17

Evolutionary Robotics

In this chapter we discuss evolutionary robotics (ER), where evolutionary algorithms are employed to design robots. Our emphasis lies on the evolutionary aspects, not on robotics per se. Therefore, we only briefly discuss the ER approaches that work with conventional evolutionary algorithms to optimize some robotic features and pay more attention to systems that can give rise to a new kind of evolutionary algorithms. In particular, we consider groups of mobile robots whose features evolve in real-time, for example, a swarm of Mars explorers or 'robot moles' mining ore deep under the surface. In such settings the group of robots is a population itself, which leads to interesting interactions between the robotic and the evolutionary components of the whole system. For robotics, this new kind of ER offers the ability to evolve controllers as well as morphology in partially unknown and changing environments on the fly. For evolutionary computing, autonomous mobile robots provide a special substrate for implementing and studying artificial evolutionary processes in physical entities going beyond the digital systems of today's evolutionary computing.

17.1 What Is It All About?

Evolutionary robotics is part of a greater problem domain that does not have a precise definition, but it can be characterised as problems involving physical environments. (The first edition of this book treated it very briefly in Sect. 6.10.) Evolutionary systems in this domain have populations whose members are not just points in some abstract, stateless search space, e.g., the set of all permutations of $1, \ldots, n$, but are embedded in real time and real space. In other words, such systems feature 'situated evolution' or 'physically embodied evolution' as discussed in [371]. Examples include robot swarms where the robot controllers are evolving on-the-fly, artificial life systems where preda-

A.E. Eiben, J.E. Smith, *Introduction to Evolutionary Computing*,
Natural Computing Series, DOI 10.1007/978-3-662-44874-8_17

tors and prey coevolve in a simulated world[1], or adaptive control systems to regulate a chemical factory where the adaptive force that changes the control policy over time is an evolutionary algorithm. In all these examples the entities undergoing evolution are *active*, in the sense that they do something, i.e., they exhibit some behaviour that can change the environment. This distinguishes them from applications evolving *passive* entities, e.g., using evolutionary algorithms for combinatorial or numerical optimisation. Hence, one of the main distinguishing features of problems in this area is that the algorithmic goal is formulated in terms of functional, rather than structural properties. In other words, it is some exhibited behaviour we are after.

The fundamental problem in the design of autonomous robots is that the targeted robot behaviour is a "dynamical process resulting from nonlinear interactions between the robot's control system, its body, and the environment" [163]. Therefore, the robot's makeup, which falls under the designer's influence, has only a partial and indirect effect on the desired behaviour. This implies that the link between the controllable parameters and the target variables is weak, ill-defined, and noisy. Furthermore, the solution space can be – and typically is – subject to conflicting constraints and objectives. Think, for instance, of trying to be fast while also sparing the batteries. Evolutionary algorithms offer a very promising approach here, because they do not require a crisply defined fitness function, they can cope with constraints and multiple objectives, and they can find good solutions in complex spaces even under noisy and dynamically changing conditions, as discussed in the previous chapters.

The use of evolutionary computing techniques to solve problems arising in robotics began in the 1990s and now evolutionary robotics has a large body of related work [322, 432, 450]. Reviewing the field is beyond the scope of this chapter, the surveys in [61, 163, 449] give a good impression of the area. The picture that arises from the literature shows a large variety of work with different robot morphologies, different controller architectures, and different tasks in different environments, including underwater and flying robots. Some of the evolved solutions are really outstanding, but many roboticists are still skeptical and the evolutionary approach to robot design is outside of the mainstream. As phrased by Bongard in [61]: "The evolution of robot bodies and brains differs markedly from all other approaches to robotics".

