Swarm Intell
DOI 10.1007/s11721-014-0092-4

# AutoMoDe: A novel approach to the automatic design of control software for robot swarms

**Gianpiero Francesca · Manuele Brambilla · Arne Brutschy · Vito Trianni · Mauro Birattari**

Received: 26 October 2013 / Accepted: 14 February 2014


**Abstract** We introduce AutoMoDe: a novel approach to the automatic design of control software for robot swarms. The core idea in AutoMoDe recalls the approach commonly adopted in machine learning for dealing with the bias–variance tradedoff: to obtain suitably general solutions with low variance, an appropriate design bias is injected. AutoMoDe produces robot control software by selecting, instantiating, and combining preexisting parametric modules—the injected bias. The resulting control software is a probabilistic finite state machine in which the topology, the transition rules and the values of the parameters are obtained automatically via an optimization process that maximizes a task-specific objective function. As a proof of concept, we define AutoMoDe-`Vanilla`, which is a specialization of AutoMoDe for the e-puck robot. We use AutoMoDe-`Vanilla` to design the robot control software for two different tasks: aggregation and foraging. The results show that the control software produced by AutoMoDe-`Vanilla` (i) yields good results, (ii) appears to be robust to the so called *reality gap*, and (iii) is naturally human-readable.

**Keywords** Swarm robotics · Automatic design · AutoMoDe · Evolutionary robotics

G. Francesca (✉) · M. Brambilla · A. Brutschy · M. Birattari (✉)
IRIDIA, Université Libre de Bruxelles, Brussels, Belgium
e-mail: gianpiero.francesca@ulb.ac.be

M. Brambilla
e-mail: mbrambil@ulb.ac.be

A. Brutschy
e-mail: arne.brutschy@ulb.ac.be

M. Birattari
e-mail: mbiro@ulb.ac.be

V. Trianni
ISTC-CNR, Rome, Italy
e-mail: vito.trianni@istc.cnr.it

Published online: 12 March 2014

## 1 Introduction

In this paper, we introduce a novel approach to automatically generate control software for robot swarms.

In swarm robotics (Şahin 2005; Dorigo et al. 2014), a large number of robots cooperate and accomplish a task that a single individual would be unable to accomplish. A robot swarm is a highly redundant system that acts in a self-organized way without the need of any form of centralized coordination. The collective behavior of the swarm is the result of the local interactions that each robot has with its neighboring peers and with the environment.

The self-organized and distributed nature of robot swarms makes them challenging to design. The requirements are typically expressed at the swarm level by specifying the task that the swarm, as a whole, has to perform. However, the swarm is a collective entity and, as such, it is an immaterial concept. In particular, the swarm itself cannot be *programmed*, only the individual robots can. The designer's task is therefore *indirect*: he has to design the individual-level behaviors of the robots that, through a complex set of robot–robot and robot–environment interactions, result in the desired collective behavior of the swarm.

At the moment, there is no general approach to the design of robot swarms, even though some preliminary proposals have been made (Hamann and Wörn 2008; Kazadi et al. 2009; Berman et al. 2011; Brambilla et al. 2012). Currently, most robot swarms are designed by hand using a trial-and-error process: an individual-level behavior is iteratively improved and tested until the desired collective behavior is obtained. This approach is closer to craftsmanship than to engineering: the quality of the result strongly depends on the experience and intuition of the designer. Moreover, this trial-and-error process is time consuming, costly, and lacks repeatability and consistency.

An alternative way to develop robot swarms is to rely on automatic design. To date, the main automatic design approach that has been adopted in swarm robotics is *evolutionary robotics* (Nolfi and Floreano 2000). In this approach, an evolutionary algorithm is used to obtain the parameters of a neural network that maps the sensor readings of the individual robot into values fed to its actuators. A large literature shows that evolutionary robotics is able to produce robot swarms that can perform a number of tasks—for a review, see Brambilla et al. (2013).

