Stability of on-line and on-board evolving of adaptive collective behavior

L. König, K. Jebens, S. Kernbach, P. Levi

Institute of Parallel and Distributed Systems, University of Stuttgart,
Universitätsstr. 38, D-70569 Stuttgart, Germany
lukas-koenig@gmx.net,{jebenskf,korniesi,levipl}@ipvs.uni-stuttgart.de

Abstract. This work focuses on evolving purposeful collective behavior in a swarm of Jasmine micro-robots. We investigate the stability of the on-line and on-board evolutionary approaches, where mutation, crossover as well as fitness calculation are performed only by interacting micro-robots without using any centralized resources. In this work it is demonstrated that the environment-adaptive collective behavior can be obtained, where the evolving fitness and behavior are partially stable. To increase stability of the approach, some reduction methodology of the search space is proposed.

1 Introduction

An important challenge in modern network researching and swarm robotics is a design of purposeful collective behavior (Kornienko *et al.*, 2004). Such a behavior should be technically useful, adaptive to environmental changes and scalable in size and functional metrics (Constantinescu *et al.*, 2004). One possible paradigm here is to use evolutionary approaches for evolving desired collective behavior (Koza, 1992). Using an evolutionary approach, robots can start with simple behavior primitives and gradually increase their cooperative complexity until the collective behavior satisfies some imposed fitness.

The application of evolutionary approaches in swarm robotics is known. Some essential references can be given in works of evolving control (Floreano *et al.*, 2008), evolving shapes (Lipson & Pollack, 2000), evolving communication (Wischmann & Pasemann, 2006), (Marocco & Nolfi, 2006) and others. However, essential obstacles for successful application of evolutionary approaches are ”on-line” and ”on-board” requirements imposed on the calculation of fitness and execution of evolutionary operators. The ”on-line” requirement means that all evolutionary results should be obtained during the life-cycle of a robot, ”on-board” means that only available on-board sensors, computational and communication resources can be used. Both requirements originate from the practical robotic field.

The present work focuses on the problems of stability in on-line and on-board evolving of collective behavior. More exactly, we assume that there exists some collective behavior with relatively high fitness. The questions are whether

on-line and on-board evolutionary processes: (a) will preserve this originally effective behavior? (b) will replace the original behavior with a better one? (c) will destroy the original behavior without creating a better one? Answering these questions, we intend to acquire more insight about possibilities of performing on-line and on-board artificial evolution.

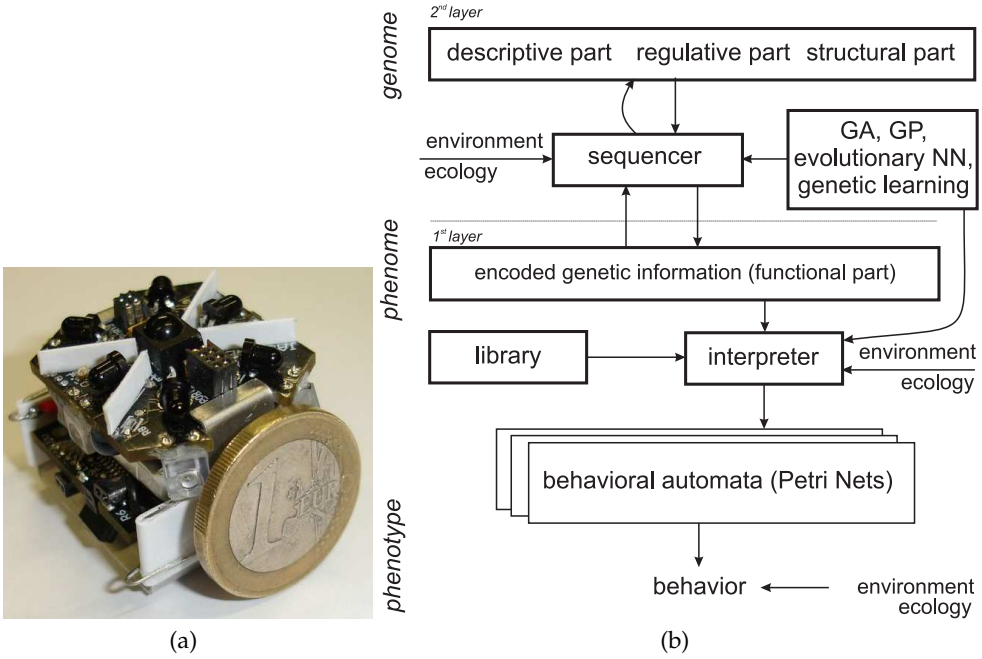
In this work, the robot controller is represented as a generating hierarchy of $genome \xrightarrow{generator} phenome \xrightarrow{interpretor} behavioral\ automaton$. The behavioral automaton is based on a finite Moore automaton (Fogel *et al.*, 1995). At each state an "atomic action" (e.g. *move*, *stop*, *turn left*) is produced. For encoding the Moore automaton a symbolic string, called *phenome* is used. The phenome contains the same information as the actual automaton and can be used to copy a behavior between different robots using local robot-robot communication. The genome contains generating rules for the phenome sequence. To simplify treating of on-board and on-line issues, we apply evolutionary operators only to *phenome strings* from one or different robots. The fitness is selected as a measure of collision, so that the final goal is to evolve an effective collision avoidance behavior. Since evolving the behavior is only possible when many robots interact, collision avoidance represents a result of collective behavior. Experiments are performed using Jasmine IIIp micro-robots.