17.2 Introductory Example

As an introductory example let us consider the problem of evolving a controller for a wheeled robot that must ride towards the light source in a flat and empty arena. Regarding the hardware we can assume a simple cylindric robot

[1] Note that we consider physical environments in a broad sense, including simulated environments, as long as they include some notion of space and time.

with a couple of wheels and a few LED lights, a camera, a gyroscope, and four infrared sensors, one on each side. As for the control software, the robot can use a neural network that receives inputs from the **sensors** (camera, infrared sensors, gyroscope) and gives instructions to the **actuators** (motors and LEDs). Thus, a **robot controller** is a mapping between sensory inputs and actuatory outputs. Its quality – which determines its fitness used in an evolutionary algorithm – depends on the behaviour it induces. For instance, the time needed from start to hitting the source can be used for this purpose.

As with any application of an EA, an essential design decision when evolving robot controllers is to distinguish phenotypes and genotypes, cf. Chap. 3. Simply put, this distinction means that we perceive the controllers with all their structural and procedural complexity as phenotypes and we introduce a structurally more simple representation of the controllers as genotypes. Obviously, we also need a mapping from genotypes to phenotypes, which might be a simple mapping, or a complex transformation. For example, a robot controller may consist of a collection of artificial neural nets (ANNs) and a decision tree, where the decision tree specifies which ANN will be invoked to produce the robot's response in a given situation. This decision tree can be as simple as calling ANN-1 when the room is lit and calling ANN-2 when the room is dark. This complex controller, i.e., the decision tree and the two ANNs as phenotype, could be represented by a simple genotype of two vectors, $w_1 \in \mathbb{R}^m$ and $w_2 \in \mathbb{R}^n$, showing the weights of the hidden layer in ANN-1 and ANN-2, respectively.

After making these application-specific design decisions, developing a suitable EA is rather straightforward because one can simply use the standard machinery and employ any EA that works in real-valued vector spaces. The only robotic-specific component is the fitness function that requires one or more test runs of the robot using the controller under evaluation. Then the behaviour of the robot is observed and some task-specific quality measure is taken to determine a fitness value. Running such an EA is not different from running any other EA, except for the fitness measurements, see Fig. 17.1. Sometimes, these can be performed by a simulator and the whole evolutionary process can be completed inside a computer. Alternatively, some or all fitness evaluations are carried out by real robots. In this case the genotype to be evaluated is sent to a robot to test it. Such a test delivers fitness values that are sent back to the computer running the EA. Employing real-world fitness evaluations implies a specific architecture, with a computational component that runs the EA and a robot component that executes the real-world fitness measurements. However, it could be argued that for the EA this dual architecture is invisible, it only needs to pause for longish periods waiting for the fitness values to be returned. A general approach to cope with time-consuming fitness evaluations is the use of surrogate models, as discussed in Chap. 14.

In terms of the problem categories discussed in Chap. 1, the task of designing a good robot controller represents a model-building problem. As explained above, a robot controller is a mapping between sensory inputs and actuator

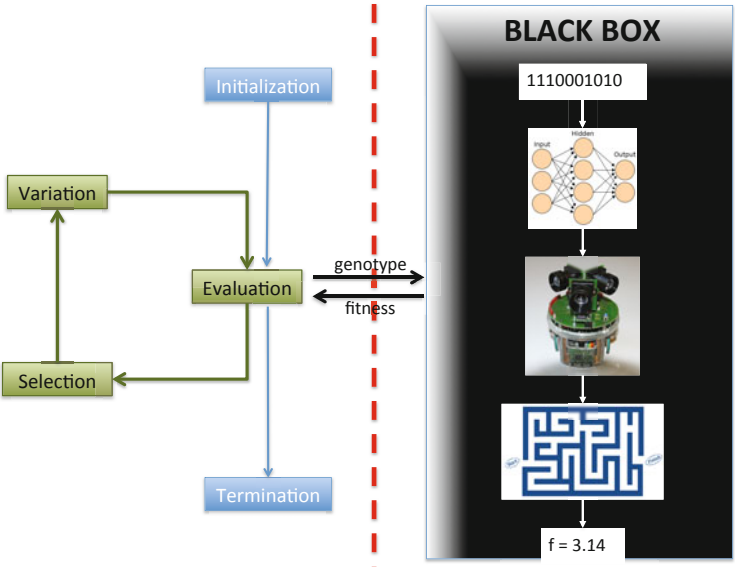

Fig. 17.1. Classic scheme of evolving robot designs. Note that any EA can be employed here, the only application-specific detail is the way fitness values are calculated

