Acquiring Moving Skills in Robots with Evolvable Morphologies: Recent Results and Outlook

Milan Jelisavcic
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
m.j.jelisavcic@vu.nl

Evert Haasdijk
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
e.haasdijk@vu.nl

A. E. Eiben
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
a.e.eiben@vu.nl

ABSTRACT

We construct and investigate a strongly embodied evolutionary system, where not only the controllers but also the morphologies undergo evolution in an on-line fashion. In these studies, we have been using various types of robot morphologies and controller architectures in combination with several learning algorithms, e.g. evolutionary algorithms, reinforcement learning, simulated annealing, and HyperNEAT. This hands-on experience provides insights and helps us elaborate on interesting research directions for future development.

CCS CONCEPTS

• **Computing methodologies** → **Evolutionary robotics**; *Mobile agents*; • **Theory of computation** → *Evolutionary algorithms*;

KEYWORDS

Evolutionary robotics, On-line evolution, Indirect encoding, Lamarckian evolution, Gait learning

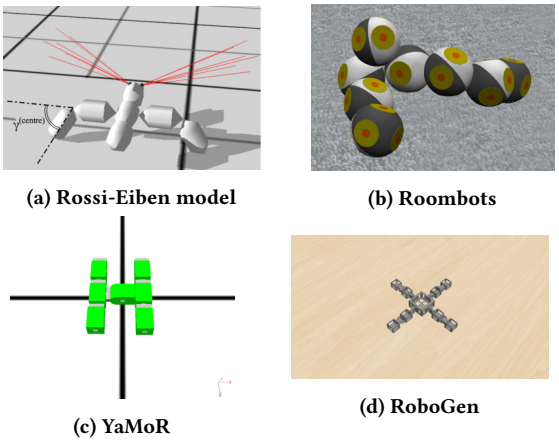
ACM Reference format:
Milan Jelisavcic, Evert Haasdijk, and A. E. Eiben. 2017. Acquiring Moving Skills in Robots with Evolvable Morphologies: Recent Results and Outlook. In *Proceedings of GECCO '17 Companion, Berlin, Germany, July 15-19, 2017,* 7 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 BACKROUND AND OBJECTIVES

The long-term vision behind the research covered in this paper foresees robotic (eco)systems that evolve in real time and real space. This implies that the robot morphologies (body, hardware), as well as the controllers (mind, software) are evolvable, i.e., subject to reproduction and selection. In other words, we are concerned with robots that can produce offspring based on their fitness determined by the environment and the functional criteria set by the given application.

Over the last six years several papers have introduced and discussed this vision as a whole [7, 12] or particular algorithmic aspects of it [5, 10, 11]. The underlying system architecture called the Triangle of Life has been put forward in 2013 in [6]. This triangle captures the pivotal life cycle of an ecosystem of self-reproducing robots. This lifecycle does not run from birth to death, but from conception (being conceived) to conception (conceiving one or more children) and it is repeated over and over again, thus creating consecutive generations of robot children. The result is a population of robotic organisms that evolves and thus adapts to the given environment. The Triangle of Life consists of 3 stages, Morphogenesis, Infancy, and Mature Life.

This paper focusses on the Infancy stage. We assume that there is a procedure for Morphogenesis that can produce new robotic organisms. Obviously, this procedure depends on the chosen type of robot morphologies, but in all cases, it holds that the body (morphological structure) and the mind (controller) of a new robotic organism will unlikely fit each other well. Even if the parents had well-matching bodies and minds, crossover and mutation can easily result in a child where this is not the case. Hence, the new robot needs to do some fine tuning; not unlike a newborn calf the 'baby robot' needs to learn how to control its own body. This problem –the Control Your Own Body (CYOB) problem– is inherent to Artificial Life systems where newborn organisms are random combinations of the bodies and minds of their parents.



(a) Rossi-Eiben model (b) Roombots (c) YaMoR (d) RoboGen

Figure 1: Illustration showing four different morphology designs used for the experiments with a modular robots.

The first essential skill for newborn robots is arguably locomotion, eventually in combination with vision. Therefore, we have investigated this issue extensively in the past in search of a generic approach to learning locomotion skills in robots with evolvable morphologies. Notice, that the evolvability of morphologies represents a particular requirement for the learning algorithms. Whereas most related work (that we, unfortunately, cannot review here by the lack of space) is based on a given specific robot body, our method

GECCO '17 Companion, July 15-19, 2017, Berlin, Germany

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

should be able to work in any body form that can be constructed within the given design space. To this end we make an assumption: the robot bodies are modular constructs with a few basic types of modules that can be combined into a large number of particular robots. Figure 1 shows some of the forms we have studied.

