EvoRoF: A Framework for On-line and On-board Evolutionary Robotics

Florian Schlachter, Patrick Alschbach and Katja Deuschl
Institute for Parallel and Distributed Systems
University of Stuttgart
Stuttgart, Germany
{Florian.Schlachter, Patrick.Alschbach, Katja.Deuschl}@ipvs.uni-stuttgart.de

***Abstract*—In this paper, we present an evolutionary robotics framework (EvoRoF) for on-line and off-line evolution, as well as on-board and off-board evolution for swarm and reconfigurable robotics. It enables both, the use of artificial neural networks and spiking neural networks and combines both with structural evolution of recurrent networks. It is evaluated with benchmark tests and several use cases are outlined.**

Keywords-on-line evolution; recurrent neural networks; reconfigurable robotics; evolutionary robotics.

I. INTRODUCTION

In swarm robotics, a group of autonomous robots with limited sensors and actuators performs in a cooperative way. These robots often have only limited power resources and local information. Therefore, these robots are forced to take care of power recharging and efficient task allocation to ensure the correct processing of the desired task.

In reconfigurable robotics, a group of robots is able to reconfigure or aggregate into various configurations to generate new functionalities and thus gain a high degree of versatility [1], [2]. New functionalities can arise and the robotic system adapts to different operational demands to solve advanced tasks.

While the swarm and reconfigurable robotics describe a class of robotic systems, Evolutionary Robotics is a way to obtain a desired controller by applying Darwinian mechanisms [3] to the controller of those robots. The artificial evolution of robot controllers enables a swarm or reconfigurable robots to evolve over time in order to adapt to a specific task or to survive in a dynamic environment.

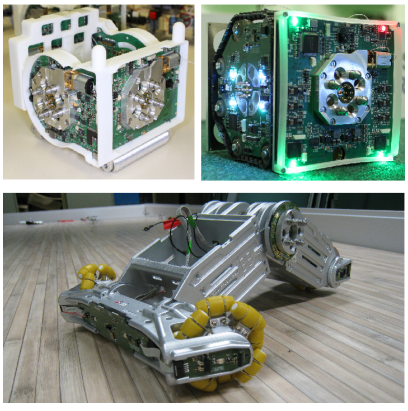

Fig. 1. The three robots developed in the Symbrion and Replicator projects. Top left: Backbone robot. Top right: Scout robot. Bottom: Active Wheel.

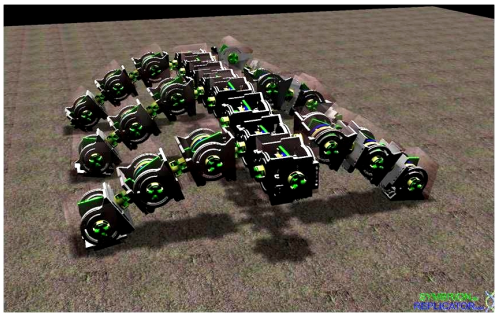

Fig. 2. An aggregated hexapod organisms of Backbone robots in the simulation to demonstrate the capabilities of the reconfigurable mobile robot platform.

Combining all three topics into one platform like in the Symbrion [4] and Replicator [5] projects, delivers a powerful robotic system for dynamic environments and unforeseen situations. Autonomous individual robots can aggregate on demand to artificial organisms with new functionaity and thus extend its operational scope. The automatic design by artificial evolution can be followed by lifelong on-line adaptation.

Thereby, the evolvability of a platform directly affects the level of adaptation and learning in robotic control. Without evolvability, a technical system is not able to change the underlying structure of control for adaptation and learning reasons. In a former paper [6], we showed the different levels of evolvability in the Symbrion and Replicator projects.

We outlined how the mechanical design has to be and the requirements of the embedded software, which we presented in [7], called Symbricator Robot API. Beside the supporting electronics, mechanics and basic software design, the platform itself extends the system by the capability of aggregation. By self-assembling, an artificial robot organism can generate new functionality in order to adapt to a changing environment or task. Figure 1 shows the heterogeneous robots in the Symbrion and Replicator projects. A detailed description can be found in [8], [9]. An aggregated multi-robot organism in simulation can be seen in Figure 2.