Nonetheless, evolutionary robotics presents some known limitations, and an engineering methodology for the design of robot swarms via evolution is still lacking (Trianni and Nolfi 2011). Most importantly, in the context of swarm robotics, the evolutionary approach has not demonstrated the capability of scaling in complexity and providing solutions for realistic applications. Among the causes, we reckon the difficulty in overcoming the reality gap, that is, having a seamless transition from simulation—the main tool for evolutionary design—to the real world.

In this paper, we conjecture that the observed limitations of evolutionary robotics result from an uncontrolled *representational power* of the control architecture that is typically adopted in evolutionary robotics. Indeed, one of the tenets of evolutionary robotics is to minimize the assumptions and the bias injected by the designer (Harvey et al. 1997; Nolfi and Floreano 2000; Bongard 2013). The idea is to rely on an evolutionary process to fine-tune the dynamics of the interaction between the robot and the environment. To this aim, a common assumption in the literature is the need of a control architecture that features a high representational power—for example, a neural network.

Unfortunately, a high representational power may be counter-productive in the peculiar working conditions faced in swarm robotics, which are highly dynamic and uncertain due to

the numerous robot–robot interactions. We claim that such working conditions offer limited regularities to be discovered and exploited by the evolutionary process. As a consequence, it is likely that evolution will produce control software that exploits "idiosyncratic features" (Floreano and Keller 2010) of the simulation, that is, the differences between simulation and reality, which unavoidably occur. This control software will generalize poorly and will be unable to overcome the reality gap. Because the *a priori* identification of the differences between simulation and reality is in general difficult, an automatic design approach must be as robust as possible to their presence. With the goal of obtaining a robust automatic design approach, we cast the generalization problem in terms of the bias–variance tradeoff formalized in the machine learning literature (Geman et al. 1992). In Sect. 3, we discuss the implications of the bias–variance tradeoff on the automatic design of robot swarms, and we conjecture that the generalization problem can be tackled through a suitable injection of bias in the control architecture adopted by the automatic design process.

Driven by our conjecture, we develop a novel automatic design approach for robot swarms, called AutoMoDe (automatic modular design). AutoMoDe generates an individual-level behavior in the form of a probabilistic finite state machine by searching for the best combination of preexisting parametric *modules*. In other words, AutoMoDe develops control software by selecting, via an optimization algorithm, the topology of the probabilistic finite state machine, the modules to be included, and the value of their parameters. The set of modules and the rules to compose a probabilistic finite state machine represent the bias injected in the automatic design process. As a result of the injection of bias, we conjecture that the variance of the behaviors designed by AutoMoDe will be consequently reduced.

As a proof of concept, we present AutoMoDe-`Vanilla`, which is a specialization of AutoMoDe for the e-puck robot (Mondada et al. 2009a,b). More precisely, AutoMoDe-`Vanilla` is specialized for a given *reference model* of the e-puck. This reference model formally describes the characteristics of the robot and the functionalities that are made available to the control software.

We evaluate AutoMoDe-`Vanilla` using two tasks commonly studied in the swarm robotics literature: aggregation and foraging. The results obtained show that AutoMoDe-`Vanilla` automatically designs control software that allows the swarm to successfully accomplish the two tasks. Moreover, the control software designed by AutoMoDe-`Vanilla` is naturally understandable for a human.

The rest of the paper is organized as follows: In Sect. 2 we discuss the related work. In Sect. 3 we discuss the bias–variance tradeoff in the automatic design of robot swarms. In Sect. 4, we introduce AutoMoDe and AutoMoDe-`Vanilla`. In Sect. 5, we describe the experimental protocol and the setup we used to evaluate AutoMoDe-`Vanilla`. In Sect. 6 we present the results we obtained and in Sect. 7 we discuss them. Finally, in Sect. 8 we draw some conclusions and we highlight directions for future work.

## 2 Related work

In this section, we provide a brief overview of the main design approaches for swarm robotics, namely the behavior-based approach and evolutionary robotics. Both have been developed and largely applied in the single-robot domain, and subsequently adopted in swarm robotics. In this context, they have shown some advantageous features and some limitations, which are discussed below. For an extensive review of the swarm robotics literature, we refer the reader to Brambilla et al. (2013).

