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

Jelisavcic, Milan; De Carlo, Matteo; Hupkes, Elte; Eustratiadis, Panagiotis; Orlowski, Jakub; Haasdijk, Evert; E. Auerbach, Joshua; Eiben, A.E.

published in
Artificial life
2017

DOI (link to publisher)
10.1162/ARTL_a_00231

document version
Publisher's PDF, also known as Version of record

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

Milan Jelisavcic[**]
Matteo de Carlo[**]
Vrije Universiteit Amsterdam

Elte Hupkes[†]
Universiteit van Amsterdam

Panagiotis Eustratiadis[**]
Vrije Universiteit Amsterdam

Jakub Orlowski[‡]
University of Warsaw

Evert Haasdijk[*,**]
Vrije Universiteit Amsterdam

Joshua E. Auerbach[§]
Champlain College

A. E. Eiben[**]
Vrije Universiteit Amsterdam

Abstract Evolutionary robotics using real hardware has been almost exclusively restricted to evolving robot controllers, but the technology for evolvable morphologies is advancing quickly. We discuss a proof-of-concept study to demonstrate real robots that can reproduce. Following a general system plan, we implement a robotic habitat that contains all system components in the simplest possible form. We create an initial population of two robots and run a complete life cycle, resulting in a new robot, parented by the first two. Even though the individual steps are simplified to the maximum, the whole system validates the underlying concepts and provides a generic workflow for the creation of more complex incarnations. This hands-on experience provides insights and helps us elaborate on interesting research directions for future development.

Keywords
Evolutionary robotics, embodied evolution, modular robots, artificial life

1 Introduction

The work described in this article forms a stepping stone towards the grand vision of the *evolution of things* as outlined in [13]. The essence of this vision is to construct, study, and utilize artificial evolutionary systems in physical substrates, that is, in the real world, not in digital worlds in computer simulations. There are various possible avenues towards this goal, including chemical, biological, and robotic approaches [12]. This study falls in the last category; the long term goal is to build robots that can evolve in real hardware [15].

The motivational scenario is that of a physical habitat where a group of robots operates: evolves, learns, and "works" [16]. The underlying system architecture is based on a model of the robotic life

[*] Contact author.
[**] Vrije Universiteit Amsterdam, Amsterdam, Netherlands. E-mail: m.j.jelisavcic@vu.nl (M.J.); matteo.dek@gmail.com (M.deC.); peustratiadis@gmail.com (P.E.); e.haasdijk@vu.nl (E.H.); a.e.eiben@vu.nl (A.E.E.)
[†] Universiteit van Amsterdam, Amsterdam, Netherlands.
[‡] University of Warsaw, Warsaw, Poland. E-mail: j.orlowski@student.uw.edu.pl
[§] Champlain College, Burlington, VT, USA. E-mail: a.e.eiben@vu.nl

cycle as outlined in [17]. This model describes a cycle not from birth to death, but from conception (being conceived) to conception (conceiving an offspring). The cycle consists of three main stages: morphogenesis, infancy, and mature life. Our system does not contain a centralized evolution manager to monitor the fitness of population members and to perform selection. This feature sets it apart from other studies on evolving morphologies of real robots. Work in this line of research was pioneered by Lipson and Pollack [30], and more recently followed by the RoboGen system of Auerbach et al. [1], who ran evolution in simulation and constructed the end result of a given evolutionary experiment: one robot, in real hardware. Broadback et al. also used a genetic algorithm to evolve robot designs, but in contrast to [30] and [1] all fitness evaluations were performed in hardware, one at a time. When evaluating a new genotype, the corresponding robot phenotype was constructed and tested, and the measured fitness value was passed back to the GA running on a desktop computer [6]. Our system model differentiates itself in the following ways:

- The robots (population members) exist and operate concurrently in the same real-world habitat.
- There is no centrally orchestrated selection-reproduction cycle; birth events and death events can take place independently.
- Robots are evaluated continuously; mate selection is performed by the robots themselves (two robots can agree to parent a child if they pass each other's selection threshold); survivor selection is performed by the environment (robots can break or run out of energy, thus becoming subject to removal and recycling).

Furthermore, we consider robots that have sensors (those in [30] and [6] do not) and can learn (those in [1] cannot; all of their properties are inheritable or evolvable).

The main contributions of this article are the following. First, we describe and discuss a system architecture for physically evolving robot populations, based on the previously published triangle-of-life model [17]. We position our system with respect to existing work and elaborate on options for implementation. Second, we describe a proof-of-concept implementation to demonstrate how the three stages of the generic model can be realized, though in a simplified form, and connected into one life cycle. To be specific, we start with a couple of robot genotypes and (1) construct the physical robots (the phenotypes) specified by these genotypes, (2) have these robots undergo a basic gait learning process and become adults, (3) let the adult robots mate, thus creating a new robot genotype, and (4) construct the robot specified by this genotype. This last step closes the loop and extends the robot population with a child. Third, we review the lessons learned from this project and identify important issues for further research and development.

2 Related Work

Evolutionary robotics is the combination of evolutionary computing and robotics [4, 10, 11, 18, 34, 39, 40, 42]. The field "aims to apply evolutionary computation techniques to evolve the overall design, or controllers, or both, for real and simulated autonomous robots" [40, p. 74]. This approach is "useful both for investigating the design space of robotic applications and for testing scientific hypotheses of biological mechanisms and processes" [18, p. 1423]. However, as noted in [4, p. ix], "the use of meta-heuristics [i.e., evolution] sets this sub-field of robotics apart from the mainstream of robotics research," which "aims to continuously generate better behaviour for a given robot, while the long-term goal of Evolutionary Robotics is to create general, robot-generating algorithms." This provides the context for our work, which aims to employ online evolution in real time and real space to deliver robot morphologies and controllers suited for a given environment.

One of the long-standing challenges in evolutionary robotics is the reality gap: the effect that solutions evolved in simulation do not work well once transferred to the real system. The reason for this is that simulations are prone to hidden biases, errors, and simplifications in the underlying models and that EAs are likely to exploit the features of the (simulated) environment, even if they lack physical plausibility. Approaches to coping with this problem include the classic *envelope of noise* [24], as well as more recent techniques for improving simulators, such as exploiting experiments with the physical robot [20, 46] and the use of a *transferability function* to predict the limits of the simulation [27]. Such approaches to closing the reality gap depart from simulated evolution and bring the physical robot into the evolutionary loop [22].

Alternatively, the reality gap can be completely circumvented by abandoning simulators altogether—successful experiments in the 1990s thus evaluated the performance of each controller using a physical robot in a real-world arena. A promising approach to scale up to more complex behaviors is to use a population of robots instead of a single one [5, 43]; in such a situation, several evaluations occur in parallel, and therefore the evolutionary process can theoretically be sped up by a factor equal to the number of robots. However, in all such works only the controllers evolve, and any real-world fitness evaluations take place on a fixed robot morphology.

There are a handful of studies related to flexible morphologies. The Symbrion project did target flexible morphologies by developing robot units that could aggregate to form multicellular robotic organisms and then disaggregate again [29]. However, Symbrion organisms were transient constructs in that organisms could not reproduce and there was no evolution of morphologies; only the controllers were evolvable. Systems within artificial life have addressed the evolution of morphologies (and control systems) in simulated worlds with virtual creatures, for example, in the pioneering works by Sims, Bongard, and Pfeifer and several subsequent studies [1, 7]. This was brought closer to the physical by jointly evolving robotic shapes and control in a simulator and 3D-printing the evolved shapes [31]. However, only the final product was materialized; furthermore, the robot had no sensors. The self-reproducing machines of Zykov et al. were modular robots that were designed or evolved to be able to make exact clones of themselves without variation, and they did not undergo evolution in the real world [47, 48]. More recent work has evolved morphologies composed of novel, soft materials, but once again only the final morphologies were constructed, and in this case they were confined to operating within a pressurized chamber, rather than operating in a real-world population [23]. A related subfield of evolutionary design has concerned itself with constructible objects, but here again evolution generally has taken place in software, with only the end result being constructed. A few projects employed fitness evaluations on the hardware itself, but these systems produced inert objects that had no controllers and that passively underwent evolution by a traditional evolutionary algorithm [28, 36].

A separate category of related work contains biologically motivated studies with evolution implemented in populations of robots [19, 41]. Using real hardware is extremely challenging, and to the best of our knowledge there has been only one successful project, that of Long et al. investigating the evolution of vertebrae through swimming robot fish [8, 32]. The project faced huge practical problems; for instance, manual construction of new generations took weeks, and the team could only run a few experiments of a few generations each.

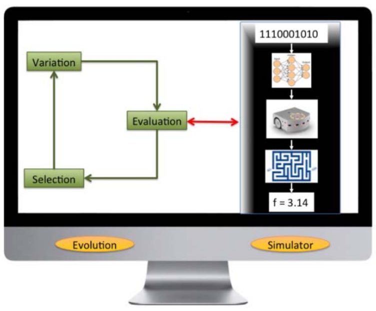
Finally, let us mention the two most important pieces of prior work for our project. The RoboGen system features modular robots as phenotypes, a corresponding space of genotypes that specifies the morphology and the controller of a robot, and a simulator that can simulate the behavior of one single robot in a given environment [1]. Evolution is implemented by a classic evolutionary algorithm that maintains a population of genotypes, executes selection, crossover, and mutation, and calls the simulator for each fitness evaluation. The system is applied to evolve robots in simulation, and physical counterparts of the simulated robots can be easily constructed by 3D-printing and manually assembling their components. RoboGen was not meant to be and has never been used to physically create each robot during an evolutionary process. The project that resembles ours the most concerns a "model-free implementation for the artificial evolution of physical systems"

[6, p. 2]. The robots are constructed from cubical modules that can be passive or active (driven by a servo). The robots do not have sensors and are driven by an external PC that communicates via Bluetooth. The evolutionary process is conducted by a centralized evolutionary algorithm running on the external PC in the same manner as in RoboGen. However, robot phenotypes are not tested in simulation, but constructed one at the time in real hardware for fitness evaluation, where fitness is the distance traveled in a given time interval.