The central research question can be phrased as follows.

> What is the best mechanism for acquiring adequate moving skills in a large space of (evolvable) robot morphologies?

Let us note that this question can be further split into two subquestions regarding the controller architecture (e.g., neural nets, periodic oscillators, decision trees) and the learning algorithm itself that obtains a high quality controller (e.g., back-propagation, reinforcement learning, or evolution in the space of controllers).

In the rest of this paper, we review our results regarding this question, draw conclusions based on these results, and identify the most important findings. We conclude this overview by discussing some promising directions for future research.

2 REVIEW OF RESULTS

In order to cover key aspects of the recent research toward constructing an efficient locomotion mechanism, a few common key aspects should be noted to make a clear distinction between results from each paper. The type of simulator, learner and controller mechanism, and morphology are the meeting points for this research. Two different simulators were used: Webots [16] and Revolve [13]. Common ground for all tested morphologies is their modularity regardless of the modules design choice. The design choices include Roombots, YaMoR, and RoboGen models, with predominant usage of latest choice. Concerning learner-controller mechanisms, most of our research investigates possibilities of RL PoWER learner in pair with spline-based controllers, but also we compare it with HyperNEAT learner which works in pair with indirectly encoded Compositional Pattern-Producing Networks (CPPNs).

Simulators. In first three presented researches, we used Webots simulator as our tool of choice. Webots is a commercial robot simulator proved to be a good research platform and is a good starting point for designing our system. Webots uses the Open Dynamics Engine (ODE) for simulating rigid body dynamics, which allows one to accurately simulate physical properties of objects such as velocity, inertia, and friction.

In our later research, simulated experiments were migrated to a new platform, custom made for our purposes, called Revolve. Revolve has several advantages over Webots; in first place, its open-source nature. It is built as a wrapper of Gazebo [15], an open source, multi-platform robotic simulation package. Apart from ODE, it also provides support for Bullet, Simbody and DART physics engines, which allows us to verify experimental results.

Morphologies. As mentioned, the robot forms are constructed from one or a few basic types of modules that are combined into larger and more complex forms. In the first experimental setup, the modular robots used within Webots are self-designed capsule modules connected to each other by two degrees of freedom (DOFs) actuated joints that allow both horizontal and vertical movement (Fig. 5a). Each module has six connection points, and each robot has a special head module provided with two distance sensors. In the second experimental setup, the simulated modules (Fig. 5b) in Webots are based on an existing hardware platform (Roombots) which makes the system, in principle, constructible [18]. Every module consists of two cubic-like blocks, ten active connection mechanisms (ACMs) and three actuated joints. In the third experimental setup, another promising module design was tested, namely YaMoR (Fig. 1c). A YaMoR module used for robot representation [17] is made of a static body and a joint on its front that has a single degree of freedom and an operating range of $[-\pi/2, \pi/2]$. It also has two connectors, one on the joint and one in the back of the body, which allows connecting modules at arbitrary angles. For the purpose of investigation, three changes were applied to the original YaMoR model. Two extra connectors on the remaining sides of the body in a central position were added, allowing the construction of complex structures.

(a) Simulated "gecko" morphology.

(b) Hardware implementation.

Figure 2: Illustration showing a software and a hardware robot model compared for gait learning.

Obviously, the specific robot design and the construction procedure are closely related. A straightforward idea is to use rapid prototyping (3D printing) in the production center. To mitigate this problem, after migrating to Revolve, we have chosen the robot design featured in RoboGen (Fig. 1d), and all subsequent research was verified on this model [1]. The RoboGen framework includes multiple different simple components which are simple to 3D print, an important aspect for implementing an envisioned embodied robotic system.

Controllers. Among selected controller approaches used to test on our modular systems, predominantly spline-based controllers were tested, but also Central Pattern Generator (CPG) controllers were used. Spline-based controllers represent an open-loop controller which lacks a sensory feedback but is easy to implement and test used learner algorithms. Considering CPGs, different approaches exist, and we focused on a sinusoidal pattern generator and a differential oscillator approaches.

Spline-based controller represents a set of splines that all together form a gait policy for particular morphology. Each spline within this set specifies the angular positions of a single actuator over a certain amount of time. With an update function, the robot can send a signal to reposition its actuators based on a spline value at a certain time point. A cyclic spline is a mathematical function that is defined using a set of n control points. Each control point is defined by (t_i, a_i), where t_i represents time and a_i the corresponding value. $t_i \in [0, 1]$ is defined as

$$t_i = \frac{i}{n-1}, \forall i = 0, \ldots, (n-1)$$

and $\alpha_i \in [0, 1]$ is freely defined.