*The behavior-based approach* In the behavior-based approach, the designer develops the robot control software manually via trial and error. The control software is organized in a modular architecture that is largely inspired by Brooks' *subsumption architecture* (Brooks 1986). The application of the behavior-based approach to swarm robotics is straightforward, and has been the most common choice to date (Brambilla et al. 2013). However, the behavior-based approach does not address the core design problem that one faces in swarm robotics: it does not provide any guideline to define what the individual robot should do so that the given swarm-level specifications are met. Some ideas to address this core design problem have been proposed (Hamann and Wörn 2008; Kazadi et al. 2009; Berman et al. 2011; Brambilla et al. 2012) but they often rely on strong assumptions and their generality is limited, as they typically refer to specific tasks.

Reinforcement learning coupled with the behavior-based approach has been successfully applied in the single-robot domain to arbitrate low-level behaviors—see for instance Maes (1991). In the multi-robot domain, reinforcement learning was also applied with some success (Parker 1996; Matarić 1997a,b). See Panait and Luke (2005) for a review. Nonetheless, the application of reinforcement learning to swarm robotics presents several major challenges, among which the difficulty to reward the individual contribution to the global behaviour (i.e., the credit assignment problem), the large size of the state space, and the dynamic and uncertain working conditions—for a discussion, see Brambilla et al. (2013).

*Evolutionary robotics* Evolutionary robotics is an automatic design approach that applies artificial evolution to the development of robot control software (Nolfi and Floreano 2000). A number of robot swarms have been designed using evolutionary robotics. For example, Baldassarre et al. (2007) used evolutionary robotics to design a coordinated motion behaviour, and Trianni and Dorigo (2006) extended this behaviour with hazard-avoidance capabilities. Trianni and Nolfi (2009) developed a synchronization behavior. Hauert et al. (2008) designed control software for a swarm of aerial robots to create a communication network.

Besides being used as a design approach, evolutionary robotics is also used to shed light on questions of relevance in evolutionary biology. Marocco and Nolfi (2007) studied the evolution of communication for solving a collective navigation problem. Waibel et al. (2009) studied the performance of homogeneous and heterogeneous teams evolved under individual and collective selective pressure. Tuci (2009) studied linguistic interaction between agents to form common perceptual categories. Mitri et al. (2011) used evolutionary robotics to investigate the correlation between genetic relatedness and reliability of evolved signaling strategies in a foraging scenario. Winfield and Erbas (2011) used evolutionary robotics to explore imitation and the emergence of a sort of artificial culture in multi-robot systems. Finally, Wischmann et al. (2012) studied the evolutionary development of robust and efficient communication strategies in robot swarms.

The suitability of evolutionary robotics as an automatic design approach in swarm robotics is discussed in Sect. 3. Before concluding this brief review, it is worth mentioning that several attempts to marry behavior-based and evolutionary approaches have been made in the single-robot domain (e.g., Urzelai et al. 1998). Recently, Duarte and co-workers used a neural network to arbitrate either hand-coded low-level behaviors (Duarte et al. 2012a) or other neural networks (Duarte et al. 2012b). Riano and McGinnity (2012), in the context of robot manipulators, used artificial evolution to obtain a probabilistic finite state machine composed of predefined low-level behaviors. A notable example of the evolution of behavior-based control software in swarm robotics is an original approach based on grammar evolution (Ferrante et al. 2013). With this approach, a foraging behavior has been designed, but experiments were limited to simulation: no validation on robots has been provided to date.