 ISBN: 978-1-61208-274-5

In the following we will demonstrate a framework for evolvable robot control, which enables the evolutionary design of robot controllers in individual robots as well as for cooperating robots and artificial organisms. The paper is organized as follow: Section II gives an overview about existing frameworks and related work. Section III outlines the requirements for such a framework and Section IV lines out the actual implementation of the framework, while section VI demonstrates use-cases and experiments. In Section VII, we summarize and conclude the paper.

II. Related Work

One of the earliest approaches is the Generalized Acquisition of Recurrent Links (GNARL) from Angeline et al. [10]. In this framework, the first time an algorithm is enabled to evolve the parameters and the structure of a neural network at the same time without any constraints to the topology of the network. New links are introduced with zero weight to avoid a radical change in the behaviour of a network. New neurons are introduced without any incident links, later mutations are connecting those neurons then in an appropriate manner.

The NeuroEvolution of Augmenting Topologies (NEAT), developed by Stanley et al. [11] is a generation-based framework which allows the evolution of recurrent networks from a minimal initial network by parametric and structural mutation. Beside this so called complexification, the key features of NEAT are the historical marking of new mutational innovations. With this mechanism crossover can be enabled by aligning the genomes and comparing the innovation history. Furthermore, to protect new structures, a niching mechanism, respectively speciation, is introduced. Again, the historical marking allows to calculate the distance between two different genomes and allows the classification of all members of the population with a configurable parameter into species.

The Evolutionary Acquisition of Neural Topologies (EANT) [12] also enables to evolve recurrent networks by parametric and structural mutation. The algorithm is based on the Common Genetic Encoding (CGE) [13] on which the mutation and a NEAT-like crossover operate. This genome encoding enables direct and indirect encoding, is complete, compact and closed. Additionally, they introduced the differentiation between an exploitation phase and an exploration phase. The exploitation phase only optimizes the existing structure by adapting the weights, without changing the structure of the network itself. The exploration phase allows the introduction of new genes by means of structural mutations. These two phases are alternating, starting with several exploitation steps followed by an exploration step so that networks with optimal structures, can adapt the weights of the links.

In Schlachter et al. [14] and Schwarzer et al. [15], we already demonstrated a neural network controller which incorporated structural evolution as well as the possibility to adapt incrementally to a dynamic environment. The advantages and the experiences are compared to all approaches and brought into the new framework.

III. Requirements

In order to evolve controllers for swarm and modular robots, some key issues, which should be fulfilled have to be addressed:

- **On-line and on-board evolution:** In addition to off-board and off-line evolution, the framework should be able to enable on-line and on-board evolution to met the requirements in the projects and allow a broad application scope.
- **Flexible Controller Types:** In order to enable the best choice for a certain scenario, the framework should support different types of control. Beside artificial neural networks it should allow to use spiking neural networks. Additional, other kinds of controllers should be easily integrable by a modular abstraction level.
- **Parametric and Structural Evolution:** Both the weights of links as well as the structure of a network need to be subject to mutational operators to allow complexification from a minimal initial network to the structure which is required by the task to fulfil and adjust the present weights.
- **Simple and flexible use:** The usability should be as simple as possible. The choices of controller type and parameters should be transparent and well organized.
- **Powerful interfaces:** The interfaces to required tools, simulations and the robot itself should be well suited to allow a complete use of the functionalities.
- **Application Range:** The evolutionary framework has to run on individual robots in a swarm, on reconfigurable robots in an artificial organism in a centralized as well as decentralized manner.

IV. Architecture and components

The framework is designed modular in object-oriented C++. It was carefully designed regarding reusability and extendibility. It supports dynamical changes of controllers during runtime, different network types, mutation operators, selection and fitness functions. An overview of the different modules can be seen in Figure 3. All modules are grouped into five associated groups, which are explained in more detail in the following.

A. Evolutionary Engine

The core of the framework is built by the evolutionary engine. This engine handles the population and the correct evaluation of it. It triggers the generation of the initial population and links the evolvable controllers with the modules for selection and fitness evaluation. Depending on the selection mode, it either processes generation by generation or allows for continuous tournament selection. Each population island consists of a configurable number of individual controllers, derived from the evolvable superclass.