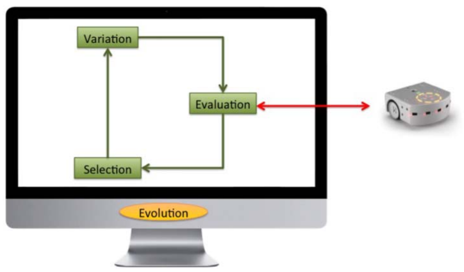
3 Robot Evolution in Real Time and Real Space

Robot evolution in real time and real space requires an appropriate system architecture. To distinguish different options and to position our system, let us consider three attributes and a couple of specific schemes to illustrate the matter.

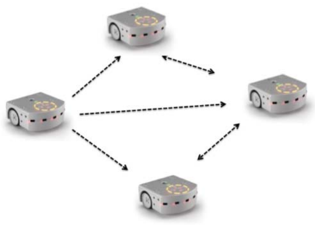
Physical versus virtual evaluations. The majority of evolutionary robotics follows scheme A shown in Figure 1, where the complete evolutionary process takes place in simulation. A handful of systems are based on scheme B, where some (usually not all) fitness evaluations are performed on real hardware.



(a) Scheme A is fully simulation based; a conventional centralized EA calls a simulator for every fitness evaluation.



(b) Scheme B relies on a conventional centralized EA that performs (some of) the fitness evaluations on real hardware.


(c) Scheme C: Embodied evolution of controllers in fixed robot bodies. There is no centralized evolution manager; selection and reproduction are regulated by the robots, and their interactions indicated by the arrows.

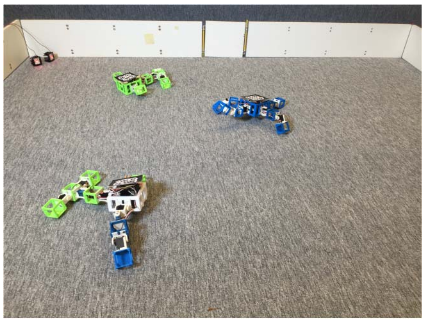

(d) Scheme D: Embodied evolution of controllers and bodies. There is no centralized evolution manager; selection and reproduction are regulated by the robots and their interactions.

Figure 1. Illustration of various system architectures for robot evolution.

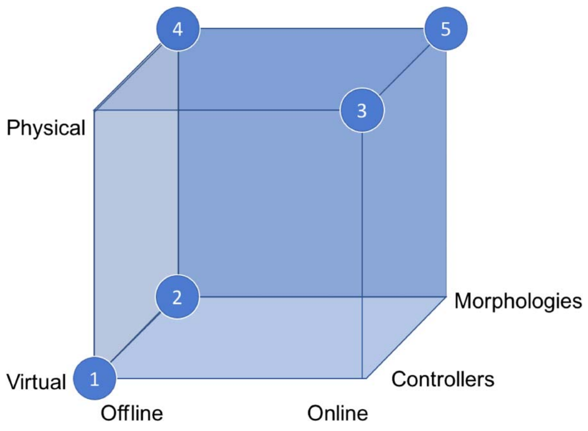


Figure 2. Positioning robot evolution systems. The place of the usual ER approach is shown by the black dot (no. 1): evolving robot controllers offline in simulation. The orange dot (no. 2) designates the works of Lipson and Pollack [30] and Auerbach et al. [1]: They employ offline evolution of robot morphologies in simulation. The embodied evolution system of Watson et al. [44] works with real robots to evolve good controllers online; hence it belongs to the green dot (no. 3). The work of Brodbeck et al. [6], where morphologies are evolved by a centralized offline EA using real-world fitness evaluations, can be positioned at the blue dot (no. 4). Finally, the system we are after sits at the red dot (no. 5): Robot morphologies (and the corresponding controllers) are evolved in an online fashion in real hardware.

Online versus offline evolution.[1] Evolutionary robotics mostly employs evolution in an offline fashion, that is, robots are evolved in the design stage and do not evolve further after deployment. The alternative is online evolution, a.k.a. embodied evolution [44], where robots undergo evolution during their operational period; see scheme C and scheme D in Figure 1.

Morphology versus controller evolution. Most articles on ER address the evolution of good controllers for given robot morphologies. Alternatively, the morphologies of the robots can be evolvable as well. In this case, morphologies and controllers evolve together.

In terms of these three attributes we can identify our system of interest as an online evolutionary system working with real robots for evolving morphologies. Although it can be argued that these attributes are not fully independent, it is helpful to visualize them as dimensions resulting in a cube, as shown in Figure 2. In this cube our envisioned system sits in the upper right rear corner opposite the huge majority of existing work in evolutionary robotics in the lower left front corner. The specific details are discussed in the next subsection.

3.1 System Design

A general architecture for evolving robots in real time and real space has been provided by the conceptual framework named the *triangle of life* [17]. This framework describes a life cycle that 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 robots. The result is a population of robots that evolves and thus adapts to the given environment. In the following we elaborate on this basic triangle and extend it with more details for a tangible implementation.

1 In general, we can distinguish the design stage and the operational stage of robots, separated by the moment of deployment. See [14, Chapter 17] and [15] for a further discussion.

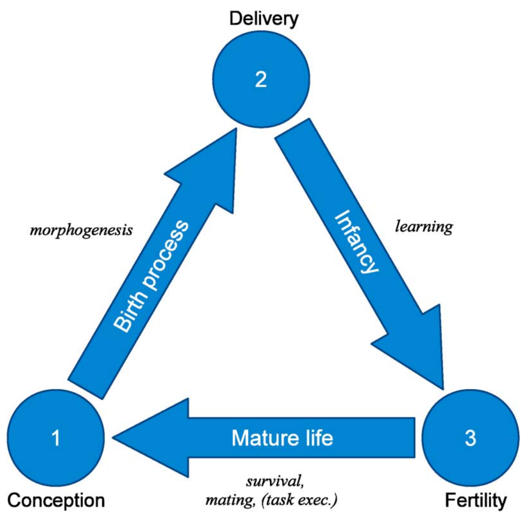


Figure 3. The triangle of life. The pivotal moments that span the triangle and separate the three stages are: (1) Conception: A new genome is activated, construction of a new robot starts. (2) Delivery: Construction of the new robot is completed. (3) Fertility: The robot becomes ready to conceive offspring.

It is important to note that evolutionary robots in the future could self-reproduce autonomously without humans in the loop. Therefore, we argue that distributed reproduction mechanisms (e.g., self-assembly or robotic equivalents of eggs and pregnancy) should be avoided and a safe system should have a central "kill switch" to stop reproduction if necessary. To this end, we choose to use a unique system component for the construction of new robots: the *production center* described below.

The triangle of life consists of three stages, morphogenesis, infancy, and mature life, as illustrated in Figure 3. A specific implementation consists of three components: the production center, the training center, and the arena that represents the world where the robots operate. The production center constructs robot phenotypes as specified by the given genotypes. New robots start as *infants* in the training center. Here they learn to control their own body (which may be different from the bodies of their parents) and to perform basic tasks under supervision—monitored by a camera+computer system and/or a human user. If a robot acquires the required set of skills, it is declared an *adult* and enters the arena, where it must survive, reproduce, and perform user-defined tasks; otherwise it is removed and recycled. The training center increases the chances of success in the arena and plays an important evolutionary role: It prevents reproduction of poorly performing robots. Lifetime learning continues in the arena, but now without centralized supervision, autonomously. Reproduction is driven by a mate selection mechanism (innate in the robots or executed by the human breeder) to identify two or more robots for parenting a child. The parents transmit their own genomes to the production center, where the genomes undergo crossover and mutation and the resulting new genome is used to construct a child robot.

Based on this generic design, we can now specify the ingredients of our online evolutionary system, working with real robots for evolving morphologies as follows:

1. A robot design and a genetic code that can specify such robots.
2. A construction procedure that starts with a genotype (code for a certain robot) and ends with a phenotype, a physical robot designated by the given genotype. This implements morphogenesis in the production center.

3. A learning method for infant robots to learn to use their own body adequately. This belongs to the infancy stage in the training center.
4. A reproduction mechanism that regulates mate selection and recombination of the parental genomes. This is the minimum to implement mature life in the arena.

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. The technology of 3D printers that can produce a fully functional robot is developing quickly, but it is still in an early stage [33, 45]. To mitigate this problem we have chosen the robot design featured in RoboGen, which combines 3D-printed components with prefabricated modules (e.g., CPUs, servos, batteries), and obtains the targeted robot by hand-assembling the parts according to the specification in the genome. This will be described in the next section.

We strive for simplicity in other components as well as for the purpose of the proof of concept. For instance, all we require in the training center is to learn a good gait for the given body, and we do not implement a task to be performed in the arena. From this perspective, our system will be natural: The sole purpose of the robots is survival and reproduction. As shown in [5], such systems can exhibit very interesting evolutionary dynamics, even if only the controllers are evolvable. The relevant details for our system will be explained in Section 5.

4 Design Decisions and Exploratory Experiments

This section sets out the major design decisions that provide the basis for the implementation as presented in Section 5. To validate the choice of physical substrate (outlined in Section 4.1), we performed exploratory experiments that consider the evolution of morphologies in an online setting. Secondly, we performed a set of experiments as a basis for selecting a suitable method for online learning of robot gaits in arbitrary morphologies.

Experimentation with real hardware consumes a lot of time and resources; therefore, we rely on simulation to test our ideas and inform our design decisions. These exploratory experiments rely on the Revolve simulator that was developed specifically for the simulation of collectives of modular robots composed of RoboGen [1] modules. Revolve is based on the Gazebo simulator and implements a set of extension tools that aim to provide a convenient way to set up this kind of experiments. Source code for the revolve simulator can be found at https://www.github.com/ElteHupkes/revolve.