To ensure cyclic splines, an additional control point (t_n, α_n) is defined that by definition has the same value as the first control point ($\alpha_0 = \alpha_n$). These control points are then used to interpolate a cubic spline with periodic boundary conditions using GSL [1] dedicated C functions. Using GSL it is possible to query a spline for a different number of points than it was defined with.

The CPG approach with a sinusoidal pattern generator was tested in Sec. 2.1. The modular robots used have two Degrees of Freedom (DOFs) and are actuated according to a periodic function, in the form

$$\phi_i(t) = \alpha_i \sin(\omega t + \beta_i) + \gamma_i^{(Target)} \cdot angle(t) + \gamma_i \cdot diff(t)^{(Obstacle)} + \gamma_i^{(Centre)}$$

where $\phi_i(t)$ is a sinusoidal function that determines the position of joint i between two body segments ($i = 1..2(n-1)$), n is the number of body segments of the robot. Parameters ω, α_i and β_i determine, respectively, the angular speed, amplitude, and phase of the oscillation of the joints, α_i being in the range $[-\pi/2, \pi/2]$. Considering other parameters, $diff(t)$ is the difference between the range sensors readings, $angle(t)$ is the angle between the robotâĂŹs orientation. A robot‘s controller can be thus described by an array of floating-point values of length $5 \cdot 2(n-1) + 15$, with 5 parameters governing the joints‘ motion, plus one for the common ω.

Differential oscillators are another approach tested as the core of CPG implementation. Each oscillator is defined by two neurons that are recursively connected as shown in Fig. 3. These generate

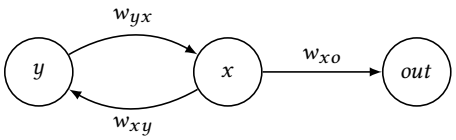


Figure 3: A differential oscillator with output node as used in the CPG controller.

oscillatory patterns by calculating their activation levels x and y according to the following differential equation:

$$\dot{x} = w_{yx} y + bias_x$$
$$\dot{y} = w_{xy} x + bias_y$$

With w_{xy} and w_{yx} denoting the weights of the connections between the neurons; $bias_x$ and $bias_y$ are parameters of the neurons. If w_{yx} and w_{xy} have different signs the activation of the neurons x and y is periodic and bounded. An oscillator‘s x node is connected to a linear output neuron that in turn connects to the robot‘s active hinge. Output neurons use the following activation function:

$$f(x) = (w_{xo} \cdot x - bias) \cdot gain.$$

with x the activation level from the oscillator, w_{xo} the weight of the connection between oscillator and output node and *bias* and *gain* parameters. Each active joint in the robot body is associated with an oscillator and connected to it through an output neuron that determines the joint‘s angle.

[1] http://www.gnu.org/software/gsl/

Learners. The RL PoWER implementation follows the description by Jens Kober and Jan Peters [14] and [2] If the organism is from the initial population the algorithm starts by creating the initial policy π_0 with as many splines as there are motors in the organism. These splines are initialised with n values of 0.5 and then adding Gaussian noise. Otherwise, the minds of the parents are combined as explained later and this mind is used as the initial policy. The initial policy is then evaluated after which it is adapted. This adapted controller is evaluated and adapted again until the stopping condition is reached. Adaptation is done in two steps which are always applied: exploitation and exploration. In the exploitation step, the current splines $\hat{\alpha}$ are optimised based on the outcome of previous controllers, this generates a new set of splines.

$$\hat{\alpha}_{i+1} = \hat{\alpha}_i + \frac{\sum_{j=1}^{k} \hat{\Delta}\alpha_{i,j} R_j}{\sum_{j=1}^{k} R_j}$$

where $\hat{\Delta}\alpha_{i,j}$ represents the difference between the parameters of the i-th policy and j-th policy belonging to a ranking of the best k policies seen so far and R_j its reward. In the exploration phase policies are adapted by applying Gaussian perturbation to the newly generated policy.

$$\hat{\alpha}'_{i+1} = \hat{\alpha}_{i+1} + \hat{\varepsilon}_{i+1}, \hat{\varepsilon}_{i+1} \sim \mathcal{N}(0, \sigma^2)$$

where $\hat{\alpha}_{i+1}$ are the parameters after the exploitation step, $\hat{\alpha}'_{i+1}$ the parameters after the exploration step and $\hat{\varepsilon}_{i+1}$ values drawn from a Gaussian distribution with mean 0 and variance σ^2.