B. EvoRoFConfig and Logger

The EvoRoFConfig module is responsible for the configuration files. In the initial phase, it reads in the files and sets

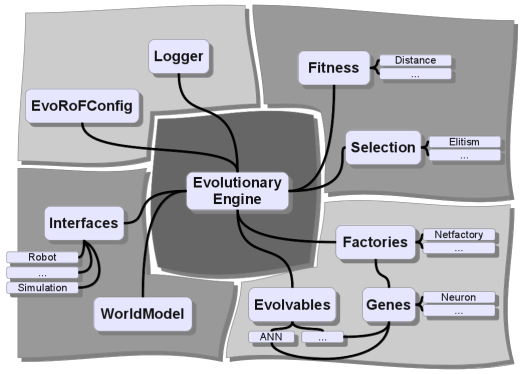

Fig. 3. Overview over the EvoRoF architecture. Central part is the evolutionary engine with the four surrounding blocks.

the configuration. During runtime, this module provides other modules with necessary information about the parameters required.

The logger gives a configurable interface to set the level of detail for logging. The complete framework is distributed in different log levels which can independently be switched on or off. In addition to program information and error states, this module takes care of logging of fitness values and genomes into files for later use or comparison.

C. Fitness and Selection

The modules for fitness and selection can be configured via the configuration file. The evolutionary engine coordinates their activities. The fitness module takes care of the correct evaluation of the fitness of the current running controller and stores the values. The selection mechanism, is responsible for the creation of the next generation, respectively of the next selected individual for evaluation. In generation based mode, this module generates depending on the selection scheme the next generation and delivers it back to the evolutionary engine. In tournament mode, the next individual will be generated and given back to the evolutionary engine.

D. Evolvables, Genes and Factories

Each controller is a subclass of the evolvable superclass. This class delivers the template to be implemented in order to be used by the evolutionary engine in the right way. All controllers have to implement the same interfaces like *initialize()* or *mutate()*.

Every controller encapsulates its own genome, which genes are derived from the genes class. Figure 4 shows an exemplary class hierarchy for the CGE genome.

To separate the creation of new controllers from the logic of a controller, several modules following the factory pattern are available. Those factories generate depending on the desired configuration the individual controllers and push them into the island population of the evolutionary engine.

E. Interface to Simulation and Real Robots

The EvoRoF framework should be able to address the relevant robots and simulation environments of the directly linked projects, thus be extendible to several platforms. For the ongoing experiments, we support interfaces, called wrappers, for the use with different simulators and the three available robot types in the Symbrion and Replicator projects. The used simulators are PlayerStage and the Robot3D simulator of the projects (see also VI Applications). The robot interfaces use the Symbricator Robot API [7]. For generic use, the evolutionary engine can be accessed with a plain wrapper.

Based on the interface functions a common worldmodel is implemented. This worldmodel serves as a container for all relevant sensor data and information from both internal and external sensors and states. In addition, the worldmodel takes care of the message processing of incoming and outgoing messages from and to other robots.

V. Implemented Concepts

A. Controller Types

A straight forward choice of the controller type for this kind of application is the use of neural networks. Following the definition of Maass [16] there are three classes of neural networks. The first generation is based on McCulloch-Pitts neurons (only digital output). These models can give digital output and are universal for every boolean function. The second generation is weighting the inputs and calculating the output via an activation function which delivers a continuous output value. The activation functions can vary from piecewise linear to sigmoid or even more complex functions. The network structure can be either a perceptron or a recurrent network. They can cope with analogue input and are universal for analogue computation. The third class of neural networks are the spiking neural networks. While in the second generation of networks, the output can be interpreted biologically as the current firing rate (number of spikes per period), the timing of spikes is in the foreground for spiking neural networks. The information can be encoded in the timing of spikes.

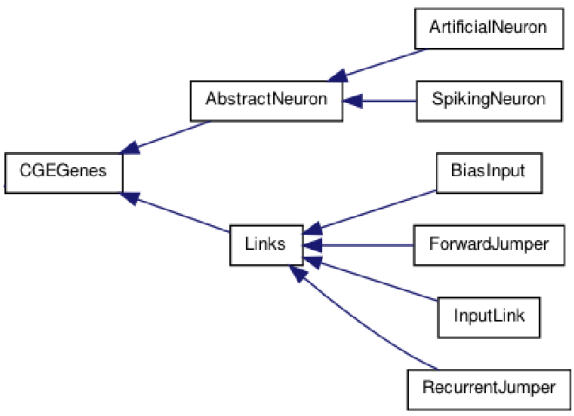

Fig. 4. The class hierarchy of the CGE gene classes.