4.1 Robot Design

The design of robots and their genetic representation is based on RoboGen [1]. The robots are constructed from basic 3D-printed modules, and each robot's genotype describes its layout. The genotype is composed of a tree structure, with the root node representing a core module from which further components branch out. Robot bodies consist of three types of component: *fixed bricks*, a *core component*, and *active hinges*. These components are depicted in Figure 4. The original

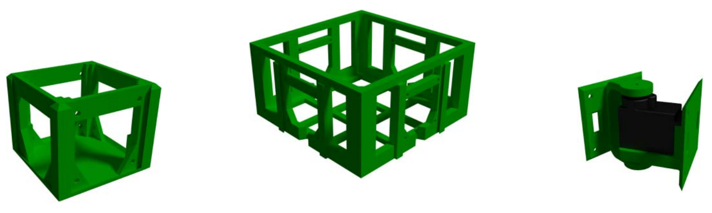
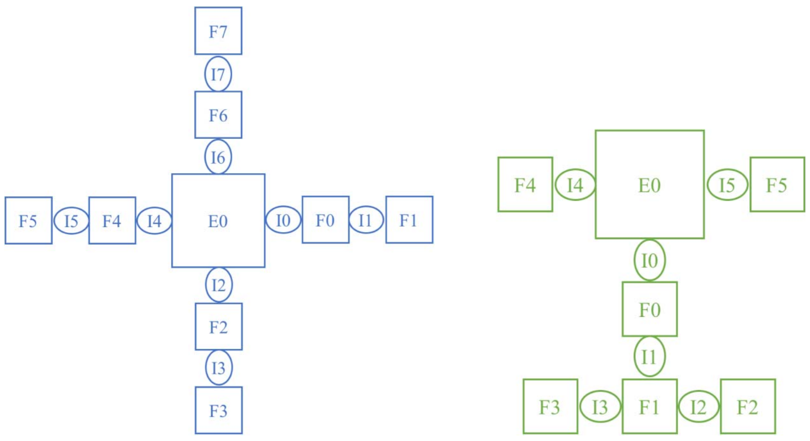

(a) Fixed brick (b) Core component (c) Active hinge

Figure 4. The 3D-printable robot components.



(a) Spider   (b) Gecko

Figure 5. Schematic diagram of two robot morphologies with core component (labelled E0), active hinges (I*x*), and fixed bricks (F*x*). Full listings of the respective genomes are provided in Appendix 2 (Listings 1 and 2).

RoboGen framework includes more components, but these are omitted from the present study. The core component houses the robot's Raspberry Pi microcontroller [35], and the active hinges contain servo motors (see Section 5 for more details). For complete details of this specification and example listings, refer to Appendix 2.

For easy identification of the heredity of each robot's morphology, the standard RoboGen specification was extended to include a color for each component. This allows the morphological traits that a robot inherits to be easily attributed to either parent by matching colors.

Figure 5 shows a schematic representation of two manually designed robot morphologies in this scheme.

4.2 Online Evolution of Morphology

To validate the choice of RoboGen as the physical substrate and genetic representation for online evolution of robot morphology, we conducted experiments where a population of simulated robots coexist in a featureless arena and are centrally evaluated and selected.

Parent and survivor selection is performed on the basis of the robots' locomotive performance, calculated as

$$f = v + 5 \cdot s, \tag{1}$$

where v is the length of the path the robot has traveled over the last 12 s, and s is the length of a straight line from the beginning to the end of that path. This value is continuously updated. Robots can only be selected when they are *mature*, that is, after running for at least 15 s.

The population is seeded with 15 randomly generated robots that are spread throughout the environment. The population is culled every 30 s by removing any robots that have a fitness less than a fraction 0.7 of the mature population mean, but a minimum of 8 robots is maintained to ensure variation and prevent extinction. If the population reaches 30 individuals and no individuals match the culling criterion, the 70% least fit robots in the population are removed to prevent convergence.

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.

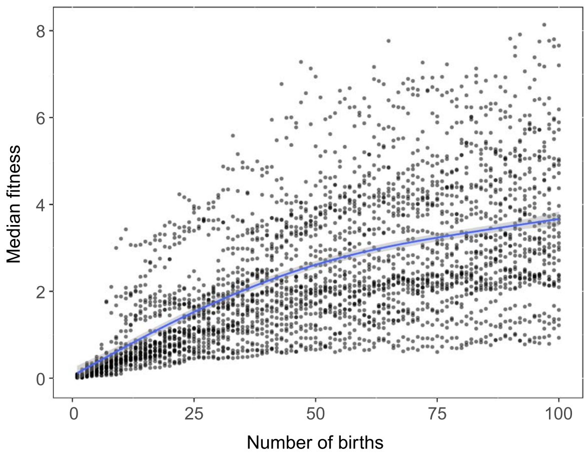


Figure 6. Fitness progression of 30 replicate runs, with the number of birth events as time scale. The *y* axis shows the fitness as defined by Equation 1. Each point represents a median value of an entire population within a birth time frame. The blue curve shows the median trend for 30 replicate runs.

The robots run artificial neural network controllers that evolve in conjunction with their morphologies. RoboGen prescribes robot controllers that are based on fully connected, recurrent artificial neural networks, and these are represented in the genome.

This is obviously not an accurate representation of the triangle of life, as selection occurs centrally and local selection schemes will lead to different levels of selection pressure. However, it does allow us to gauge the suitability of the selected substrate in an online setting.

Figure 6 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.

4.3 Online Learning

Learning is an important aspect of the triangle-of-life conceptual model. In the triangle of life, evolution is not just an optimizer of some robot features, but a force of continuous and pervasive adaptation. New individuals are likely to be morphologically different from their parents, and therefore any recombined controllers that they inherit may not suit their bodies as a matter of course. Therefore, every newly created robot needs to learn to control its own body, necessitating online individual learning capabilities.

Earlier work identified RL PoWER [26] as a reliable and efficient algorithm for gait learning in arbitrary morphologies with modular robots consisting of homogeneous modules [9]. We verified these findings 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. This resulted in significantly better performance for a similar convergence time [25]. A detailed description of the resulting learning algorithm is given in Appendix 4.

Table 1 provides configuration settings for the algorithm as it was applied. As an illustration of the findings reported in [25], Figure 7 shows the results for gait learning in two robot shapes. The shapes considered in this excerpt are those with the manually designed *spider* and *gecko* shapes shown in Figure 5. Both shapes show an improvement from ≈1.0 m/s up to ≈3.5 m/s after 100 evaluations of 30 s each. The results of the experiments with RL PoWER indicate that it is an appropriate choice for online learning of gaits.

Table 1. RL PoWER online settings.

Parameter	Description
Environment	Infinite flat plane
Type	Two-parent selection with binary tournament
Evaluation	Fitness measured at the end of each spline evaluation period[a]
Initial spline size	3
Maximum spline size	20
Interpolated spline size	100
Evaluation rate (s)	30
Population size	10
Variance	0.008
Variance decay	0.98
Maximum evaluations	1,000

Note. [a]Fitness is measured over a 30-s sliding time window. See Equation 3.

5 Implementation in Hardware

This section describes the real-world instantiation of the triangle-of-life concepts based on the choices outlined in the previous section. As explained in Section 4.1, for reasons of simplicity, we base our design on the well-established RoboGen system [1].

To express a genotype, that is, to physically instantiate a robot, components of the appropriate colors are printed and manually assembled with added electronics as described in Section 5.1. The

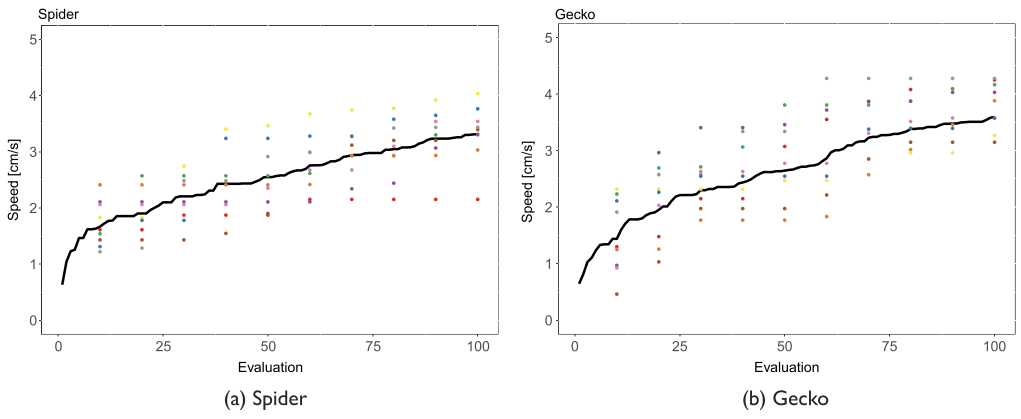


Figure 7. Development of locomotion speed for two robot shapes over 10 replicate runs. The colored dots denote individual runs; the black curve indicates the average of all 10 runs.

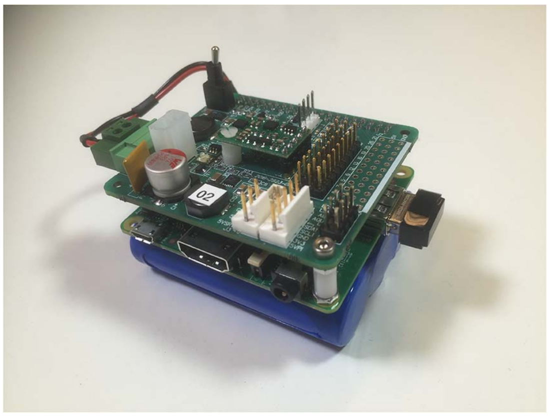


Figure 8. Raspberry Pi with hat board design on top and battery mounted below.

original RoboGen specification provides for Arduino microcontroller boards to host the controller code and operate the robots. We use the Raspberry Pi [35] (Figure 8) instead, as it offers greater flexibility, allowing Python-based controller code. This substantially increases the speed and ease of software development and debugging, and offers a wide selection of libraries. The Raspberry Pi runs the robot controller as well as further implementation logic (e.g., communication protocols).