Each controller is evaluated for a fixed time as follows:

$$R_i = \left(100 \frac{\sqrt{\Delta_x^2 + \Delta_y^2}}{\Delta_t} \right)^6$$

where Δ_x and Δ_y is the displacement over the x and y axes measured in meters and Δ_t the time of an evaluation.

Proposed HyperNEAT [20] , an indirectly encoded evolutionary algorithm for neural networks. The idea behind HyperNEAT is to assign the nodes in a substrate neural network a location in an n-dimensional hypercube. The assigned relative positions should in some way reflect a relationship between the nodes. allowing the algorithm to exploit the geometry of the problem. The coordinates of two nodes in the hypercube are then input values for a Compositional Pattern Producing Network (CPPN), which outputs a value for the weight of their connection. The CPPN evolves using NEAT [21] so that the substrate network‘s performance is optimised.

Like a neural network, a CPPN is a network of mathematical functions with weighted connections. Unlike neural networks, the network can contain a variety of activation functions including Sine, Cosine, Gaussian and Sigmoid. To determine the weight of a connection in the neural network that controls the robot (the substrate), the coordinates of the two substrate nodes are fed into the CPPN which then returns the connection weight [19].

The CPG nodes are positioned in a three-dimensional hyperspace. Such *modular differentiation* allows specialisation of the active hinge‘s movements depending on its relative position in the robot. The hinge coordinates are obtained from a top-down view of the robot body. Thus, two coordinates of a node in the CPG controller correspond to the relative position of the active hinge

it is associated with. The third coordinate depends on the role of the node in the CPG network: output nodes have a value of 0 and differential nodes have values of 1 for x and -1 for y nodes. The CPPNs have six inputs denoting the coordinates of a source and a target node. The CPPNs have three outputs: the weight of the connection from source to target as well as the bias and gain values for the target node

For spline-based controllers, the CPPNs must output the α_i value for a given t_i for each spline in the controller. Obviously, t_i must be one of the inputs for the CPPN. As before, two further inputs correspond with the position of the active hinge associated with the spline. Thus, CPPNs for splines have three inputs: the coordinates of the active hinge and t_i. They have one output: α_i.

2.1 Simultaneous versus Incremental Learning of Multiple Skills by Modular Robots

The main question concerned in this paper is whether it is better to learn multiple skills simultaneously (all-at-once) or incrementally (one- by-one). An experimental study was conducted with modular robots of various morphologies that need to acquire three different but correlated skills, efficient locomotion, navigation towards a target point, and obstacle avoidance, using a real-time, on-board evolution as the learning method.

In this research, the main focus is on generating controllers that are capable of dealing with the generic motion task (i.e., moving from one place to another, avoiding obstacles in between) according to their sensory input without the need of switching between behaviors, that is, without a higher level behavior control. One strategy would be trying to learn all the skills at the same time, eg. *simultaneous learning*. The simultaneous learning approach is compared with an alternative strategy based on a 'syllabus' that consists of three sequential classes. In Class 1, robots have to learn how to control their bodies in order to generate gaits for efficient unconstrained locomotion. In Class 2, robots have to learn how to move towards a target point, and in Class 3, robots have to learn how to move around obstacles.

The robot ecosystem is modeled in Webots simulator based on Rossi-Eiben model (Fig. 5a) of the design of the modules.

The adopted learning strategy is a simple (1 + 1) Evolutionary Strategy running on every robot in pair with controller implemented as sinusoidal CPG mentioned earlier. This choice of a learner is motivated by the natural encoding of the robots controllers as arrays of floating point values.

The first tests compare locomotion and target reaching and was carried out in the following way. The simultaneous learning process was allotted a fixed learning time, i.e. a fixed number of generations, set to 100. The incremental learning process was stopped when it reaches comparable fitness to the simultaneous learning process. The results show that, in general, learning in two steps takes less than 100 generations

In the second test of this series, the learning of the three tasks (locomotion, target reaching and obstacle avoidance) was compared. Here, a wall was put in between the robots' initial position and the target, and the robots had to learn how to avoid it to reach their target. Again, incrementally learning the tasks takes less time than learning them simultaneously.