The rest of the paper is organized as follows. The framework of evolutionary experiments is described in Sec. 2, evolutionary operators in Sec. 3 and experiments in Sec. 4. In Sec. 5 we discuss and conclude this work.

2 Hardware and software framework

In this section we briefly describe the used hardware and software framework. Experiments are performed with the Jasmine IIIp micro-robots, see Fig. 1(a). Each micro-robot has two Atmel AVR-microcontrollers with 8 MIPS each and totally 2 kB RAM, 24 kB Flash memory and 1 kB non-volatile memory. For the evolutionary approach only 6.5 kB Flash and 700 bytes RAM are available, the phenome strings are written in 512 bytes non-volatile memory. Each robot has a local 360° IR-based communication with an effective communication radius of about 2-10 cm. When two robots meet within this radius, they establish a bi-directional communication channel and can exchange phenome strings at 500 bytes/sec. The communication system can also be used for proximity sensing (see more on www.swarmrobot.org).

The software framework consists of a low-level BIOS (interface to hardware) and an operational system (Kornienko *et al.*, 2005), and the high-level genetic framework. For the genetic framework there are several design alternatives, e. g. to use classical robot control, to use paradigms from the evolutionary community or to use bio-inspired ideas. We have chosen the bio-inspired way, firstly, due to interest of exploring alternatives to well-known solutions, secondly, because this framework is used in the projects (SYMBRION, 2008-2012) and (REPLICATOR, 2008-2012), which are related to bio-inspired artificial evolution.



(a) (b)

Fig. 1. (a) Micro-robot Jasmine IIIp. **(b)** Structure of the high-level genome framework.

The structure of the genome framework is shown in Fig. 1(b). The behavior of the robot is controlled by a behavioral automaton (Petri-nets in the overall framework (Kornienko *et al.*, 2005)) and is influenced by the environment. We call this mutual influence the ***phenotype*** of the robot. Behavioral automata represent an explicit description of the phenotype. We call this the ***phenome***. This is a contradiction to biological systems, where the phenome is not directly available. More exactly, the phenome is a symbolic string which generates the automata. This symbolic string contains direct low-level as well as high-level (from libraries) behavioral commands, therefore can be thought of as a functional descriptor. In turn, the phenome is generated by the ***genome*** of a robot. The genome is also a symbolic string and consists of descriptive, structural and regulative parts and usually does not contain any direct behavioral commands. By analogy, the genome is a structural descriptor of the system. In this framework, both genome and phenome can underly evolutionary operators and can also be influenced by the environment.

The two-layer control structure *genome-phenome* is very effective for reconfigurable multi-robot organisms, where the robot system can change its own structure and therefore the functionality (Kornienko *et al.*, 2007). However, in this work, where we do not change the structure of robots, evolutionary operators will be applied to the phenome. In the following we describe the theoretical approach for evolving the phenome and show experimental results.

3 Implementation of evolutionary framework

This section describes the theoretical model for our evolutionary approach. We use a finite automaton based model to describe robot behavior (Fogel *et al.*, 1995) and a genetic programming approach (Koza, 1992) for evolving new behaviors.

3.1 The automaton model

We denote a set of byte values and a set of positive byte values as $B = \{0, ..., 255\}$ and $B^+ = \{1, ..., 255\}$. The behavior of a robot depends on the sensor data. We assume a set H of n sensor variables $H = \{h_1, ..., h_n\}$. The sensor variables stand as placeholders for sensor data from real or virtual sensors (i. e. any internal variables of the robot). Every variable h_i can be set to a byte value.