The microcontroller is extended with a *hat* extension board that was created to save space inside the core module and ease robot assembly. It contains two dc-to-dc converters (one for the Raspberry Pi, one for the servos) to adjust the voltage given by the battery to the voltage needed by the components, an inertial measurement unit (IMU), a power switch, an I^2C adapter (for the photosensors), and pins that connect directly to general-purpose input-output (GPIO) pins on the microcontroller, used to control the servos and the light.

In order to accommodate the Raspberry Pi and hat assembly, we enlarged the size of the core module. All the code used in this project, including the design files for 3D-printing the modules and the hat design, is available at https://github.com/ci-group/revolve/hw.

5.1 Morphogenesis

Morphogenesis is, for the current proof of concept, a manual process of robot assembly where a robot's genome is expressed. In the long-term vision, this eventually is also an automated process, but the development of an automated robot assembly is beyond the possibilities of the current proof of concept.

First, the modules of the body are 3D-printed on the Flashforge CreatorPro printer using a fused-deposition modeling technique in which an element is built by laying down a plastic filament, layer by layer. The printed components are then assembled according to the layout specified in the genotype; active hinges (Figure 4, right image) are equipped with servomotors, and the connector cables of those are threaded through the body to the core module. After fitting the servo inside the hinge and connecting it to the robot, it is set into a neutral position (0°), the hinge is adjusted to form a straight joint, and then the position is fixed with screws. Once activated, the servos' angles are controlled by sending pulse-width-modulated signals to set them at a specific angle (in the range of −45° to 45°) and maintain that position.

The extension board and battery are connected to the microcontroller, and the assembly is inserted into the core module and connected to the servos' connector cables. The core module is then covered with a fiducial symbol used for tracking (see Section 5.2.1).

Finally, a copy of the controller code is copied to the robot and configured by means of a JSON-like formatted configuration file. This file contains parameters for gait learning (see Table 1),

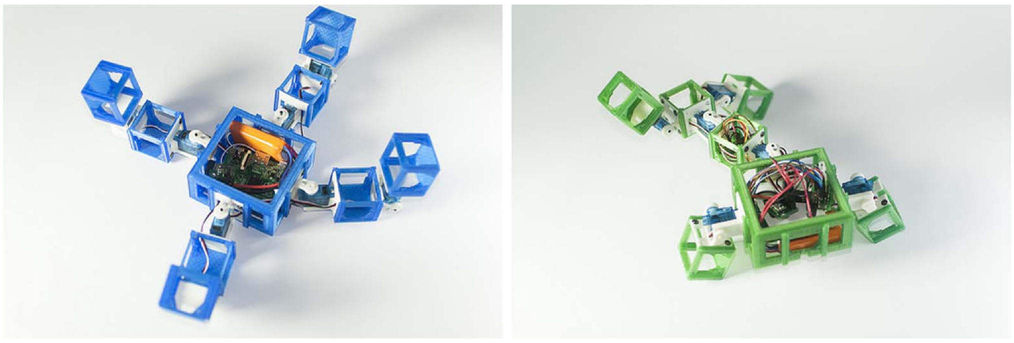


Figure 9. The real robots: the spider (left) and the gecko (right).

connection parameters for tracking and mating servers, the hardware configuration of the robot, and additional information that may vary from one robot to another, such as its name or the ID of the tracking symbol used.

The robot is then placed in the arena, and the learning process that represents the infancy phase is activated. Figure 9 shows two assembled robot bodies corresponding to the genomes depicted in Figure 5.

5.2 Infancy

The infancy phase of this proof-of-concept implementation consists of the robots learning a gait that allows for efficient locomotion. What a suitable gait for a particular body plan looks like is hard to determine a priori, and it is likely that gaits that work well for a robot's parent have to be substantially adapted for the child's morphology. Therefore, we opted to implement non-Lamarckian evolution: The learned gaits are not inheritable, and each individual has to develop them from scratch.

5.2.1 Localization

To provide feedback for the learning process, an overhead camera and attached localization server track the robot's position using ReacTIVision.[2] This software implements tracking of fiducial markers (Figure 10) that are printed on the cover of each robot. The software captures output from the camera, analyzes it to find the markers' positions on the screen, and sends the information about the ID and the position to a client application. In our case, the client application is a fitness localization server that keeps track of the robots' trails and serves a fitness evaluation to the robots themselves in order to let them evaluate their walking abilities. This server acts as a virtual sensor that allows the robots to assess the performance of candidate gaits.

The localization module in the robot communicates with the localization server through a predefined protocol:

Start tracking. Start tracking the robot and set current fitness to 0.

Get current position. Retrieve the current position of robot in the arena.

Get current fitness. Retrieve the displacement between the start of tracking and the current position, and the sum of displacements between consecutive positions reported by ReacTIVision.

2 http://reactivision.sourceforge.net/

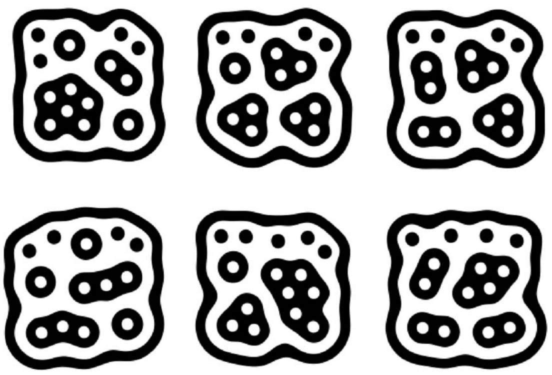


Figure 10. ReacTIVision markers used for tracking of the robots, representing IDs from 0 to 5 (left to right, top to bottom).

The tracking system uses a QuickCam Pro 9000 camera, mounted on an aluminum scaffolding frame 2.5 m above the arena, pointing straight downwards. The camera is connected to a computer running the ReacTIVision 1.5 software and the tracking-fitness server.

5.2.2 Learning

As explained in Section 4.3, 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. Appendix 4 provides details of the RL PoWER algorithm that the robots employ to achieve online learning of locomotion. We implement RL PoWER to allow for online learning of gaits for arbitrary morphologies: The robots are controlled through a set of splines that define an open-loop gait as described in [9]. This use of sets of cyclic spline functions was taken from [38] and is detailed in Appendix 3. Since our aspiration is to compare the results of gait learning of robots in real life with that in simulation, we replicate exactly the RL PoWER implementation that was tested in simulation.

The robot controller was implemented in C++ and Python and is composed, in addition to the splines-based brain, of two modules: one for localization (see Section 5.2.1) and one for communication with other robots and the mating server (see Section 5.3.2). The robots were tested with five experimental runs each in the 2 × 2-m arena. After each run, the battery was replaced with a recharged one, and the robot "brain" was reset.

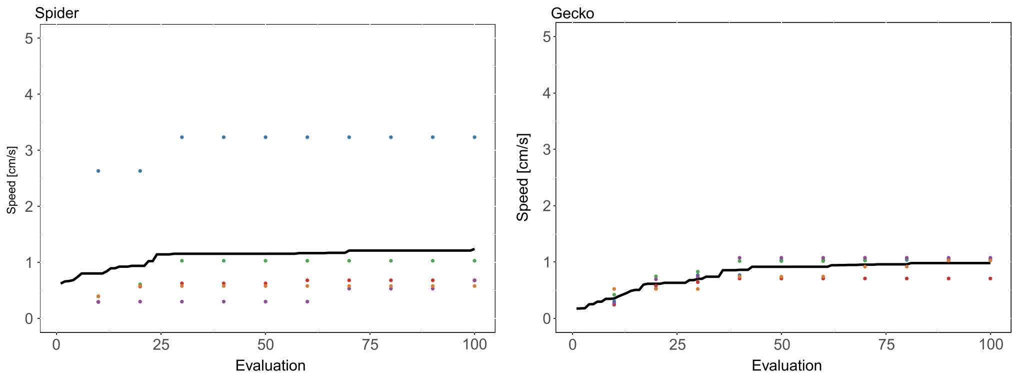


Figure 11. Results from learning in hardware for the spider (left) and the gecko (right).

Figure 11 shows the results from testing the learning algorithm 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 breaking—on average 1.5 motors per run; (2) the weight of a robot's head significantly influences the performed gait; (3) robots were bounded in a small arena rather than the infinite plane in 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.

5.3 Mature Life

Once robots reach the mating area, they are deemed fertile and can communicate and exchange genomes with other robots to create offspring. At the beginning of their life, robots are intentionally placed far from the mating area so that only robots that achieve efficient locomotion capabilities can reach the fertile state and generate new offspring.

The mating area is defined by two stationary red LEDs, situated in a corner of the arena. When a robot's photosensor detects red light above a certain intensity threshold, the robot starts emitting and listening for mate request messages. In our experiments, the threshold itself is not in any way adaptive or evolved, but remains static throughout the whole mating process. This ensures that the robots will eventually meet and reproduce in a reasonable amount of time.

When the two robots are in the mating area and successfully exchange mating requests, they agree to mate. This implies that the robots do not actively pursue or even perceive each other, an addition that will be possible when future robots are equipped with cameras.

5.3.1 Selection and Reproduction

The current implementation has only the most basic selection mechanism: When a potential mate is available, a robot will agree to produce offspring. With only two robots available, more sophisticated selection procedures would make no sense. However, when the setup is extended to contain more individuals, selection schemes can be considered that take any aspect, ranging from task performance to genetic suitability, into account.

Recombination and mutation of genomes is implemented through the standard operators defined in RoboGen. As illustrated in Section 4.1, the morphologies of the robots can be represented as trees, the nodes of which are hardware components, with a shape, a color, and a potential functionality (e.g., wheel rotation). Therefore, conveniently, the recombination and mutation operators that may be used are well established in genetic programming practice [3].

More specifically, the recombination of parent genomes is implemented as random subtree exchange; the parameters of these operations are defined in the robot startup configuration file. Since the tree genome representation is nonlinear, we may run into the problem of the tree becoming too large, and that is the reason why experimenting and tweaking these random parameters is essential. A common practice is to let them evolve; for simplicity, however, we do not do this in our implementation.