2.2 A Robotic Ecosystem with Evolvable Minds and Bodies

This research is primarily concerned with investigating whether the robots, endowed with reinforcement learning capabilities, learn to locomote efficiently during their lifetime and how this learning ability evolves. Apart from this main task, it is also beneficial to investigate whether the system could develop a sustainable population of evolving organisms.

The system is implemented in the Webots simulator based on an existing hardware platform Roombots.

Organisms are not required to perform any specific task and are free to move in any direction. This reduces the locomotion problem to gait learning. Particularly, it requires the generation of rhythmic functions for the activation of the organisms' step motors. The RL PoWER algorithm has been chosen for gait learning in this project based on previous investigations [3]. Learning is not restricted to the infancy period, but organisms continue learning for their full lifetime.

Considering the research objective "to investigate if robots, endowed with reinforcement learning capabilities, learn to locomote efficiently during their lifetime and how this learning ability evolves." The robots are indeed capable of lifetime learning and profoundly improve their locomotion capabilities over their lifetime. The evolution of the initial minds does not, however, seem to have a great impact on the learning ability, but this may change with longer evolutionary runs.

Another stated research objective was "to show that the system enables sustainable populations of evolving organisms that are born, learn and procreate autonomously." Results have indeed shown that in the environment as it is defined, with movement through the environment promoting fecundity, the robots evolve to move around the arena and so encounter mates. This allows them to procreate, resulting in a viable population that spans several generations.

2.3 Online Gait Learning for Modular Robots with Arbitrary Shapes and Sizes

Even if the parents had well-matching bodies and minds, recombination and mutation can easily result in a child where this is not the case. The main question in this work was: "What is the best and fastest gait learning approach for modular robots? "

All tests were done in the simulation with the Webots based on YaMoR module design. The environment chosen for the experiments is an infinite plane free of obstacles so to avoid any extra complexity and the need of supervision. Each experiment starts with the organism lying completely flat at the plane origin.

In order to test and verify different learning approaches, several algorithms were compared which includes RL PoWER, Simulated Annealing, and HyperNEAT. RL PoWER and Simulated Annealing were tested in pair with a spline-based controllers and HyperNEAT evolves a neural network's connectivity pattern indirectly, using a generative encoding mentioned as a CPPNs.

The controller used with RL PoWER and Simulated Annealing is a set of cyclic splines that define an open-loop gait. The HyperNEAT experiments use a neural network that controls a closed-loop gait composed of three layers: input, hidden and the output layer. Each

layer is an $m \times n$ matrix of nodes where $m = (OrganismSize_x \cdot 2) - 1$ and $n = (OrganismSize_y \cdot 2)$, with $OrganismSize_x$ and $OrganismSize_y$ the sizes of the organism respectively on the x and y axes measured by the number of modules. The inputs are the angular positions of each module's servo at the previous time step together with a sinusoidal signal $s = \sin(\omega t)$ where ω represents the maximum angular velocity of the modules servo and t the current time. The network outputs the angular positions of each module servo for the current time step.

The experiments were conducted with three complexity levels: organisms with two extremities (I-shape), three extremities (T-shape), and four extremities (H-shape). The second test suite was constructed by generating 270 random shapes.

In most cases, Simulated Annealing fares worse than RL PoWER, particularly on the T and H shapes. On the I shapes Simulated Annealing fares much better, even matching RL PoWER performance for the I-7 shape. Once converged, RL PoWER provides consistent performance, while Simulated Annealing's performance of consecutive policies is more erratic. The difference in performance between organisms of the same size, but the different shape is significant in all cases. This supports the conclusion that with either algorithm the complexity of the shape has a larger influence on the performance than the size of the shape.

The results for HyperNEAT show very competitive controllers, in many instances outperforming the best results of RL PoWER and Simulated Annealing. The controllers from the best run (blue dots) are among the best found in our experiments. On the other hand, the median performances are much worse than those for RL PoWER: HyperNEAT succeeds in finding very good controllers at the cost of evaluating many poorly performing controllers as well.

This highlights an important issue in *on-line* evolution: because the robot controllers evolve while the robots perform their tasks, the robots' actual performance is determined by the quality of all the controllers they evaluate, not only by the best controllers they consider. In reinforcement learning terms, one must consider the balance between exploration and exploitation when employing evolution in on-line scenarios. This is a radical departure from the optimisation-centred mindset in most evolutionary robotics research that implies the off-line development of controllers that do not evolve once deployed.