outputs and the essence of constructing a good controller is to find a mapping that provides an appropriate output for all inputs. In general, a model-building problem can be converted into an optimisation problem by creating a training set of input–output records and measuring model quality by the number or percentage of correct outputs it produces on the training set. For our example we could set up a training set consisting of pairs of sensory input patterns and corresponding actuator commands. However, this requires a microscopic-level description of desired robot behaviour and in practice this is not feasible. Instead, we could follow a macroscopic-level approach and use training cases where the input component describes the starting conditions (e.g., position of the robot in the arena and the brightness in the room) and the output component specifies that the robot is close enough to the light source.

17.3 Offline and Online Evolution of Robots

The usual approach in evolutionary robotics is to employ **offline evolution**, cf. Fig. 17.2. This means that an evolutionary algorithm to find a good controller is applied before the operational period of the robot. When the user is satisfied with the evolved controller, then it is deployed (installed on the physical robot) and the operational stage can start. In general, the evolved controllers do not change after deployment during the operational stage, or at

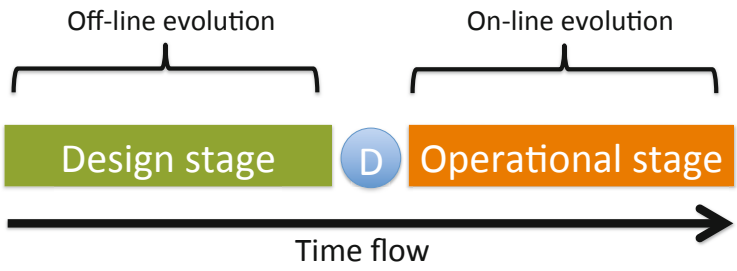

Fig. 17.2. Workflow of robot design distinguishing the design stage and the operational stage of the robots, separated by the moment of deployment (circle in the middle). Offline evolution takes place in the design stage and the evolved features (controllers or morphological details) do not change after deployment. Online evolution is performed during the operational period, which means that the robot's features are continually changed by the evolutionary operators

least not by evolutionary operators. The alternative is to apply **online evolution** to evolve controllers during the operational period. This implies that evolutionary operators can change the robot's control software even after its deployment.

The difference between the offline and online approaches was identified early in the history of the field, for instance by Walker et al. in [449] who used the names "training phase evolution" and "lifelong adaptation by evolution", respectively. However, the online variant has received very little attention and the huge majority of existing work is based on the offline approach.[2] This preference is understandable, because it is fully in line with the widespread use of EAs as optimizers. However, natural evolution is not a function optimizer. The natural role of evolution is that of permanent adaptation and this role is expected to become more and more important in the future of robotics, as phrased by Nelson et al. in [315].

> "Advanced autonomous robots may someday be required to negotiate environments and situations that their designers had not anticipated. The future designers of these robots may not have adequate expertise to provide appropriate control algorithms in the case that an unforeseen situation is encountered in a remote environment in which a robot

[2] This is true of machine learning in general – the huge majority of algorithms and papers about model building are concerned with learning from a fixed training set rather than continuous learning.

> cannot be accessed. It is not always practical or even possible to define every aspect of an autonomous robots environment, or to give a tractable dynamical systems-level description of the task the robot is to perform. The robot must have the ability to learn control without human supervision."