Unlike the recombination operator, the mutation operator is applied directly in the offspring, and replaces a randomly selected subtree with a randomly generated tree of the same kind—according to the format RoboGen defines. Once again, the random parameters may be tweaked, but it needs to be kept in mind that mutations need to be minor and happen rarely; Banzhaf et al. suggest a mutation rate of 0.05 or lower [3]. Refer to Appendix 1 for a more detailed and implementation-specific illustration of our recombination and mutation operators.

5.3.2 Mating Protocol

The mating protocol involves the robots and the mating server. The communication between the robots happens in a distributed manner, so selection is distributed and localized, while recombination and the subsequent morphogenesis phase are centralized. This is analogous to a population living and mating in an ecosystem, while their offspring are born in a central clinic.

When a mating sequence initiates, the robots start communicating over the wireless network. Their communication involves states—*learning*, *evaluating*, *ready to mate*, and *initiate mating*—and every interaction they have may affect or change the state of each robot. *Learning* indicates the state of infancy, in our case developing the ability to walk towards the light source. *Evaluating* is the short-period state when the robot receives a fitness feedback from the localization server. During this process there is a constant evaluation loop that determines whether the learning goal has been achieved, and essentially decides whether the robot is ready to mate or not. When the robot approaches the red light source closely enough so that the photocell receives a light signal over the threshold of 85% of the maximum measurable intensity, it initiates the *ready to mate* state. If two robots initiate this same state and finish their handshake communication successfully, they transition to *initiate mating* state. The basic communication algorithm follows the three-way-handshake pattern and is described in Algorithm 1.

The protocol starts when a robot declares its availability to mate by broadcasting a message to the wireless network. The robot then listens to the same network in order to discover messages sent by other robots of the same kind. Once a suitable mate is identified, the robot transmits the mate's ID. If each robot receives a message with its own ID in return, the agreement is complete, and both robots transmit their genomes to the mating server.

The mating server module listens on the wireless network and accepts TCP packets with genomes and unique IDs of the robots. The logic of this workflow implies that the robots have autonomously come to an agreement, so the mating server does not need to perform any further checks: The decision of finding a good mating partner is made by the robots themselves without

Algorithm 1. Communication protocol.

```
this ← Robot:object(id, state);
this.state ← ready to mate;
available ← true;
while available do
    broadcast(this.state);
    messages ← receive_messages();
    if ready to mate in messages then
        this.state = finding mate;
        broadcast(this.id, potential mate id);
    end
    if this.id in messages then
        this.state = mate found;
        broadcast(this.id, this.state);
        available = false;
        send_genome(mating server);
    end
end
```

referral to any central authority. Once a pair of genomes is complete, the mating server recombines them, applies variation operators, and produces the offspring genome, ready for morphogenesis, completing the triangle of life.

5.4 The Life Cycle

We have integrated the components described in the foregoing to execute one life cycle. To this end, we built a robot habitat where the training center and the arena were not separated. That is, we use the same space for educating the infants and having the adults meet and mate. This habitat is a bounded 2.2 × 3.0-m rectangle lined with a gray carpet. The boundaries are made of wood to prevent the robots from escaping the area.

The cycle started with two genotypes shown in Listing 1 and Listing 2 (Appendix 2). These genotypes were used to carry out the morphogenesis process that resulted in the blue and green robots shown in Figure 9. Printing and assembling all components took approximately a day per robot. The speed of the 3D printer is a crucial factor here; in our case 3 h was needed to print one block. The initial population consisted of two robots, the spider and the gecko, as shown in Figure 12 (left). The extended population shown in Figure 12 (right) is explained further in Appendix 2.

The infancy stage focused on gait learning. Because controllers of newborn robots are randomly initialized, their first task is to learn effective locomotive behavior. This happened under supervision of the overhead camera system as outlined in Appendix 4. The robots are equipped with a photocell that allows them to detect a light source placed at the edge of the arena. The robots can gauge the efficacy of their controllers by monitoring the intensity of the light source. If the robots succeed in learning to locomote, they will reach the light source. Once they are close enough (again indicated by light intensity), they are deemed fit and mature enough to procreate and change their status from infant to adult.

Due to our simplified setup, the difference between the arena for adult robots and the training center for infant learning is only conceptual. In fact, they are the same physical space. Hence, adult robots need not move to another location; they can start communicating to carry out mate selection. Here again, we had to simplify the system and make the selection criterion void—after all, there was only one option for choosing a partner. Thus, the two would-be parents passed each other's test by default and decided to exchange genetic material. The robots transmitted their genomes to a server that recombined the received genomes and produced the code for a new individual. (Note that the server is only a channel of communication; it does not constitute a central overseer of the evolutionary process.) The new genome is exhibited in Listing 3 (Appendix 2).

A new morphogenesis process started with this genome; component parts were printed and assembled. The image on the left-hand side of Figure 13 shows the result, the first robot baby parented by two parent robots in a real-world (not simulated) environment.

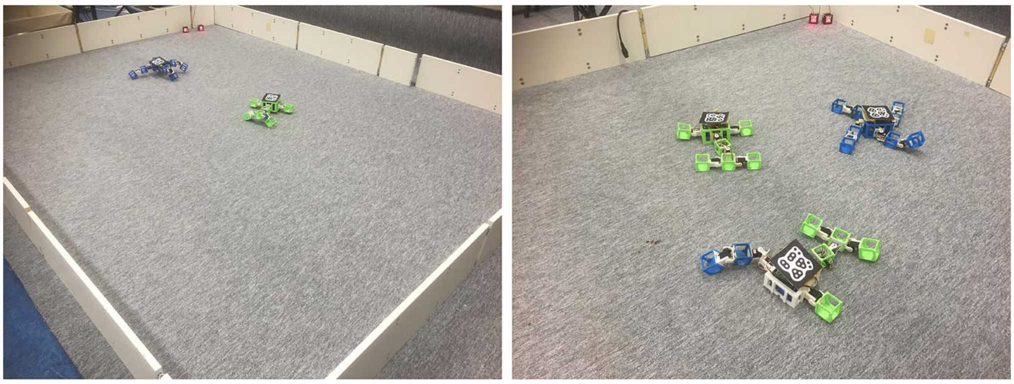


Figure 12. Overview of the entire habitat and the initial population consisting of the spider and the gecko (left) and the extended population with the parents and the offspring in the habitat (right).

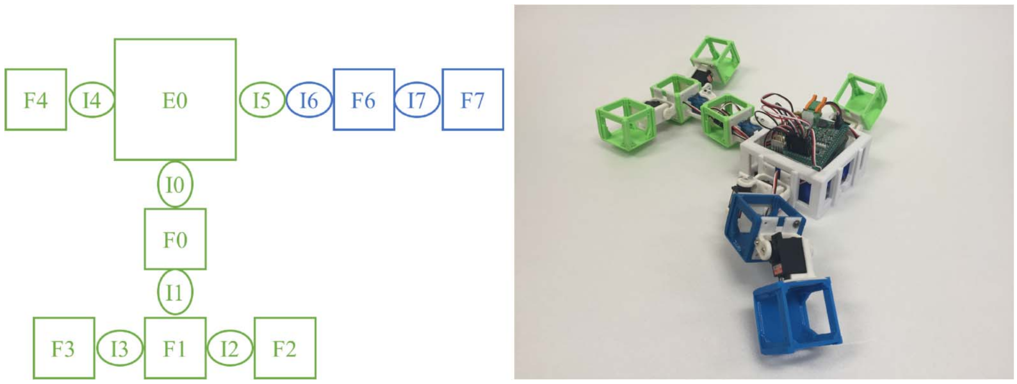


Figure 13. Schematic diagram of the newly assembled robot baby (left) and close-up of the physical robot (right).

6 Discussion and Possible Extensions

The project we have described above is a proof of concept. At the cost of several simplifications we have demonstrated the feasibility of physically evolving robots, validated the suitability of an instantiation of the triangle of life, and obtained insights into the related challenges.

It is important to note that we did not aim at creating an evolving robot population, because it would require several consecutive generations with many selection and reproduction cycles. Instead, we demonstrated one reproduction cycle, which is the basic unit to be repeated for evolution. The amount of handwork was significant, but we argue that this does not invalidate the main concept; it merely reflects the current level of technology and the resources available to us at the moment.

In the following we discuss the workflow for creating physically evolving robot populations and elaborate on its elements, based on the know-how we have obtained through this project. The workflow consists of the following steps:

1. Define the makeup of the robots and a genetic code that can specify such robots.
2. Establish a production procedure that starts with a genotype (code for a certain robot) and ends with a phenotype, a physical robot designated by the given genotype.
3. Set up a learning method for infant robots to learn a set of basic skills under supervision.
4. Implement a reproduction mechanism through (a) a policy that regulates mate selection and (b) a recombination operator that works with the parental genomes as defined by the given genetic code. Specify a task to be performed by the robots and ways to measure task performance (this is optional). Equip robots with learning abilities in a nonsupervised fashion (this is optional).

When designing the robot makeup we can distinguish the morphologies (bodies, hardware) and the controllers (brains, software). Regarding the morphologies, two aspects play an essential role. First, the components of the robots. The RoboGen system we used here is based on the idea of combining 3D-printable and prefabricated components. This provides a practicable approach, and any specific system can be easily extended through adjusting either type of components. For instance, we can make the dimensions of the 3D-printed blocks evolvable, allow the use of flexible plastic that bends, or add different sensors and cameras. Such extensions enrich the design space and make the set of possible robot morphologies and behaviors larger. The second aspect concerns the constructibility of the robots. In our current system robots are constructed by hand. This is a practical shortcut for an academic project, but advanced technologies for automating the assembly of machines are available, for instance, in the car industry. In a more advanced follow-up project such technologies might be employed to reduce the role of human involvement and to increase the speed of (re)production.