2.4 Improving RL Power for On-Line Evolution of Gaits in Modular Robots

The main objective of this work is to increase the performance of RL PoWER by altering the main search operators used therein. The basis for the objective is (re)describing RL PoWER as an evolutionary algorithm (EA) with a specific mutation and crossover operator. Considering RL PoWER from an EA perspective provides hints for possible improvements by using a different crossover, a different mutation, or both.

RL PoWER can be viewed as an evolutionary algorithm with policies as individuals, fitness defined as the corresponding reward, population size k, an elitist $(k + 1)$ selection strategy, a k-parent crossover, and Gaussian mutation [8].

Regarding mutation, using self-adaptive step-sizes (σs) is a promising option that is known to work well for numerical optimization problems. The simplest version of this method uses one step size globally. This σ is mutated each time step *before* using it to create a new individual by multiplying it by a term $\exp^{\Gamma}$ where Γ is random variable from normal distribution.

$$\sigma' = \sigma \cdot \exp^{\tau \cdot N(0,1)}$$

The used simulator is Revolve, the Robot Evolve toolkit developed at our department with RoboGen modules as the basis for a robot design.

The results showed a difference between employing the RL PoWER algorithm in its original form and using it with modified mutation and crossover operators. The original RL PoWER is fast to learn, but also that it converges very quickly to suboptimal solutions. Reducing the number of parents in the crossover from ten to two improves performance and so does the usage of self-adaptive mutation step-sizes. The overall 'winner' regarding the final solution quality is algorithm version that combines both extensions.

2.5 Real-World Evolution of Robot Morphologies: A Proof of Concept

The first of the two research objectives presented here is to investigate whether it is possible to construct the system that enables sustainable populations of evolving organisms that are born, learn and procreate autonomously, in simulation as well as in hardware. Secondly, given the opportunity, it is highly relevant to investigate the differences between the behaviour of designed robots in simulation and in the real world, eg. to measure the 'reality gap'.

To validate the choice of RoboGen as the physical substrate and genetic representation for online evolution of robot morphology, experiments where conducted with a population of simulated robots that coexist in a featureless arena and are centrally evaluated and selected. New individuals are inserted at a fixed rate of one every 15 s. Two parents are selected using using four-tournament selection, and their offspring is generated using RoboGen's recombination and variation operators. The offspring is then placed at a random position within a circle of radius 2 m around the origin.

Earlier works identified RL PoWER as a reliable and efficient algorithm for gait learning in arbitrary morphologies with modular robots consisting of homogeneous modules. Here these findings were verified for the RoboGen-based morphologies using the Revolve simulator. In these experiments, RL PoWER was revisited, and it was noted that it is, in essence, an evolutionary algorithm, which subsequently was improved by adding two-parent crossover with binary tournament selection.

Figure 4 shows the development of fitness over time. The experiments were terminated after the birth of 100 individuals, which is a reasonable number of individuals to consider also in real-world experiments. It is clear that the robots rapidly improve their locomotion capabilities, showing that the substrate, genetic encoding, and variation operators are suitable for online evolution and can yield interesting results in a limited number of evaluations.

As mentioned, the task of the robot's learning algorithm is to optimize the robot's controller so that performance –in this case, the distance covered by the robot– is maximized. Here we compare the performance in the simulation, as well as in real system. Figure 5 shows the results from testing the learning algorithm

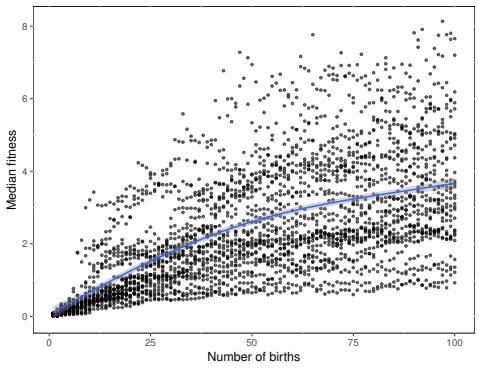
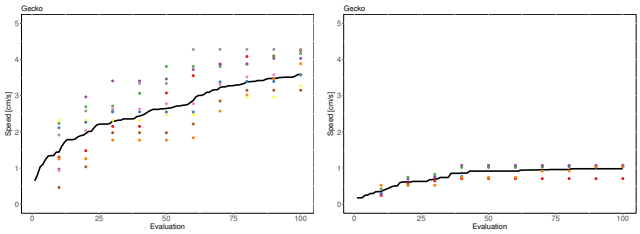


Figure 4: Fitness progression of 30 replicate runs, with the number of birth events as the time scale. Each point represents a median value of an entire population within a birth time frame.

on real hardware. The real hardware results show a significant decrease in performance when compared to the simulation. This can be attributed to several factors: (1) servo motors were constantly breakingâĂŤon average 1.5 motors per run; (2) the weight of a robotÊijs head significantly influences the performed gait; (3) robots were bounded in a small arena rather than the infinite plane in the simulation. However, there is an evident positive trend in performance, and after 30 evaluations (which take 20 to 30 min) the robots have obtained gaits that allow them to traverse the arena.