In this framework, we implemented support for all types of networks in order to allow a higher flexibility in choice. Depending on the scenario and computational demands, the hidden layers can be disabled and even so recurrent connections can be switched off. Thus, the complete range from simple perceptrons to complex recurrent neural networks is feasible.

B. Genotype and Phenotype Representation

For the genotype we adopted the idea of the common genetic encoding (CGE) [13] which is also used in the EANT framework, described by Kassahun et al. [12]. The CGE is a linear genome representation and is in comparison to other approaches like GNARL or NEAT, complete, closed and modular. It further supports direct and indirect encoding and allows for direct evaluation of the genotype without decoding it to a phenotype. The structure of the network is implicitly given and due to its linear nature it is simple to serialize for transmission in order to exchange genomes.

The initial population can be selected as proposed in NEAT without hidden nodes and all inputs connected to all outputs or with an additional initial hidden layer. It is mandatory to start with a minimal configuration in order to find a minimal solution by continuous complexification. Alternatively, the EANT approach, starting with the same minimal network, but increasing the diversity of the start population by some random initial mutations, can be chosen.

C. Evolutionary Operators

The evolutionary operators allow to mutate both the weight parameters of links and the structural complexification by adding new links and nodes. Due to the nature of the CGE, either a forward jumper, a recurrent jumper or a complete subgenome with an arbitrary number of incident input connections can be inserted. The recombination is as described by Stanley et al. [11] in the NEAT framework. The genomes are aligned and combined to generate a new structure containing the common parts as well as the differing parts of both parents. Instead of the global tracking numbers for innovations, the identifiers can be used.

D. Evolutionary process, Fitness and Selection

To better support the on-line and on-board capabilities of the evolutionary framework, we adopted the idea of island evolution from [17]. Each robot represents an island with its own population. The population consists of configurable size of genomes. In the tournament selection mode, one or two genomes, depending if mating is enabled, are taken to generate new offspring. This new individual is then evaluated for a certain time and the fitness value is compared to the existing members of the island. If the fitness is higher as the worst member, this one will be replaced by the new genome. In the generation based mode, all genomes on an island are evaluated. Afterwards, the new offspring is generated. Depending on the configuration this could be for example elitism selection which allows only the children of the best 50 per cent to create the offspring for the next generation.

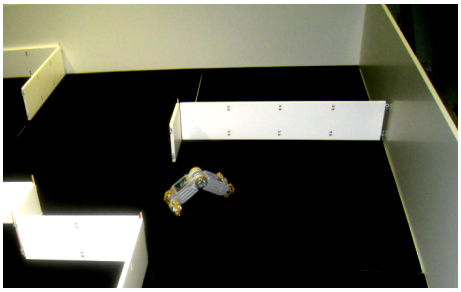

Fig. 5. An Active Wheel robot controlled by EvoRoF performing collision avoidance in a maze-like arena.

VI. Applications

The described framework is used in several application scenarios. It has been shown, that it is powerful enough, to be used in coevolution as well as in distributed on-line evolution for organisms control.

A. Evolution of Collision Avoidance

In this scenario, we used a prototype of the Active Wheel developed in the Symbrion and Replicator projects. We only used the IR sensors on the front and back side. The IR sensors on the back have to be taken into account, because the Active Wheel can collide with the back when turning due to the omnidirectional locomotion.

The population size was 15, the controller type a standard artificial network. There were six IR sensor inputs from the front and additional six sensors at the back extended by bias neuron. In all test runs, the Active Wheel evolved a collision free locomotion and walked randomly through the maze 5.

B. Coevolution of Coordinated Behavior

In a further experiment, we wanted to see the capabilities of the evolution of coordinated behaviour of robots [18]. For this reason, we set up a scenario with two target zones, in which both robots have to be at the same time to gain fitness. Beside a collision free locomotion, the robots have to develop a coordinated locomotion strategy in order to be in this target zones at the same time. Figure 6 shows the two robots, both the blue and red one, and the two yellow target zones in PlayerStage [19].