The vision articulated here identifies a problem, and evolutionary computing can be (part of) the solution. Of course, online adaptation need not be based on or restricted to evolutionary techniques. One may even argue that evolution is too slow for such applications. However, we have seen in Chap. 11 that EAs can be used for dynamic optimization and there are good examples of using EAs for online learning, e.g., [75].

17.4 Evolutionary Robotics: The Problems Are Different

After the above considerations, we can discuss the properties that make robotics a very interesting context for evolutionary computing. To begin with, let us consider the problems, in particular the fitness functions for robotics. These exhibit a number of particular features that may not be unique one by one, but together they form a very challenging combination.

- The fitness function is very noisy.
- The fitness function is very costly.
- The fitness function is very complex.
- There may not be an explicit fitness function at all.
- The fitness landscape has 'no-go areas'.

The first four challenges occur in all segments of ER, including offline evolution based on simulation, the fifth one is characteristic for applications with online evolution is real hardware. In what follows we discuss these one by one.

Noisy Fitness: Noise is inherent in the physical world. For instance, two LEDs on a robot are not fully identical and the light they emit can be different even though they receive the same controller instruction. Furthermore, seemingly small environmental details can cause differences, e.g., the left wheel may be a bit warmer than the right wheel and make the robot deviate from the intended trajectory. This means that the link between actually controllable responses and robot behaviour is nondeterministic and the range of variations can be quite large.

Costly Fitness: Calculating the tour length for a given route of a traveling salesman can be done in the blink of an eye. However, measuring the time a robot needs for navigating to a target location can take several minutes. Moreover, it is an essential requirement for robot controllers that they work under many different circumstances. To this end, the controller must be tested under different initial conditions, for instance starting at different locations of

the arena and/or under various lights. Ultimately this means that many time-consuming measurements are needed to assess the fitness of a given controller.

Complex Fitness: The phenotypes encoded by genotypes in ER are controllers that define the response of the actuators for any given combination of the inputs provided by the sensors.[3] However, a controller can only determine the actuator response on a low level (e.g., the torque on the left wheel or the colour of the LED light), whereas the fitness depends on exhibited robot behaviour on a high level (e.g., the robot is driving in circles). Hence, the phenotypes, i.e., the controllers, cannot be directly evaluated and one has to observe and assess the robot behavior induced by the given controller. Thus, in EC we have a 3-step chain, genotype–phenotype–fitness, while in ER the chain is 4-fold, genotype–phenotype–behavior–fitness. Furthermore, in the case of robot swarms, we have one more level, that of the group behaviour that raises further questions about considering the individual or the group when executing selection [448]. Thus, in general, the link between actually controllable values in the genotypes and robot behaviour is complex and ill-defined without analytical models.

Implicit Fitness: In the true spirit of biological evolution, EC can be used in an objective-free fashion to invent robots that are well suited to some (previously unknown and/or changing) environment. In such cases robots do not have a quantifiable level of fitness; they are fit if they survive long enough to be able to mate and propagate their genetic makeup. Such problems are very different from classic optimization and design problems in that they require a 'curiosity instinct' for pursuing novelty and open-ended exploration, rather than a drive for chasing optima.

'No-Go-Areas' in Fitness Landscape: Evaluating a poor candidate solution of a traveling salesman problem can waste time. Evaluating a poor candidate solution (bad controller) can destroy the robot. When working in real hardware, such candidate solutions must be avoided during the evolutionary search.

For further elaboration let us focus on a particular subarea in robotics: applications that involve (possibly large) groups of autonomous robots that undergo online evolution. One noteworthy property of such applications is that artificial evolution is required to play two roles at the same time: optimising towards some quantifiable skills of the robots as specified by the user as well as enabling open-ended adaptation to the given environment. To illustrate this recall the examples about Mars explorers and deep mining 'robot moles'. These are problems where the environment is not fully known to the designers in advance. The robot designs at the time of deployment are therefore just initial guesses and the designers' work needs to be finalised on the spot. As a necessary prerequisite for surviving at all, the robots must be fit for the environment (viability) and to fulfill the users' expectations they should

[3] Genotypes can of course also encode for morphological features. We disregard this option here for the sake of simplifying the argument.

perform their tasks well (usefulness). Artificial evolutionary systems today are typically developed with either but not both roles in mind. Optimising task performance is the leading motive in evolutionary computing and mainstream evolutionary robotics, whereas 'goal-less' adaptation to some environment is common in artificial life, but recent work has demonstrated that the two priorities can be integrated elegantly in one system [204].