(a) Learning with RL PoWER in simulation.

(b) Learning of the same algorithm in hardware.

Figure 5: Illustration showing a gait learning trends in "gecko" morphology tested in the software and in the hardware, making 'reality gap' problem vivid.

2.6 Analysis of Lamarckian Evolution in Morphologically Evolving Robots

Implementing lifetime learning by means of on-line evolution, we establish an indirect encoding scheme that combines CPPNs and CPGs as a relevant learner and controller for open-loop gait controllers in modular robots which have evolving morphologies. Here, we address this challenge by means of a lifetime learning approach for robot controllers, focussed on scenarios where robots' morphologies evolve. In particular, we show how the use of an indirect encoding can enable a Lamarckian evolutionary set-up, allowing learned control knowledge of parent robots to be transferred to o spring.

The experiment was conducted in Revolve simulator with RoboGen morphologies. By using this indirect encoding, the CPPN can easily be transferred from one robot to a robot with a di er- ent morphology. The control system consists of CPGs with a differential oscillator in every active joint, with connections linking neighboring CPGs.

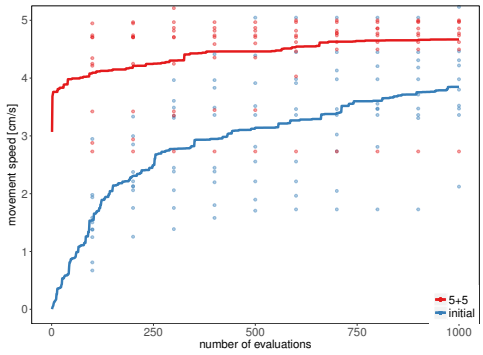


Figure 6: Comparison of Baldwinian approach (blue) opposed to Lamarckian approach (red). Scenarios are tested on an offspring created combining "spider" and "gecko" morphologies mentioned earlier.

Experimental validation on the morphologically evolved robots shows that a Lamarckian setup with CPPN-CPG provides substantial benefits compared to controllers learned from scratch. At the end of the evaluation budget (1000 evaluations), the approaches have reached mostly similar locomotion performances, although the Lamarckian approach seems to have a slight advantage. This may indicate that, given enough time, both approaches could reach the practical limits for a specific morphology. However, the Lamarckian approach clearly has an advantage in the earlier stages of the learning process, reaching higher locomotive performance in the first 100 evaluations. This characteristic of quick convergence to a high locomotive performance is particularly interesting for a scenario where the learning is performed on real robots. In such cases, evaluations are both time-consuming and may cause considerable wear and tear on the robots, and thus an early termination of the learning phase would be a considerable advantage.

3 SUMMARY AND OUTLOOK

In the foregoing, we have reviewed our results regarding the challenge of finding a good mechanism that can learn a good controller for locomotion in 'newborn' modular robots whose morphology is not known in advance. In these studies, we have been using various types of robot morphologies and controller architectures in combination with several learning algorithms, e.g. evolutionary algorithms, reinforcement learning, simulated annealing, and HyperNEAT. An important aspect in all these experiments is the on-line nature of the learning algorithm: learning takes place *during* (and not *before*) the operational period of the robot. This implies that the computational budgets are typically quite low. Overall,

our findings can be summarized in a few observations. First and foremost, it is possible to learn a good gait with a relatively low number of trials that make on-line learning in real time practicable. We found that the (10+1) evolution strategy –a generalization of the RL PoWER reinforcement method– is a good algorithm for this purpose. Using splines for the robot controllers is successful for the simple gait learning task in an open loop without sensory feedback. However, it has limitations for more complex tasks that require a closed loop that uses visual and/or tactile inputs for tasks like directed locomotion or obstacle avoidance. Last but not least, we have promising results that support the use of Lamarckian evolution, where the controllers are partly inheritable and partly learnable. We expect that such mechanisms can accelerate the individual learning process in the 'newborn' robots.

Regarding future work let us only mention two interesting directions: the use of generative encodings because it promises increased levels of evolvability and closed loop controller mechanisms that can accommodate sensory inputs for these are necessary for purposeful behavior, e.g., approaching mating partners and food (energy sources).