As mentioned before, the main behavior of a robot is controlled by a finite Moore automaton[1]. At each state, an output is produced, which is interpreted as an *atomic instruction* (which can be a mechanical action like *move* or a whole C-program) to be performed by the robot. The transitions between states depend on the values of the sensor variables $h_1, ..., h_n$.

States. The set of states is denoted by $Q = \{q_0, ..., q_m\}$, where q_0 is the initial state of the automaton. A state contains the following information:

- an identification number $N \in B^+$,
- an atomic instruction $I \in B^+$ to be performed,
- an additional parameter $P \in B^+$ to gain more information about the instruction, and
- the definition of all outgoing transitions from that state.

Transitions. We associated a condition to each transition, which has to evaluate to *true* for the transition to be taken. We define for this purpose a set of conditions over the sensor variables as follows:

Definition 1. *Conditions*
A condition is an element of the set, defined by:

$$c ::= true \mid false \mid z_1 \triangleleft z_2 \mid (c_1 \circ c_2),$$

$z_1, z_2 \in B^+ \cup H$,
$\triangleleft \in \{<, >, \leq, \geq, =, \neq, \approx, \not\approx\}$, *where* $\approx$ *means the range of* $\pm$ *5.*
$\circ \in \{AND, OR\}$,
c_1, c_2 *are conditions themselves.*

The set of all conditions is denoted by C.

[1] Moore automaton with an underlying operational system and BIOS, which perform e.g. sensor data acquisition or interruption handling.

Valid conditions are e. g. $h_1 < h_2$, $((h_1 \not\approx h_2 \ AND \ h_3 = 104) \ OR \ h_5 \geq h_7)$, $(true \ \ AND \ \ false)$. The result of a condition (*"true"* or *"false"*) is calculated in the obvious way, by feeding the variables with actual sensor values and evaluating the comparisons and logical operations. A transition is taken, if its corresponding condition evaluates to *true*. At this point two special cases have to be considered:

1. A state can have no condition that evaluates to *true*.
2. A state can have more than one condition that evaluates to *true*.

In case 1, we define an implicite transition to the initial state to be taken. In case 2, the "first" outgoing transition of that state is taken (the corresponding order is not important, as long as the transition to be taken does not vary from case to case; we took the order in which the transitions were generated).

Moore automaton for robot behavior. A Moore automaton for robot behavior is built by states and transitions as mentioned above. Each state carries identification N, instruction I, parameter P and its outgoing transitions, which are defined by a condition $c \in C$ and a following state $f \in B^+$. The automaton is defined as follows:

Definition 2. *Finite Moore automaton for robot behavior*
A finite Moore automaton for robot behavior A is defined as follows:

$$A = (Q, \Sigma, \Omega, \delta, \lambda, q_0, F),$$

where:

- *The set of states* $Q = B^+ \times B^+ \times B^+ \times (C \times B^+)^*$, *C being the set of conditions. Let* $\forall q \in Q$*:*

$$q = \left(N^q, I^q, P^q, \left(c_1^q, f_1^q\right), ..., \left(c_{|q|}^q, f_{|q|}^q\right)\right),$$

  where $|q|$ *denotes the number of transitions of state q.*
- *The input alphabet* $\Sigma = H = (B^+)^n$.
- *The output alphabet* $\Omega = (B^+)^2$.
- *The transition function (for* $q \in Q$, $h \in H$*):* $\delta : Q \times H \rightarrow Q$:

$$\delta(q,h) = \begin{cases} q', & \text{if } \exists\, k \in \{1,...,|q|\} : f_k^q = q' \text{ and } c_k^q \text{ evaluates to } true \text{ under } h \\ & \text{and } \forall\ j \in \{1,...,|q|\}, \text{ where } c_j^q \text{ evaluates to } true \text{ under } h, \\ & \text{it holds: } k \leq j, \\ q_0 & \text{otherwise} \end{cases}$$

- *The output function* $\lambda : Q \rightarrow (B^+)^2 : \lambda(q) = (I^q, P^q)$ *(for* $q \in Q$*).*
- *The initial state* q_0.
- *The empty set of final states:* $F = \emptyset$.