Another special property here is that robots can be passive as well as active components of the evolutionary system.[4] On the one hand, robots are *passive* from the evolutionary algorithm perspective if they just passively undergo selection and reproduction. This is the dominant perspective when using traditional EAs to optimise robot designs in an offline fashion. In these cases the robots' only role in the EA is to establish fitness values that the selection mechanism needs in order to decide about who is to reproduce and who is to survive. On the other hand, robots can be *active* from the evolutionary algorithm perspective because they have processors on board that can perform computations and execute evolutionary operators. For instance, a robot could select another robot to mate with, perform recombination on the two parent controllers, evaluate the child controllers, and select the best one for further usage. Obviously, different robots can apply different reproduction operators and/or different selection preferences. This constitutes a new type of evolutionary system that is not only distributed, but also heterogeneous regarding the evolutionary operators.

A further property worth mentioning is that robots can also influence the evolutionary dynamics implicitly by structuring the population. This influence is grounded in the physical embedding which makes a group of robots spatially structured. Spatially structured EAs are, of course, nothing new, see for instance Sect. 5.5.7. However, in evolving robot swarms this structure is not designed, but emergent, without being explicitly specified by the EA designer, and it can have a large impact on the evolutionary process. For instance, the maximum sensor and communication ranges imply (dynamically changing) neighbourhoods that affect evolutionarily relevant interactions, e.g., mate selection and recombination. Obviously, sensor ranges and the like are robot attributes that can be controlled by the designer or experimenter, but their evolutionary effects are complex and to date there is no know-how available on adjusting evolutionary algorithms through such parameters.

To conclude this section let us briefly consider simulated and real fitness evaluations. A straightforward way of circumventing hardware-related difficulties and coping with costly fitness functions is the use of simulators. The simulator must represent the robot and the test environment including obstacles and other robots, if applicable. Using a simulator with a high level of abstraction (low level of details) can make a simulation orders of magnitude faster than a real-life experiment. Meanwhile, detailed simulations with a hi-fidelity physics engine and robots with many sensors can be much slower than

[4] This is not the same passive–active division we discussed in Sect. 17.1.

real life. Parallel execution on several processors can mitigate this problem and, in the end, simulations can save a lot of time when assessing the fitness of robot controllers. However, this comes at a price, the infamous **reality gap** that stands for the inevitable differences between simulated and real-life behaviour [233]. For evolutionary robotics, the reality gap means that the fitness function does not capture the real target correctly and even the high fitness solutions may perform poorly in the real world.

17.5 Evolutionary Robotics: The Algorithms Are Different

As mentioned above, standard EAs can be used to help design robots in an offline fashion. However, new types of EAs are needed when controllers, morphologies or both are evolved on the fly. Despite a handful of promising publications, there is not much know-how on such EAs, indicating the need for further research. In the following we try to identify some of the related issues, arranged by the following aspects:

- the overall system architecture (evolutionary vs. robotic components);
- the evolutionary operators (variation and selection);
- the fitness evaluations;
- the use of simulators.