REFERENCES

[1] Auerbach, J., Aydin, D., Maesani, A., Kornatowski, P., Cieslewski, T., Heitz, G., Fernando, P., Loshchilov, I., Daler, L., and Floreano, D. (2014). RoboGen: Robot Generation through Artificial Evolution. In Sayama, H., Rieffel, J., Risi, S., Doursat, R., and Lipson, H., editors, *Artificial Life 14: Proceedings of the Fourteenth International Conference on the Synthesis and Simulation of Living Systems*, pages 136–137, New York, New York, USA. The MIT Press.

[2] D'Angelo, M., Weel, B., and Eiben, A. (2014). Hyperneat versus rl power for online gait learning in modular robots. In Esparcia-Alcázar, A., editor, *Proceedings of EvoApplications 2014: Applications of Evolutionary Computation*, number 8602 in Lecture Notes in Computer Science, pages 777–788. Springer, Berlin, Heidelberg, New York.

[3] D'Angelo, M., Weel, B., and Eiben, A. E. (2013). Online Gait Learning for Modular Robots with Arbitrary Shapes and Sizes. pages 45–56.

[4] De Jong, K. and Sarma, J. (1995). On decentralizing selection algorithms. In Eshelman, L., editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 17–23, San Francisco, CA. Morgan Kaufmann.

[5] Eiben, A. (2014a). Grand challenges for evolutionary robotics. *Frontiers in Robotics and AI*, 1(4).

[6] Eiben, A., Bredeche, N., Hoogendoorn, M., Stradner, J., Timmis, J., Tyrrell, A., and Winfield, A. (2013). The triangle of life: Evolving robots in real-time and real-space. In Liò, P., Miglino, O., Nicosia, G., Nolfi, S., and Pavone, M., editors, *Advances In Artificial Life, ECAL 2013*, pages 1056–1063. MIT Press.

[7] Eiben, A. and Smith, J. (2015a). From evolutionary computation to the evolution of things. *Nature*, 521(7553):476–482.

[8] Eiben, A. and Smith, J. (2015b). *Introduction to Evolutionary Computing*. Springer, 2nd edition.

[9] Eiben, A. E. (2002). Multiparent recombination in evolutionary computing. In Ghosh, A. and Tsutsui, S., editors, *Advances in Evolutionary Computing*, Natural Computing Series, pages 175–192. Springer.

[10] Eiben, A. E. (2014b). In Vivo Veritas: Towards the Evolution of Things. In Bartz-Beielstein, T., Branke, J., Filipič, B., and Smith, J., editors, *Parallel Problem Solving from Nature âĂŞ PPSN XIII*, pages 24–39, Ljubljana, Slovenia.

[11] Eiben, A. E. (2015). EvoSphere: The World of Robot Evolution. pages 3–19, Mieres, Spain. Springer International Publishing.

[12] Eiben, A. E., Kernbach, S., and Haasdijk, E. (2012). Embodied artificial evolution – artificial evolutionary systems in the 21st century. *Evolutionary Intelligence*, 5(4):261–272.

[13] Hupkes, E. (2016). Revolve: An Evolutionary Robotics Toolkit.

[14] Kober, J. and Peters, J. (2009). Learning motor primitives for robotics. In *2009 IEEE International Conference on Robotics and Automation*, pages 2112–2118, Kobe, Japan. IEEE.

[15] Koenig, N. and Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2149–2154. IEEE.

[16] Michel, O. (2004). Webots: Professional mobile robot simulation. *Journal of Advanced Robotics Systems*, 1(1):39–42.

[17] Möckel, R., Jaquier, C., Drapel, K., Dittrich, E., Upegui, A., and Ijspeert, A. (2006). YaMoR and Bluemove – an autonomous modular robot with Bluetooth interface for exploring adaptive locomotion. In Tokhi, M. O., Virk, G., and Hossain, M. A., editors, *Proceedings of the 8th International Conference on Climbing and Walking Robots (CLAWAR) 2005*, pages 685–692. Springer.

[18] Sproewitz, A., Billard, A., Dillenbourg, P., and Ijspeert, A. (2009). Roombots – mechanical design of self-reconfiguring modular robots for adaptive furniture. In *IEEE International Conference on Robotics and Automation (ICRA'09)*, pages 4259–4264. IEEE.

[19] Stanley, K. O. (2007). Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, 8(2):131–162. Special issue on developmental systems.

[20] Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212.

[21] Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127.