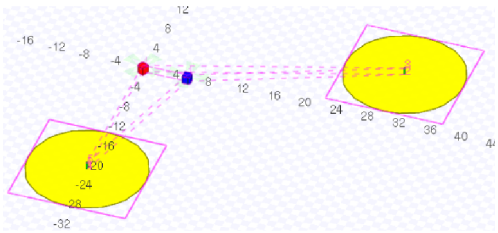

Fig. 6. The scenario: Two robots (blue and red) shall move in a coordinated manner from the left yellow power source to the one on the right upper side. Once a power source is “harvested” the robots have to move to the opposite target. Only when both robots are there, they gain power, respectively can increase their fitness.

 ISBN: 978-1-61208-274-5

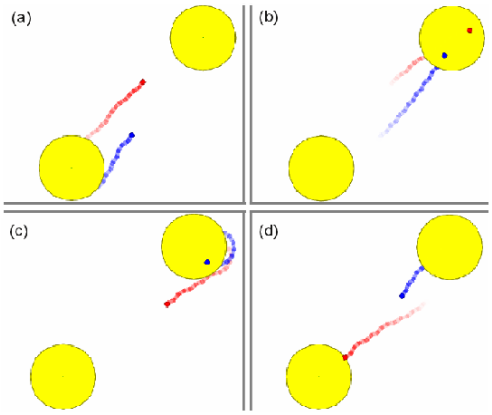

Fig. 7. The sequence shows the final behaviour of a run in alphabetical order in time. The red and blue robots start in the bottom left corner, reach the upper right corner and go back in a coordinated fashion.

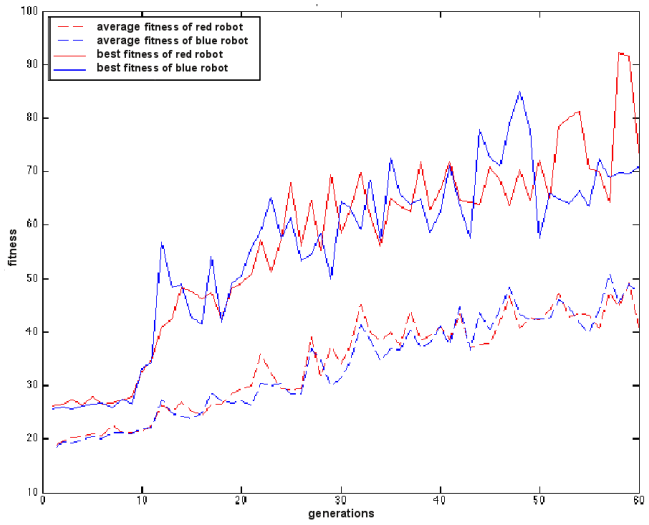

Fig. 8. Fitness development in the scenario. The graphs show the average fitness of both robots (doted lines) and the best individual per generation (solid line).

An evolved behaviour can be seen in Figure 7 where the position tracking of the two robots is depicted. In (a) they approach the upper target, in (b) they reached it and turn around in (c) to approach the lower yellow target in a coordinated way (d).

The used controller type was a recurrent neural network of class two using a cubical robot imitating the Backbone, respectively the Scout robot, of the Symbrion and Replicator projects. The input sensors were two front and two rear IR sensors and eight virtual sensors measuring the distance to the other robot and target zones. The population size was 10 in a generation-based run with different settings regarding the use of hidden neurons and the use of structural mutation. Figure 8 shows an average fitness development in a treatment with structural mutation enabled.

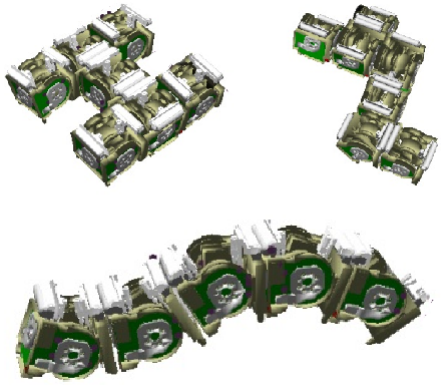

Fig. 9. Different types of multi-robot organisms tested in simulation.

