2010, pp. 4181–4186.

[23] *Using JMP 9*, SAS Institute Inc., Cary, NC, USA, Oct. 2010.

[24] R Development Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2011. [Online]. Available: http://www.R-project.org