Concerning the controllers, the user has an extra decision to make regarding the evolvable and learnable features. In general, the properties of the robots can be divided into three categories: fixed (e.g., the property that the controllers are neural networks), evolvable (e.g., the structure of the neural network that controls the robot), and learnable (e.g., the weights inside the neural network). In a non-Lamarckian system the set of evolvable features is disjoint from the set of learnable ones and it is up to the experimenter to determine which features will be inheritable/evolvable and which ones will be subject to learning. In our current project we use one spline-based controller with learnable parameters for each servo motor. Thus, the controllers do not contain evolvable parts, although one could argue that inheritance plays an implicit role via the morphologies in that the number and the position of the servo motors was evolvable.

The supervised learning in the training center serves to equip the robots with sufficient skills to earn the status of a fertile adult. In our current version this was limited to gait learning, which is fundamental for a system where a new robot can have a new morphology that differs from the morphologies of the parents. Our corresponding fertility test was also simple. By walking to the red lights, the robots proved to be good enough to reproduce. For any practically useful system the set of skills needs to be larger. The specific list of skills will depend on the application at hand, but it stands to reason that it should at least include gait learning, obstacle avoidance, directed locomotion, foraging (recharging), and the recognition of other robots (possible mates). Whether or not learning multiple skills can be best done sequentially or in parallel is an open issue with contradicting advice in the literature [2, 37]. Another aspect concerns the separation of the training center and the arena. Our current system was simple, as we did not have physically separated compartments. However, in general the training center can be separated from the arena to contain the necessary monitoring and feedback facilities required for supervised learning. Furthermore, the environmental conditions can be made easier than in the arena to allow for gradual development, and the training center can be composed of different sections belonging to different modules of the total learning syllabus.

The fourth stage in our workflow belongs to reproduction, task execution, and learning in the arena. Also here, we (over)simplified things in our proof-of-concept project. The fact that we only had two robots implied that the mate selection policy was trivial: Accept the other robot without conditions. In a more realistic system, robots should prefer mating partners with desirable properties. These can be indicators of viability and/or based on task performance if the robots have tasks to carry out. As mentioned in Section 3.1, a task is not required to obtain an interesting robot evolution system, but it is natural to expect that the majority of future applications will concern robots that do something useful.[3] This raises the issue of interfacing the task to the robot population. By the nature of such systems this can be done through the fitness function that drives evolution and the reward functions used in the learning mechanisms (in the training center as well as in the arena). A fundamental issue here is how to combine environmental selection towards viability and task-based selection towards utility. A suitable approach is presented in [21].

Last but not least, let us consider the technical possibilities of producing several consecutive generations with many selection and reproduction cycles. This would require higher production capacity, recharging stations, and some definition of *death*, that is, determining when to remove and recycle a robot. With our current system, the production of a new robot takes about a day and the gait learning process needs 20 to 30 min. Most of the time to produce a new robot is spent printing, but more and faster printers can be used, the components can be printed ahead of time, and the production time would then only involve assembly of the components. Thus, it would be feasible to produce at least six robots per day. In a month's time 100 robots would be feasible. These need not

3 This is not necessary for biologically motivated studies, where the only challenge is to survive and reproduce.

exist simultaneously: Some will fail and be recycled. We estimate that the population size would not exceed 40 robots at any given moment. Alloting 2 m^2 per robot, the arena should be enlarged to about 80 m^2. All in all, we deem this feasible; the limitations are mainly practical.

7 Concluding Remarks

In this article we take a modest step towards systems of physically evolving robots. Our long-term vision foresees entire robotic ecosystems that evolve and work for long periods in challenging environments without the need for direct human oversight. Possible examples include robot colonies for monitoring remote regions on Earth, ore mining at extreme depth, and terraforming on other planets.

The current state of the art is very far from this vision. The field of real-world robot evolution is in a nascent state, and the system we describe here is a very rudimentary implementation. Nevertheless, this article contains three contributions to this area. First, we discuss a system architecture for physically evolving robot populations, based on the triangle-of-life model [17]. Second, we describe a proof-of-concept implementation to demonstrate how the three stages of the generic model can be realized, though in a simplified form, and connected into one life cycle. Third, we review the lessons learned from this implementation and identify important issues for further research and development.

References

1. Auerbach, J., Aydin, D., Maesani, A., Kornatowski, P., Cieslewski, T., Heitz, G., Fernando, P., Loshchilov, I., Daler, L., & Floreano, D. (2014). RoboGen: Robot generation through artificial evolution. In H. Sayama, J. Rieffel, S. Risi, R. Doursat, & H. Lipson (Eds.), *Artificial life 14: Proceedings of the Fourteenth International Conference on the Synthesis and Simulation of Living Systems* (pp. 136–137). Cambridge, MA: MIT Press.
2. Auerbach, J. E., & Bongard, J. C. (2014). Environmental influence on the evolution of morphological complexity in machines. *PLoS Computational Biology*, *10*(1), e1003399.
3. Banzhaf, W., Nordin, P., Keller, R., & Francone, F. (1998). *Genetic programming: An introduction*. San Francisco: Morgan Kaufmann.
4. Bongard, J. C. (2013). Evolutionary robotics. *Communications of the ACM*, *56*(8), 74–83.
5. Bredeche, N., Montanier, J.-M., Liu, W., & Winfield, A. F. (2012). Environment-driven distributed evolutionary adaptation in a population of autonomous robotic agents. *Mathematical and Computer Modelling of Dynamical Systems*, *18*(1), 101–129.
6. Brodbeck, L., Hauser, S., & Iida, F. (2015). Morphological evolution of physical robots through model-free phenotype development. *PLoS One*, *10*(6), e0128444.
7. Cheney, N., MacCurdy, R., Clune, J., & Lipson, H. (2013). Unshackling evolution: Evolving soft robots with multiple materials and a powerful generative encoding. In C. Blum (Ed.), *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation, GECCO '13* (pp. 167–174). New York: ACM.
8. Cho, A. (2014). The accidental roboticist. *Science*, *346*(6206), 192–194.
9. D'Angelo, M., Weel, B., & Eiben, A. E. (2013). Online gait learning for modular robots with arbitrary shapes and sizes. In A.-H. Dediu, C. Martín-Vide, B. Truthe, & M. A. Vega-Rodríguez (Eds.), *Theory and practice of natural computing* (pp. 45–56). Berlin, Heidelberg: Springer.
10. Doncieux, S., Bredeche, N., Mouret, J.-B., & Eiben, A. (2015). Evolutionary robotics: What, why, and where to. *Frontiers in Robotics and AI*, *2*(4).
11. Doncieux, S., Mouret, J.-B., Bredeche, N., & Padois, V. (2011). Evolutionary robotics: Exploring new horizons. In S. Doncieux, N. Bredèche, & J.-B. Mouret (Eds.), *New horizons in evolutionary robotics: Extended contributions from the 2009 EvoDeRob Workshop* (pp. 3–25). Berlin, Heidelberg: Springer.
12. Eiben, A., Kernbach, S., & Haasdijk, E. (2012). Embodied artificial evolution. *Evolutionary Intelligence*, *5*(4), 261–272.
13. Eiben, A., & Smith, J. (2015). From evolutionary computation to the evolution of things. *Nature*, *521*(7553), 476–482.
14. Eiben, A., & Smith, J. (2015). *Introduction to evolutionary computing*, 2nd ed. Berlin, Heidelberg: Springer.

15. Eiben, A. E. (2014). In vivo veritas: Towards the evolution of things. In T. Bartz-Beielstein, J. Branke, B. Filipič, & J. Smith (Eds.), *Parallel problem solving from nature—PPSN XIII* (pp. 24–39). Berlin, Heidelberg: Springer.

16. Eiben, A. E. (2015). EvoSphere: The world of robot evolution. In A.-H. Dediu, L. Magdalena, & C. Martín-Vide (Eds.), *Proceedings, Theory and Practice of Natural Computing: Fourth International Conference, TPNC 2015* (pp. 3–19). Berlin, Heidelberg: Springer.

17. Eiben, A. E., Bredeche, N., Hoogendoorn, M., Stradner, J., Timmis, J., Tyrrell, A., & Winfield, A. F. T. (2013). The triangle of life: Evolving robots in real-time and real-space. In P. Liò, O. Miglino, G. Nicosia, S. Nolfi, & M. Pavone (Eds.), *Advances in artificial life, ECAL 2013* (pp. 1056–1063). Cambridge, MA: MIT Press.

18. Floreano, D., Husbands, P., & Nolfi, S. (2008). Evolutionary robotics. In *Springer Handbook of Robotics* (pp. 1423–1451). Berlin, Heidelberg: Springer.

19. Floreano, D., & Keller, L. (2010). Evolution of adaptive behaviour in robots by means of Darwinian selection. *PLoS Biology*, *8*(1), e1000292.

20. Glette, K., Klaus, G., Zagal, J. C., & Torresen, J. (2012). Evolution of locomotion in a simulated quadruped robot and transferral to reality. In *Proceedings of the Seventeenth International Symposium on Artificial Life and Robotics* (pp. 1–4).

21. Haasdijk, E., Bredeche, N., & Eiben, A. E. (2014). Combining environment-driven adaptation and task-driven optimisation in evolutionary robotics. *PLoS ONE*, *9*(6), e98466.

22. Hemker, T., Sakamoto, H., Stelzer, M., & von Stryk, O. (2006). Hardware-in-the-loop optimization of the walking speed of a humanoid robot. In *CLAWAR 2006* (pp. 614–623).

23. Hiller, J., & Lipson, H. (2012). Automatic design and manufacture of soft robots. *IEEE Transactions on Robotics*, *28*(2), 457–466.

24. Jakobi, N., Husbands, P., & Harvey, I. (1995). Noise and the reality gap: The use of simulation in evolutionary robotics. In F. Moran, A. Moreno, J. Merelo, & P. Chacon (Eds.), *Advances in artificial life* (pp. 704–720). Berlin, Heidelberg: Springer.