C. Evolution of CPG control

In [20], locomotion for a multi-robot based on a spiking neural network was evolved. By distributed evolution, the organism should be able to emerge a global organism locomotion, by evolution of local control on each individual robot. The basic concept was a central pattern generator scheme, in which the parameters of a sine wave are modified in order to incorporate the sensor input and status messages from other modules. The individual robots have to learn considering the sensor input, which phase shift and amplitude to perform the necessary local behaviour.

In Figure 9, three exemplary organisms in the Robot3D simulator [21] are shown. Beside a caterpillar, we evolved central pattern generated behaviour for several different organism morphologies. The population size of each robot was ten and spiking neural networks are structurally mutated over 30 generations for 800 ticks evaluation time. Figure 11 shows the individual hinge positions of the five robots in the caterpillar-like configuration in one of the evaluation phases. The resulting behaviour, emerged by the individual hinge movement, is depicted in the sequence of Figure 10. A caterpillar is moving from the centre towards the left side in order to leave the sight of view.

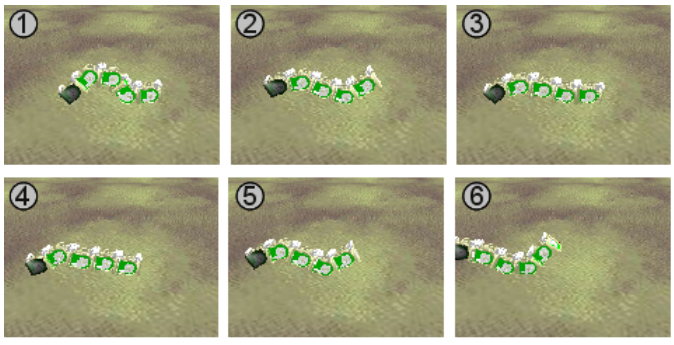

Fig. 10. The sequence shows the final behaviour of a caterpillar-like evolved locomotion.

 ISBN: 978-1-61208-274-5

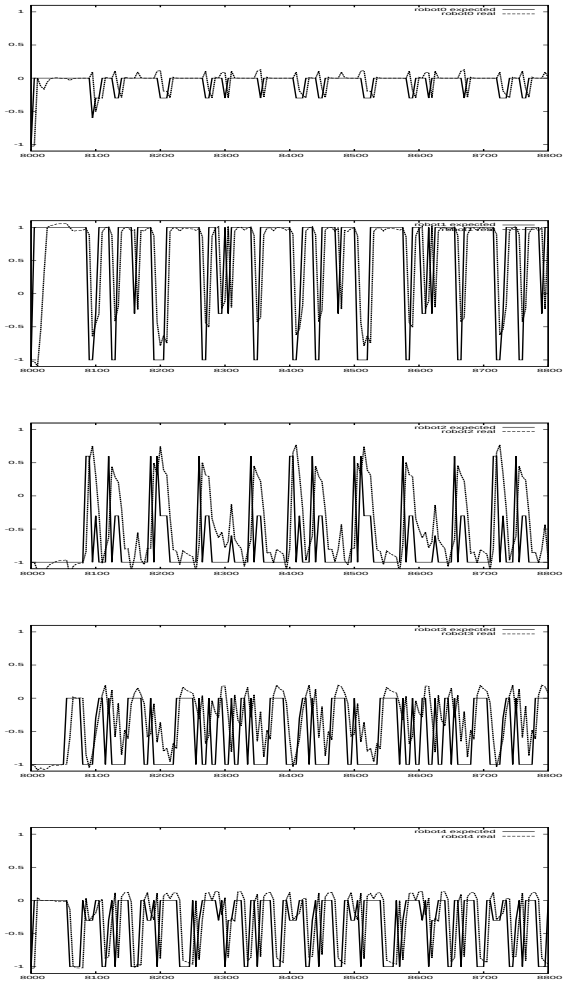

Fig. 11. The hinge positions in the evolved caterpillar-like robot organism.

VII. Conclusion

In this paper, we presented a framework for evolutionary robotics with the special focus on structural on-line and on-board evolution of neural network controllers. This framework supports standard neural networks as well as spiking neural networks in both generation based and tournament selection based evolutionary design processes. We have applied the framework to different scenarios in simulation and real robots to proof the feasibility. In future work, we will investigate the influence of structural mutation in more detail and will focus on the comparison of standard neural networks and spiking networks.