By system architecture we mean the structural and functional relations between the evolutionary and the robotics system components as outlined in [123]. The classic offline architecture relies on a computer that hosts a population of genotypes that represent robot controllers, cf. Fig. 17.1. This computer executes the evolutionary operators (variation and selection), managed in the usual EC fashion. Fitness evaluations can be performed on this computer using a simulator. In this case the whole evolutionary process can be completed inside this computer. Alternatively, (some) fitness evaluations can be carried out by real robots. To this end, the genotype to be evaluated is sent to a robot to test it. After an evaluation period, the robot sends the fitness values back to the computer. Using more robots, evaluation of genotypes can happen in parallel which can lead to significant speed up. After finishing the evolutionary process the best genotype is deployed in the robot(s).

Online architectures are aimed at evolving robot controllers during their operational period. Here we can distinguish between centralised and distributed systems. One type of centralised system works as the online variant of the classic architecture. A master computer oversees the robots and runs the evolutionary process: it collects fitness information, executes variation and selection operators, and sends new controllers to the robots for them to use and test. As opposed to the offline scheme, the operational usage and fitness evaluation of controllers are not separated here. This is illustrated in Fig. 17.3,

left. In the other type of centralised online system the computer running the evolutionary algorithm is internal to the robot, see Fig. 17.3, right.

Distributed online architectures also come in two flavours. In a pure distributed system each robot carries one genome that encodes its own phenotypic features. Selection and reproduction require interaction between more robots, see Fig. 17.4, left. In a hybrid system the encapsulated and the pure distributed approaches are combined, as shown in Fig. 17.4, right.

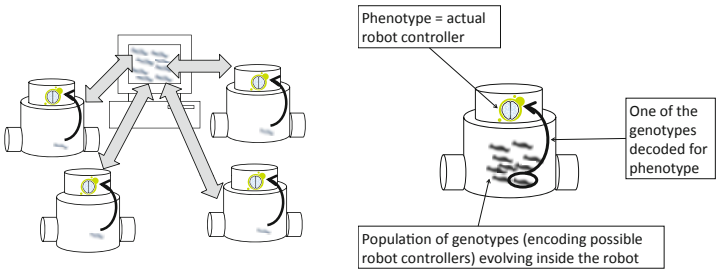

Fig. 17.3. Centralised online evolution architectures. Left: an external master computer oversees the robots and runs the evolutionary process. Right: an internal computer (the robot's own processor) runs an encapsulated evolutionary process, where selection and reproduction do not require information from other robots

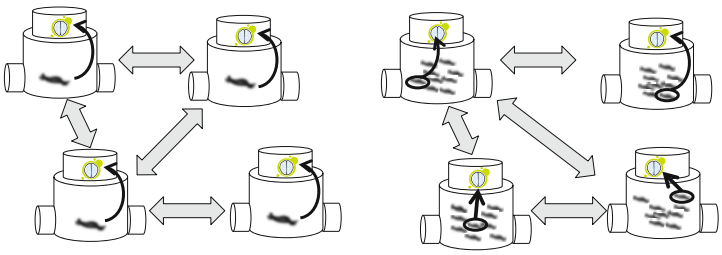

Fig. 17.4. Distributed online evolution architectures. Left: distributed system with one-robot–one-genome. Selection and reproduction require interaction between more robots. Right: a hybrid system is encapsulated and distributed. Encapsulated populations form islands that evolve independently but can crossfertilize.

Regarding the evolutionary operators, we can mention two prominent requirements. First and foremost, reproduction must be constrained to physically viable candidate solutions. This requirement is related to the issue of the 'no-go areas' on the fitness landscape: a poor candidate solution (a bad controller) can break, immobilize, or otherwise disable the robot. Hence, the

evolutionary path should avoid these regions. The area of evolutionary constraint handling, cf. Chap. 13, could be relevant here, especially the use of intelligent reproduction operators and repair mechanisms. However, these are always based on application-specific heuristics and as of today there are no general recipes in EC that would offer solutions here. One promising avenue to avoid damaging robots is the use of a surrogate model to estimate the viability of new candidate solutions before launching them in the real world. We discussed the use of surrogates for interactive EAs in Sect. 14.2. Such models can be maintained and continually updated in every robot independently, but the robot population could also collaborate and develop the models collectively. The second requirement concerns the speed of evolution. Because of the online character of the robotic application, rapid progress is desirable. In other words, it should not take many iterations to reach good performance levels. To this end, there are generic options supported by common evolutionary computing wisdom. For example, using high selection pressure or multiparent crossovers can be considered as accelerators, but with a caveat: greedy search can easily get trapped in local optima.