25. Jelisavcic, M., De Carlo, M., Haasdijk, E., & Eiben, A. (2016). Improving RL power for on-line evolution of gaits in modular robots. In *2016 IEEE symposium series on computational intelligence (SSCI)* (pp. 1–8). New York: IEEE.

26. Kober, J., & Peters, J. (2009). Learning motor primitives for robotics. In *2009 IEEE International Conference on Robotics and Automation* (pp. 2112–2118). New York: IEEE.

27. Koos, S., Mouret, J.-B., & Doncieux, S. (2013). The transferability approach: Crossing the reality gap in evolutionary robotics. *IEEE Transactions on Evolutionary Computation*, *17*(1), 122–145.

28. Kuehn, T., & Rieffel, J. (2012). Automatically designing and printing 3-D objects with evoFab 0.2. In M. A. Bedau (Ed.), *Artificial life 13* (pp. 372–378). Cambridge, MA: MIT Press.

29. Levi, P., & Kernbach, S. (Eds.) (2010). *Symbiotic multi-robot organisms: Reliability, adaptability, evolution.* Berlin, Heidelberg, New York: Springer.

30. Lipson, H., & Pollack, J. B. (2000). Automatic design and manufacture of robotic lifeforms. *Nature*, *406*, 974–978.

31. Lipson, H., & Pollack, J. B. (2000). Automatic design and manufacture of robotic lifeforms. *Nature*, *406*, 974–978.

32. Long, J. (2012). *Darwin's devices: What evolving robots can teach us about the history of life and the future of technology.* New York: Basic Books.

33. MacCurdy, R., Katzschmann, R., Kim, Y., & Rus, D. (2015). Printable hydraulics: A method for fabricating robots by 3D co-printing solids and liquids. arXiv preprint arXiv:1512.03744.

34. Nolfi, S., & Floreano, D. (2000). *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. Cambridge, MA: MIT Press.

35. Upton, E., & Halfacree, G. (2014). *Raspberry Pi user guide.* West Sussex, UK: Wiley.

36. Rieffel, J., & Sayles, D. (2010). EvoFab: A fully embodied evolutionary fabricator. In G. Tempesti, A. M. Tyrrell, & J. F. Miller (Eds.), *Proceedings, Evolvable Systems: From Biology to hardware: 9th International Conference, ICES 2010* (pp. 372–380). Berlin, Heidelberg: Springer.

37. Rossi, C., & Eiben, A. (2014). Simultaneous versus incremental learning of multiple skills by modular robots. *Evolutionary Intelligence*, *7*(2), 119–131.

38. Shen, H., Yosinski, J., Kormushev, P., Caldwell, D. G., & Lipson, H. (2012). Learning fast quadruped robot gaits with the RL power spline parameterization. *Cybernetics and Information Technologies*, *12*(3), 66–75.

39. Trianni, V. (2008). *Evolutionary swarm robotics: Evolving self-organising behaviours in groups of autonomous robots*. Berlin, Heidelberg: Springer.

40. Vargas, P. A., Di Paolo, E. A., Harvey, I., & Husbands, P. (2014). *The horizons of evolutionary robotics*. Cambridge, MA: MIT Press.

41. Waibel, M., Floreano, D., & Keller, L. (2011). A quantitative test of Hamilton's rule for the evolution of altruism. *PLOS Biology*, *9*(5), e1000615.

42. Wang, L., Tan, K. C., & Chew, C. M. (2006). *Evolutionary robotics: From algorithms to implementations*. Singapore: World Scientific.

43. Watson, R. A., Ficici, S. G., & Pollack, J. B. (2002). Embodied evolution: Distributing an evolutionary algorithm in a population of robots. *Robotics and Autonomous Systems*, *39*(1), 1–18.

44. Watson, R. A., Ficiei, S., & Pollack, J. B. (1999). Embodied evolution: Embodying an evolutionary algorithm in a population of robots. In *Proceedings of the 1999 Congress on Evolutionary Computation, CEC 99, vol. 1*. New York: IEEE.

45. Wehner, M., Truby, R. L., Fitzgerald, D. J., Mosadegh, B., Whitesides, G. M., Lewis, J. A., & Wood, R. J. (2016). An integrated design and fabrication strategy for entirely soft, autonomous robots. *Nature*, *536*(7617), 451–455.

46. Zagal, J. C., & Ruiz-Del-Solar, J. (2007). Combining simulation and reality in evolutionary robotics. *Journal of Intelligent and Robotic Systems*, *50*(1), 19–39.

47. Zykov, V., Mytilinaios, E., Adams, B., & Lipson, H. (2005). Self-reproducing machines. *Nature*, *435*(7039), 163–164.

48. Zykov, V., Mytilinaios, E., Desnoyer, M., & Lipson, H. (2007). Evolved and designed self-reproducing modular robotics. *IEEE Transactions on Robotics*, *23*(2), 308–319.

Appendix

Appendix 1: Genome Recombination and Mutation

The purpose of this appendix is to illustrate the recombination and mutation operators we selected for the mating process. For simplicity, we consider parent genomes less complex than the spider and the gecko used in the experiments.

When the recombination operator is used on the parent genomes shown in Figure 14, subtrees are randomly exchanged, resulting in an offspring displayed in Figure 15.

Finally, the mutation operator may make a random mutation on the offspring—preferably with substantially low probability. Note that as long as the node is of valid format (i.e., some valid robot

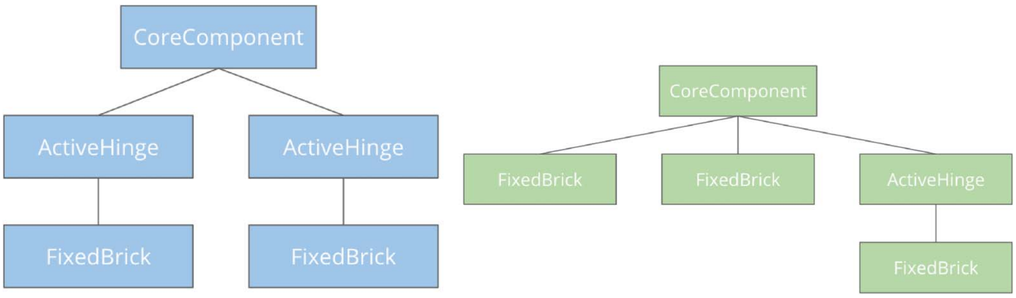


Figure 14. Tree representation of parent genomes.

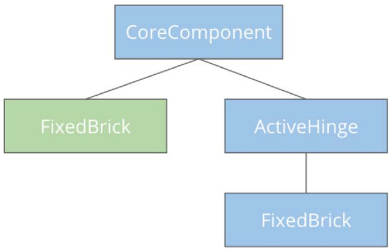


Figure 15. Offspring after recombination operator is applied.

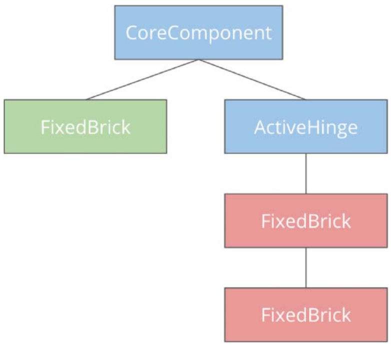


Figure 16. Offspring after mutation operator is applied.

part in the definition of RoboGen), the mutation is valid as well, meaning that it does not need to follow any paradigm (such as color) from the parent genomes. To illustrate this notion, Figure 16 shows a mutated child where the leaf of the right subtree was exchanged for two red nodes.

Appendix 2: Internal Description of Robot Morphologies

The building blocks of robots' bodies and their configuration are described in a configuration file, as shown in Listings 1, 2, and 3. The syntax of the file is the same as used in the RoboGen system.

Each line of the file describes a single module using a fixed set of parameters. The configuration starts with a single core component, hosting the controller. The level of indentation describes a parent-child relationship between the modules, forming a tree structure.

The available component types in our robots are as follows:

- Core component (up to 4 children components),
- Fixed brick (up to 3 children components),
- Active hinge (1 child component).

The rendered images of the elements are displayed in Figure 4. The current RoboGen specification uses four additional types of modules, as well as a number of discontinued, legacy part types. In our research, however, we have decided to limit the complexity by choosing only the aforementioned subset.

Each line can be broken down into five sections:

1. Attachment position on parent part: 0–3
2. Part type
3. Unique identifier

4. Orientation relative to parent: 0–3, representing increments of 90°
5. Parameters

As we do not use any of RoboGen's parametrized parts, the only parameter in the configuration files presented here is the color of each component.

As an example, let us now break down the configuration from Listing 1, encoding the quadruped blue robot (the spider). It all starts with a core component named E0:

```
0 CoreComponent E0 0 BLUE
```

Then, the four active hinges are added, one to each side:

```
0 CoreComponent E0 0 BLUE
       0 ActiveHinge I0 1 BLUE
       1 ActiveHinge I2 1 BLUE
       2 ActiveHinge I4 0 BLUE
       3 ActiveHinge I6 0 BLUE
```

The hinges I0 and I2 are mounted to the front and the back face of the core module, and they are rotated 90°. Hinges I4 and I6, mounted to the left and right faces, have no rotation. The next level contains the fixed blocks, attached to the distal ends of the hinges:

```
0 CoreComponent E0 0 BLUE
       0 ActiveHinge I0 1 BLUE
             0 FixedBrick F0 0 BLUE
       1 ActiveHinge I2 1 BLUE
             0 FixedBrick F2 0 BLUE
       2 ActiveHinge I4 0 BLUE
             0 FixedBrick F4 0 BLUE
       3 ActiveHinge I6 0 BLUE
             0 FixedBrick F6 0 BLUE
```

All the blocks are identical, with no rotation. Then come another set of active hinges (I1, I3, I5, and I7), mounted to those blocks:

```
0 CoreComponent E0 0 BLUE
       0 ActiveHinge I0 1 BLUE
             0 FixedBrick F0 0 BLUE
                0 ActiveHinge I1 0 BLUE
       1 ActiveHinge I2 1 BLUE
             0 FixedBrick F2 0 BLUE
                0 ActiveHinge I3 0 BLUE
```

```
    2 ActiveHinge I4 0 BLUE
        0 FixedBrick F4 0 BLUE
          0 ActiveHinge I5 0 BLUE
    3 ActiveHinge I6 0 BLUE
        0 FixedBrick F6 0 BLUE
          0 ActiveHinge I7 0 BLUE
```

All of them are mounted to the face of the fixed block opposite its attachment point, and they are not rotated. Finally, the last set of fixed blocks with no rotation, attached to the other end of the hinges, is added to complete the robot:

```
0 CoreComponent E0 0 BLUE
    0 ActiveHinge I0 1 BLUE
        0 FixedBrick F0 0 BLUE
          0 ActiveHinge I1 0 BLUE
            0 FixedBrick F1 0 BLUE
    1 ActiveHinge I2 1 BLUE
        0 FixedBrick F2 0 BLUE
          0 ActiveHinge I3 0 BLUE
            0 FixedBrick F3 0 BLUE
    2 ActiveHinge I4 0 BLUE
        0 FixedBrick F4 0 BLUE
          0 ActiveHinge I5 0 BLUE
            0 FixedBrick F5 0 BLUE
    3 ActiveHinge I6 0 BLUE
        0 FixedBrick F6 0 BLUE
          0 ActiveHinge I7 0 BLUE
            0 FixedBrick F7 0 BLUE
```

All of the parts making up the spider are blue. The colors are irrelevant to the operation of the robot, but they are a useful tool to visualize which parts of the offspring come from which parent.