Acknowledgment

The "SYMBRION" project is funded by the European Commission within the work programme "Future and Emergent Technologies Proactive" under the grant agreement no. 216342. The "REPLICATOR" project is funded within the work programme "Cognitive Systems, Interaction, Robotics" under the grant agreement no. 216240.

References

[1] S. Murata and H. Kurokawa, "Self-reconfigurable robots," *Robotics Automation Magazine, IEEE*, vol. 14, no. 1, pp. 71–78, 2007.

[2] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. Chirikjian, "Modular self-reconfigurable robot systems [grand challenges of robotics]," *Robotics Automation Magazine, IEEE*, vol. 14, no. 1, pp. 43–52, 2007.

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

[4] Symbrion: Symbiotic evolutionary robot organisms, 7th framework programme project no fp7-ict-2007.8.2, 2008-2013. http://www.symbrion.eu , visited on May 17th 2013.

[5] Replicator: Robotic evolutionary self-programming and self-assembling organisms, 7th framework programme project no fp7-ict-2007.2.1, 2008-2013. http://www.replicators.eu , visited on May 17th 2013.

[6] F. Schlachter, E. Meister, S. Kernbach, and P. Levi, "Evolve-ability of the robot platform in the symbrion project," in *SASOW: Conference on Self-Adaptive and Self-Organizing Systems Workshops*. IEEE Computer Society, 2008, pp. 144–149.

[7] F. Schlachter, C. Schwarzer, B. Girault, and P. Levi, "A Modular Software Framework for Heterogeneous Reconfigurable Robots ," in *P. Levi et al. (eds.), Autonomous Mobile Systems, AMS*, 2012.

[8] S. Kernbach, O. Scholz, K. Harada, S. Popesku, J. Liedke, R. Humza, W. Liu, F. Caparrelli, J. Jemai, J. Havlik, E. Meister, and P. Levi, "Multi-robot organisms: State of the art," *CoRR*, vol. abs/1108.5543, 2011.

[9] S. Kernbach, F. Schlachter, R. Humza, J. Liedke, S. Popesku, S. Russo, T. Ranzani, L. Manfredi, C. Stefanini, R. Matthias, C. Schwarzer, B. Girault, P. Alschbach, E. Meister, and O. Scholz, "Heterogeneity for increasing performance and reliability of self-reconfigurable multi-robot organisms," *CoRR*, vol. abs/1109.2288, 2011.

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

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

[12] Y. Kassahun, J. Metzen, M. Edgington, and F. Kirchner, "Incremental acquisition of neural structures through evolution," in *Design and Control of Intelligent Robotic Systems*, ser. Studies in Computational Intelligence. Springer, 2009, pp. 187–208.

[13] Y. Kassahun, M. Edgington, J. H. Metzen, G. Sommer, and F. Kirchner, "A common genetic encoding for both direct and indirect encodings of networks," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, ser. GECCO '07. New York, NY, USA: ACM, 2007, pp. 1029–1036.

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

[15] C. Schwarzer, F. Schlachter, and N. K. Michiels, "Online evolution in dynamic environments using neural networks in autonomous robots," *Internation Journal on Advances in Intelligent Systems*, vol. 4, no. 3&4, 2011.

[16] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, pp. 1659–1671, 1996.

[17] N. Bredeche, E. Haasdijk, and A. Eiben, "On-line, on-board evolution of robot controllers," in *Proceedings of the 9th international conference on Artificial Evolution (Evolution Artificielle - EA'09)*, 2009.

[18] K. Deuschl, "Evolution of coordinated behavior in a heterogenous robot swarm," *Diploma Thesis*, University of Stuttgart, 2012.

[19] "Playerstage," Website, 2013, available online at https://launchpad.net/robot3d; visited on May 17th 2013.

[20] P. Alschbach, "Online evolution and adaptation of central pattern generators for multi-robot organisms," *Diploma Thesis*, University of Stuttgart, 2012.

[21] "Robot3d, open source modular swarm robot simulation engine," Website, 2013, available online at https://launchpad.net/robot3d; visited on May 17th 2013.

 ISBN: 978-1-61208-274-5