There are also special considerations for selection in the distributed cases. The physical embedding implies a natural notion of distance, that in turn induces neighbourhoods for robots. The composition, location, and size of these neighbourhoods will depend on the specific circumstances, e.g., ranges for sensing and communication, the environment, noise, etc., but in general it will hold that robots do not have direct access to all other robots. Thus, the selection operators will act on local and partial information, using fitness data of only a fraction of the whole population. In principle, this can be circumvented by using some form of epidemic or gossiping protocols that help in estimating global information. The first results with autonomous selection in peer-to-peer evolutionary systems are very promising [463].

Concerning fitness evaluations, perhaps the most important robotics-specific feature is their duration. In typical EAs for optimization or for offline robot design the real-time duration of fitness evaluations does not impact the evolutionary dynamics. Figuratively speaking, the EA can freeze until the fitness values are returned and proceed with the search afterwards. Thus, no matter how long a fitness evaluation really takes, from the EA perspective it can be seen as instantaneous. Using online evolution in robots this is different, because the real world will not freeze awaiting the fitness evaluations to finish. In such systems, the time needed to complete fitness evaluations is a parameter with a paramount impact. On the one hand, long evaluations imply fewer evaluations in any given time interval. This is certainly disadvantageous for the (real-time) speed of the evolutionary progress. On the other hand, short evaluations increase the risk of poor fitness estimations. This misleads selection operators and drives the population to suboptimal regions. Consequently, the parameter(s) that determine the duration of fitness evaluations should have robust parameter values that work over a wide range of circumstances, or a good parameter control method should be applied that adjusts

the values appropriately on the fly. Research in this direction is just starting, but promising results have been published recently based on a (self-)adaptive mechanism [117].

Finally, let us return to the issue of using simulations to assess the quality of robot designs. A very promising idea here is to use simulations *within* robots for continuous self-modelling [60]. In principle, this means that fitness evaluations can be performed without real-life trials during an online evolutionary process. This approach bears great advantages because it can save time, it can save robots (that otherwise may run into irreparable problems), and it can help experimenters learn about the problem (if only these models are readable). A very recent study showing the benefits of this approach in a robot swarm undergoing online evolution is presented in [323].

17.6 A Glimpse into the Future

Let us conclude this chapter with a somewhat speculative section about the future of evolutionary robotics and evolutionary computing itself. We believe that the combination of robotics and evolutionary computing has a great potential with mutual benefits. On the one hand, EC can provide solutions to hard problems in robotics. In particular, evolution can help design and optimise robot controllers, robot morphologies, or both and it can provide adaptive capabilities for on-the-fly adjustments without human intervention. On the other hand, robotics forms a demanding testing ground for EC and the specific (combination of) challenges in robotics can drive the development of novel types of artificial evolutionary systems.

A detailed discussion of the robotics perspective is beyond the scope of this book. For the interested reader, we recommend [119], which distinguishes four major areas of development at different levels of maturity:

- automatic parameter tuning in the design space (mature technique),
- evolutionary-aided design (current trend),
- online evolutionary adaptation (current trend),
- automatic synthesis (long-term research).