3.2 Evolutionary operators

Mutation. We developed a mutation operator, which is *complete* and *smooth*, which means that every part of the search space is reachable and that **every single mutation causes only a small step in the search space**. The mutation operator consists of 10 atomic mutations, one of which is randomly chosen, when the operator is used. The atomic mutations are:

1. Toggle inactive transitions:
   (a) Remove a random transition, associated with the condition $false$.
   (b) Add a random transition, associated with the condition $false$.
2. Remove a state:
   (a) Without incoming transitions.
   (b) With all outgoing transitions being associated with the condition $false$ and the state being associated with the instruction *IDLE*.
3. Add a new state with:
   – no incoming transitions, and
   – no outgoing transitions, and
   – arbitrary action, and
   – parameter $\leq k$ ($k \in \{1, ..., 255\}$ being a constant).
4. Change a condition: Let $a, b \in \{1, ..., 255, h_1, ..., h_{|H|}\}$, $A \in C$ a condition. Every part of a condition that matches the following patterns can be mutated:
   (a) $false \leftrightarrow a = b \leftrightarrow a \approx b \leftrightarrow \begin{matrix} a \leq b \leftrightarrow a < b \\ a \geq b \leftrightarrow a > b \end{matrix} \leftrightarrow a \not\approx b \leftrightarrow a \neq b \leftrightarrow true$
   (b) One of the following:

$$\begin{array}{rcccl} (A\ AND\ true) & \leftrightarrow & A & \leftrightarrow & (true\ AND\ A) \\ (A\ OR\ true) & \rightarrow & true & \leftarrow & (true\ OR\ A) \\ (A\ AND\ false) & \rightarrow & false & \leftarrow & (false\ AND\ A) \\ (A\ OR\ false) & \leftrightarrow & A & \leftrightarrow & (false\ OR\ A) \end{array}$$

   (c) Let i be a number in a condition. Let $i' = i + rand[-k, k]$, $k \in \{1, ..., 255\}$ being a constant parameter. Mutate:

$$i \rightarrow \begin{cases} i', & \text{if } 1 \leq i' \leq 255 \\ 1, & \text{if } i' < 1 \\ 255, & \text{if } i' > 255 \end{cases}.$$

   (d) Let h_i be a sensor variable in a condition. Mutate:

$$h_i \rightarrow h_{rand[1,...,|H|]}.$$

5. Change a state: Let $I_1, I_2, ..., I_l$ be the set of instructions, $(N, I, P, *Trans*)$ a state.
   Mutate: $(N, I, P, *Trans*) \rightarrow (N, J, (P + m) \mod 255 + 1, *Trans*)$, where $m = rand[-k, k]$, $k \in B^+$ a constant parameter,

$$J = \begin{cases} I, & \text{if } P + m > 1 \\ I_{rand[1,l]} & \text{otherwise} \end{cases}.$$

Crossover. We used a simple crossover operator, where two parental phenomes produce one child phenome, which is a clone of the better parent phenome. A usual crossover operator has not been implemented due to insufficient RAM memory resources (1 kB only).

3.3 Fitness Function

For each behavior to evolve, a separate fitness function has to be defined. Concerning the on-board, on-line approach, some problems arise:

1. The fitness cannot be calculated exactly from the phenome, since the environmental influence cannot be calculated.
2. Global fitness cannot be calculated because there is no central instance, which collects information about the whole population.
3. There exists a delayed fitness, i.e. fitness can be first approximated after a certain time period.
4. There may be unknown anomalies of a robot's and an environment's physical properties, which have influences on the fitness function.

For the experiments, we have developed a fitness function, which measures the goodness of a collision avoidance behavior.

```
void CalcFitness_CollAvoid(void) {
    fitness += 2;
    if (LastAction != MOVE_FLAG) fitness -= 1;
    for (int i = 1; i < 7; i++) {
        if (Sensor(i) > 100) {
            fitness -= 1;
            break;
        }
    }
    if (time_in_s % 30 == 0) fitness /= 2;
}
```

MOVE_FLAG indicates, whether the last action was *move*; the function *Sensor*(*i*) returns the value of the *i*-th sensor. The fitness value is changed each time the fitness function is executed (5 times per second), depending on the last performed instruction and the distance to the nearest obstacle. The idea is to keep in motion, but away from obstacles. Assuming such a capability can evolve evolutionary, it should be stable and therefore not disappear. Once each 30 seconds the fitness is divided by 2 so that the fitness points from earlier behaviors cannot be accumulated. Initially, the fitness is equal to zero, but within the first minute of the experiment, it approximates the current behavior.

4 Experiments

We have conducted six experiments to check whether the selective process is stable enough to keep a once developed behavior through the whole experiment. Each experiment was performed with 20 robots, which initially had a

collision avoidance automaton. The arena size was $70 \times 115\ cm^2$ of rectangular geometry, mutation was performed each 15 seconds, crossover was performed when robots meet each other, duration of each experiment was 25 min.