Listing 1. Genotype example specifying the blue spider in Figure 9. See main text for details.

```
0 CoreComponent E0 0 BLUE
  0 ActiveHinge I0 1 BLUE
    0 FixedBrick F0 0 BLUE
      0 ActiveHinge I1 0 BLUE
        0 FixedBrick F1 0 BLUE
  1 ActiveHinge I2 1 BLUE
    0 FixedBrick F2 0 BLUE
      0 ActiveHinge I3 0 BLUE
        0 FixedBrick F3 0 BLUE
  2 ActiveHinge I4 0 BLUE
    0 FixedBrick F4 0 BLUE
      0 ActiveHinge I5 0 BLUE
        0 FixedBrick F5 0 BLUE
  3 ActiveHinge I6 0 BLUE
    0 FixedBrick F6 0 BLUE
      0 ActiveHinge I7 0 BLUE
        0 FixedBrick F7 0 BLUE
0 CoreComponent E0 0 GREEN
  0 ActiveHinge I0 1 GREEN
    0 FixedBrick F0 0 GREEN
      0 ActiveHinge I1 1 GREEN
        0 FixedBrick F1 0 GREEN
          1 ActiveHinge I2 1 GREEN
            0 FixedBrick F2 0 GREEN
          2 ActiveHinge I3 3 GREEN
            0 FixedBrick F3 0 GREEN
```

Listing 1. (*continued*)

```
  1 ActiveHinge I4 1 GREEN
    0 FixedBrick F4 0 GREEN
  3 ActiveHinge I5 3 GREEN
    0 FixedBrick F5 0 GREEN
0 CoreComponent E0 0 WHITE
  0 ActiveHinge I0 1 GREEN
    0 FixedBrick F0 0 GREEN
      0 ActiveHinge I1 1 GREEN
        0 FixedBrick F1 0 GREEN
          1 ActiveHinge I2 1 GREEN
            0 FixedBrick F2 0 GREEN
          2 ActiveHinge I3 3 GREEN
            0 FixedBrick F3 0 GREEN
  1 ActiveHinge I4 3 GREEN
    0 FixedBrick F4 0 GREEN
  3 ActiveHinge I5 0 GREEN
    0 ActiveHinge I6 1 GREEN
      0 FixedBrick F5 1 BLUE
        0 ActiveHinge I7 1 BLUE
          0 FixedBrick F7 0 BLUE
```

Listing 2. Genotype specifying the green gecko.

```
0 CoreComponent E0 0 GREEN
  0 ActiveHinge I0 1 GREEN
    0 FixedBrick F0 0 GREEN
      0 ActiveHinge I1 1 GREEN
        0 FixedBrick F1 0 GREEN
          1 ActiveHinge I2 1 GREEN
            0 FixedBrick F2 0 GREEN
          2 ActiveHinge I3 3 GREEN
            0 FixedBrick F3 0 GREEN
  1 ActiveHinge I4 1 GREEN
    0 FixedBrick F4 0 GREEN
  3 ActiveHinge I5 3 GREEN
    0 FixedBrick F5 0 GREEN
```

Listing 3. Genotype specifying the offspring of the green gecko and the blue spider.

```
0 CoreComponent E0 0 WHITE

  0 ActiveHinge I0 1 GREEN

    0 FixedBrick F0 0 GREEN

      0 ActiveHinge I1 1 GREEN

        0 FixedBrick F1 0 GREEN

          1 ActiveHinge I2 1 GREEN

            0 FixedBrick F2 0 GREEN

          2 ActiveHinge I3 3 GREEN

            0 FixedBrick F3 0 GREEN

  1 ActiveHinge I4 3 GREEN

    0 FixedBrick F4 0 GREEN

  3 ActiveHinge I5 0 GREEN

    0 ActiveHinge I6 1 GREEN

      0 FixedBrick F5 1 BLUE

        0 ActiveHinge I7 1 BLUE

          0 FixedBrick F7 0 BLUE
```

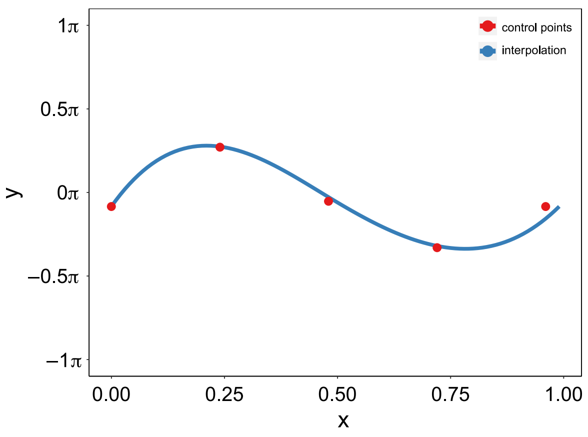


Figure 17. Example of a spline interpolated from six control points (note that five of these control points are freely defined, and the sixth is added to ensure periodicity).

Appendix 3: Controller

The controller represents a set of splines that altogether 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, α_i), where t_i represents time and α_i the corresponding value. $t_i \in [0, 1]$ is defined as

$$t_i = \frac{i}{n-1} \qquad \forall i = 0, \ldots, n-1, \tag{2}$$

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 dedicated GSL[4] C functions. Using GSL, it is possible to query a spline for a different number of points than it was defined with; an example is shown in Figure 17.

Appendix 4: Learner

The algorithm creates the initial policy with as many splines as there are *active hinge* modules, and each spline is initialized to have two control points. These control points are initialized at 0.5 and then perturbed using Gaussian noise. The algorithm then enters an evaluation-adaptation loop to refine the policy, until the stopping condition is reached. A ranking of k best policies encountered so far is kept to inform the adaptation of the current policy.

Adaptation consists of three components: spline size increase, exploitation, and exploration. The spline is gradually refined by incrementing the number of control points periodically as proposed in [38], depending on the start and end sizes of the splines and the number of evaluations. The size increase amounts to incrementing the number n of control points that define the spline by 1. For the k best archived policies this new point is interpolated from their definition. In the exploitation

4 http://www.gnu.org/software/gsl/

step, the current parameters are adapted based on the values of the k best policies. In the exploration phase, policies are adapted by applying Gaussian perturbation to the policy resulting from exploitation. Over the course of the run the variance σ^2 is diminished, which decreases exploration and increases exploitation. The pseudocode for the algorithm, as defined in [26], is displayed in Algorithm 2.

The reward awarded to a controller is calculated as

$$R = \left(100 \frac{\sqrt{\Delta_x^2 + \Delta_y^2}}{\Delta_t}\right)^6, \tag{3}$$

where Δ_x and Δ_y are the displacements along the x and y axes measured in meters, and Δ_t the evaluation time, as in [38].

Each controller is evaluated for 30 s. The evaluation period is determined by striking a balance between battery consumption and evolved gait distinction; if the evaluation period is shorter, more gaits could be evaluated with the current battery capacity, but that would make it harder to distinguish a good gait from a bad one.

The RL PoWER parameter settings were taken from [9] and are summarized in Table 1.

A new spline is generated by taking the current spline, adding a Gaussian perturbation to every control point with mean 0 and variance $\sigma^2(t)$, and then adding a weighted sum of the best k splines in ranking with weights w_i defined as

$$w_i = \frac{f_i}{\epsilon + \sum_{j=1}^{k} f_j},$$

where f_i is the fitness of the ith gait in ranking and ϵ is a parameter to avoid division by 0 and is set to 10^{-10}.

Algorithm 2: RL PoWER

```
1  initialization;
2  policy ← initialization;
3  evaluate(policy);
4  while evaluation < total_evaluations do
       /* Update the ranking of k best policies */
5      ranking.insert(policy);
6      if ranking.size > k then
7          ranking.remove_worst();
8      end
       /* Spline size increase */
9      if evaluation mod increase_delta = 0 then
10         spline_size ← spline_size + 1;
11         reinterpolate_all(ranking);
12         reinterpolate(policy);
13     end
       /* Exploitation */
14     rewards ← 0;
15     weighted_total ← 0;
16     for p in ranking do
17         rewards ← rewards + p.reward;
18         weighted_parameters ← p.reward * (policy.parameters - p.parameters);
19         weighted_total ← weighted_total + weighted_parameters;
20     end
21     next_policy.parameters ← policy.parameters + weighted_total / (rewards + $\epsilon$);
       /* Exploration */
22     next_policy.parameters ← next_policy.parameters + normrnd(0,sqrt(variance));
23     policy ← next_policy;
24     variance ← variance * variance_decay;
25     evaluate(policy);
26 end
```