Regarding EC, our main premise is that we are at the verge of a major breakthrough that will open up a completely new substrate for artificial evolutionary systems. This breakthrough is the technology of self-reproducing physical artefacts. This may be achieved through, for instance, 3D printing or autonomous self-assembly in the very near future. This achievement would extend the meaning of evolvable hardware [466]. To be specific, this means a technology that enables the autonomous construction of a new robot based on a genetic plan produced by mutation or crossover applied to the genetic plan(s) of its parent(s). Given this technology it will be possible to make systems where bodies and brains (i.e., morphologies and controllers) of robots coevolve without the human in the loop. The *Triangle of Life* framework of

Eiben et al. [132] provides a general description of such systems and the paper demonstrates a rudimentary implementation of some of the main system components with cubic robots as building blocks. The Triangle of Life can be considered as the counterpart of the general scheme of an evolutionary algorithm in Chap. 3.

From a historical perspective, the technology of self-reproducing physical artefacts will be a radical game changer in that it will allow us to create, utilize, and study artificial evolutionary processes outside of computers. The entire history of EC can be considered as the process of learning how to construct and operate artificial evolutionary systems in imaginary, digital spaces. This very book is a collection of knowledge accumulated over the last few decades about this. The technology of self-reproducing artifacts will bring the game into real physical spaces. The emerging area will be concerned with Embodied Artificial Evolution [136] or the Evolution of Things [122]. The historical perspective is illustrated by Fig. 17.5 that shows two major transitions of Darwinian principles from one substrate to another. The first transition in the 20th century followed the emergence of computer technology that provided the possibility of creating digital worlds that are very flexible and controllable. This brought about the opportunity to become active masters of evolutionary processes that are fully designed and executed by human experimenters. The second transition is about to take place through an emerging technology based on material science, rapid prototyping (3D printing) and evolvable hardware. This will provide flexible and controllable physical, *in materio*, substrates and be the basis of Embodied Artificial Evolution or the Evolution of Things [147].

Finally, let us mention a new kind of synergy between evolutionary computing, artificial life, robotics, and biology. As of today, the level of technology will not allow for emulating all chemical and biological micro-mechanisms underlying evolution in some artificial substrate. However, even a system that only mimics the macro-mechanisms (e.g., selection, reproduction, and heredity) in a physical medium is a better tool for studying evolution than pure software simulations because it will not violate the laws of physics and will be able to exploit the richness of matter. A recently published list of grand challenges for evolutionary robotics includes "Open-Ended Robot Evolution" where physical robots undergo open-ended evolution in an open environment [121]. With such systems one could investigate fundamental issues, e.g., the minimal conditions for evolution to take place, the factors influencing evolvability, or the rate of progress under various circumstances. Given enough time, we may even witness some of the events natural evolution encountered, such as the emergence of species, perhaps even the Cambrian explosion. This line of development can bridge evolutionary computing, artificial life, robotics, and biology, producing a new category: Life, but not as we know it.

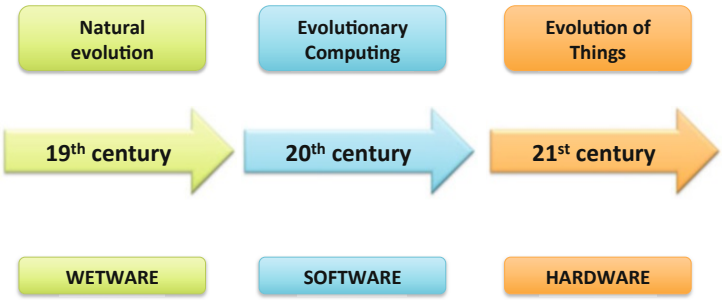

Fig. 17.5. Major transitions of evolutionary principles from one substrate to another. Natural evolution *in vivo* was discovered and described in the 19th century. Computer technology in the 20th century provided digital, *in silico* substrates for creating, utilizing, and studying artificial evolutionary processes. This formed the basis of Evolutionary Computing. An emerging technology based on material science, rapid prototyping (3D printing) and evolvable hardware will provide physical, *in materio* substrates. This will be the basis of Embodied Artificial Evolution or the Evolution of Things.

For exercises and recommended reading for this chapter, please visit www.evolutionarycomputation.org.