In Fig. 2(a)-(c) we show three different states of the fifth experiment: 0, 13 and 25 minutes from the beginning. There were several commonalities in all

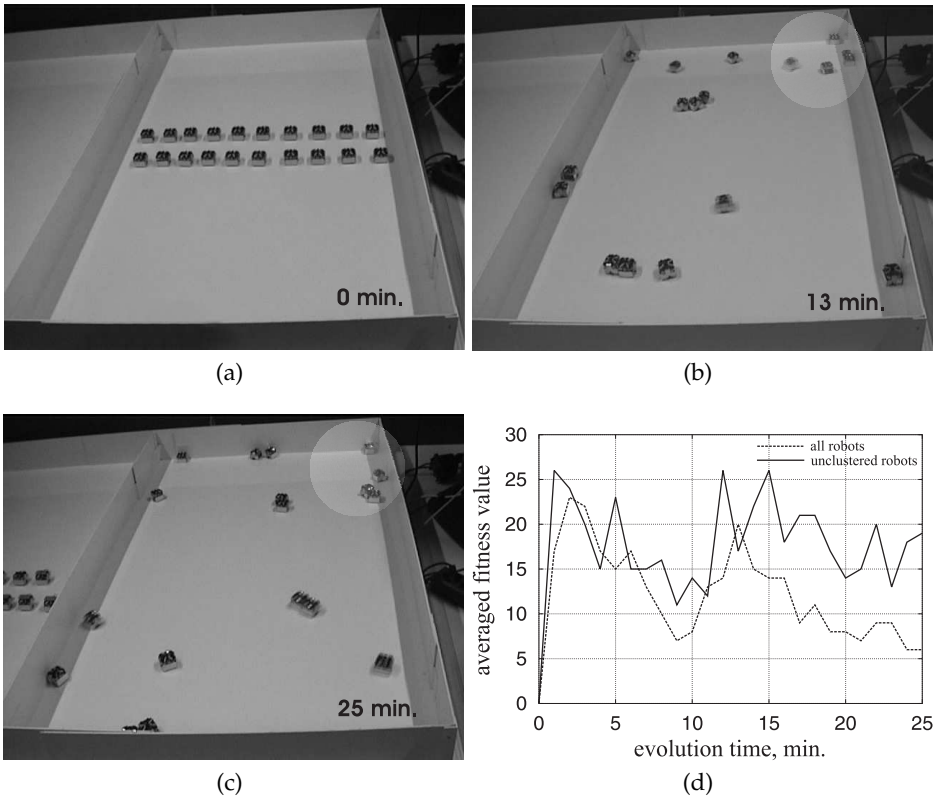


Fig. 2. Course of the experiment: **(a)** Start; **(b)** 13 minutes; **(c)** Finish; **(d)**; The light spot in (b) and (c) shows clustered robots. Changes of the averaged fitness value during 25 min. of experiments.

experiments: several robots showed a behavior which can be described as collision avoidance, but different than with the initial automaton. Most robots, however, clustered in corners and were principally dead for the experiment, see marked spots in upper right corner in Figs. 2(b)-(c). Some robots showed an extremely robust wall following behavior (which is related to collision avoidance). Although wall following has been implemented in the past several times manually, we were never able to implement it in such a robust and fluent way.

Fig. 2(d) shows the fitness averaged through all experiments. The robots can be divided into two groups: moving ones or ones caught in clusters. The clustered robots, shown by the light spot in Figs. 2(b),(c), are dissociated from the evolutionary process. The dotted curve shows the fitness for *all robots*. This curve decreases averagely through all six experiments. The solid curve shows the average fitness only over *robots, which were not stuck in clusters* and, therefore, were still participating in the evolutionary process. The average fitness of these robots stays at a similar level until the end of the experiments.

In the end of the experiments, more than 75% of the unclustered robots had a positive fitness. More than 33% showed an observable behavior close to collision avoidance. However, other robots also had automata that were close to a sort of collision avoidance, but were unable to show this behavior because of hardware or environmental anomalies.

These experiments indicate several important properties:

1. It appears, that the selective pressure is not sufficient, since a majority of robots is lost in corners. Most likely the reason is an insufficient selection process since the communication between robots has hardware difficulties in large clusters of robots. This anomaly, effecting the search space, is also influencing the fitness function as these robots change the environment of the other robots by blocking them. Other anomalies are also thinkable: e. g. one robot might shift another robot "blindly" and make its movement instructions ineffective.
2. Though the number of experiments is statistically low and the population size is small, the appearance of several very good solutions could point to a limited search space.

5 Discussions

The experiments have shown, that an on-board and on-line evolution is partially capable of keeping a once developed behavior. Problems lie in the clustering of robots, which cannot move and communicate. These robots dramatically decrease collective fitness. The reason for clustering is diverse: robots hook each other with docking connectors, anomalies in sensor data, the delayed fitness, limited communication capability in clusters and so on. Additional experiments need to be performed to fix hardware and communication anomalies.

However, it is demonstrated that experiments with robots cannot be performed in a statistically significant manner. It is hardly possible to perform even 100 experiments in a large population of micro-robots. It seems that there is a need of finding new approaches by combining evolutionary paradigms with other non-evolutionary techniques. For example, the fitness measurement can be improved by using several plausibility checks, e. g. for a collision avoidance the robot could first check if there even exists a *move* instruction in its automaton and otherwise continue to mutate. Such a virtual measurement combined with the actual measurement could even more dramatically reduce the search space. This seems to be a realistic alternative for evolutionary approaches in robotics.

The last issue to be mentioned is the possibility to perform a robotic evolution first in simulation and then to copy the solution to the real robots. The problem is that the fitness calculation is embedded into the environment. Since we are unable to reproduce the complexity of the real environment in the simulative one, it could be expected that the evolved behavior cannot achieve the same qualities as in the real environment. However, the following experiment is thinkable: the real sensor data from robots can be transferred via wireless communication into an evolutionary simulation. It could be expected that two scenarios appear: either the evolution will develop similar solutions in real and virtual environments or even a small inaccuracy in simulation can create an essential change in the quality of the evolved behavior.

The next work will deal with a more sophisticated crossover operator, which will be applied when some spatial task is achieved successfully. In this way a more collective form of behavior can be evolved, moreover spatial statistics can be collected.

References

Constantinescu, C., Kornienko, S., Kornienko, O., & Heinkel, U. 2004. An agent-based approach to support the scalability of change propagation. *Pages 157–164 of: Proc. of ISCA04*.

Floreano, Dario, Husbands, Phil, & Nolfi, Stefano. 2008. Evolutionary Robotics. *In: Handbook of Robotics*. Berlin: Springer Verlag.

Fogel, L.J., Angeline, P.J., & Fogel, D.B. 1995. An Evolutionary Programming Approach to Self-Adaptation on Finite State Machines. *Pages 355–365 of: Evolutionary Programming*.

Kornienko, S., Kornienko, O., & Levi, P. 2004. About nature of emergent behavior in micro-systems. *Pages 33–40 of: Proc. of the Int. Conf. on Informatics in Control, Automation and Robotics (ICINCO 2004), Setubal, Portugal*.

Kornienko, S., Kornienko, O., & Levi, P. 2005. IR-based communication and perception in microrobotic swarms. *In: Proc. of the IROS 2005, Edmonton, Canada*.

Kornienko, S., Kornienko, O., Nagarathinam, A., & Levi, P. 2007. From real robot swarm to evolutionary multi-robot organism. *In: Proc. of the CEC2007, Singapure*.

Koza, J. 1992. *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge, Massacgusetts, London, England.

Lipson, H., & Pollack, J.B. 2000. Automatic design and Manufacture of Robotic Lifeforms. *Nature*, **406**, 974–978.

Marocco, D., & Nolfi, S. 2006. Origins of Communication in Evolving Robots. *Pages 789–803 of:* et al., Nolfi S. (ed), *SAB06*. Springer Verlag.

REPLICATOR. 2008-2012. *REPLICATOR: Robotic Evolutionary Self-Programming and Self-Assembling Organisms, 7th Framework Programme Project No FP7-ICT-2007.2.1*. European Communities.

SYMBRION. 2008-2012. *SYMBRION: Symbiotic Evolutionary Robot Organisms, 7th Framework Programme Project No FP7-ICT-2007.8.2*. European Communities.

Wischmann, S., & Pasemann, F. 2006. The Emergence of Communication by Evolving Dynamical Systems. *Pages 777–788 of:* Nolfi, S., Baldassarre, G., Calabretta, R., Hallam, J., Marocco, D., Meyer, J.-A., & Parisi, D. (eds), *From animals to animats 9: Proceedings of the Ninth International Conference on Simulation of Adaptive Behaviour*. LNAI. Springer Verlag.