## 3 The bias–variance tradeoff in the automatic design of robot swarms

A widely recognized problem in evolutionary robotics is the so-called *reality gap problem*: the control software developed in simulation does not produce the same behavior and performance when instantiated in the physical system. Several techniques have been proposed to mitigate this problem. For example, Miglino et al. (1995) increased the realism of simulation using samples of the responses of sensors and actuators of the robot; Jakobi (1997) suggested the inclusion of noise in the simulation of sensors and actuators and in the conditions experienced by the robots during the design process; more recently, Bongard et al. (2006) and Koos et al. (2013) alternated simulation with tests on the physical system to correct the simulator nuisances. The reality gap problem is a specific instance of a wider problem related to the generalization abilities of evolutionary robotics—and of any automatic design approach—that is, the overfitting of the solution to the particular conditions encountered during the design process. By continuously refining the control software in a subset of the possible operating conditions, solutions are obtained that match the specificities or idiosyncrasies of these conditions (Floreano et al. 2008).

Our contention is that the inability to generalize to unexperienced working conditions in the automatic design of robot swarms should be considered in the light of the bias–variance tradeoff, which is a well known concept developed in the domain of machine learning (Geman et al. 1992). With respect to the training of neural networks, it has been formally shown that a low bias—i.e., the potential capability of reproducing any input–output mapping—entails a high variance, that is, a hypersensitivity to contingent elements in the training set, which eventually results in overfitting and in the inability to generalize to an independent test set. In evolutionary robotics, the standard approach to the overfitting problem amounts to the attentive definition of the conditions experienced by the robots during the design process. This usually corresponds to the introduction of variability in order to remove the regularities that can cause overfitting (Floreano et al. 2008). In the single robot domain, Pinville et al. (2011) proposed to promote generalization by taking inspiration from the three data sets approach adopted in supervised learning. Overall, if variability is introduced in the working conditions experienced by the robot at design time, a low bias does not hinder generalization. In this context, one can adopt a control architecture that features a high representational power and therefore manage the robot–environment interaction in a very fine-grained way. This allows obtaining solutions that could not be obtained otherwise—for a discussion in the single-robot domain, see Nolfi (2002). It is interesting to note that the idea of injecting variability at design time has been adopted also in the training of neural networks within the domain of supervised learning: indeed, it has been formally shown that the addition of noise to training data is equivalent to a form of regularization (Bishop 1995).

Unfortunately, this idea does not scale well with the number and complexity of operating conditions (Floreano et al. 2008). In particular, this idea does not appear to be appropriate in swarm robotics because the number of states in which a robot swarm can be found increases exponentially with the number of its robots, and augments the conditions against which the control software must prove flexible and robust. In this context, it is not trivial to define opportunely-varied operating conditions so as to guarantee that the robot swarm can experience similar states sufficiently often. Additionally, the highly dynamic working conditions faced in swarm robotics offer limited regularities to be found by the evolutionary process. Also, any small modification in the control software may lead to strong changes in the robot–robot interactions, hindering a progressive refinement of the global behavior. This does not mean that evolutionary robotics is deemed to failure, as the several successful

experiences demonstrate (Trianni and Nolfi 2011; Brambilla et al. 2013). Instead, it means that the design problem is worsened, and the variance of the obtained solutions increased.

The mainstream approach adopted in supervised learning to deal with the bias–variance tradeoff amounts to limiting the variance by injecting an appropriate bias (Dietterich and Kong 1995). In practice, this means restricting the representational power while trying, at the same time, to preserve the ability of representing the system at hand. Indeed, simple approximators characterized by a low representational power often perform better than competitors that display a much higher representational power (Geman et al. 1992).

We believe that the injection of bias is a suitable strategy to be explored in swarm robotics. As discussed above, automatic design approaches might be unable to properly exploit the high representational power offered by a control architecture that enables a fine-grained control of the robot–environment interaction. By reducing the representational power, and therefore working at a coarser level of granularity, it is possible to reduce the overfitting problem and obtain effective solutions. In robotics, this amounts to generating solutions that overcome the realty gap and that can be effectively used in real-world applications.

In this paper, we propose an automatic design approach in which the control software has a much reduced representational power with respect to those so far adopted in evolutionary robotics, but proves able to produce behaviors that are of interest in swarm robotics.

## 4 AutoMoDe

AutoMoDe (automatic modular design) is an approach to automatically generate modular control software in the form of a probabilistic finite state machine. We chose probabilistic finite state machines as a control architecture because they are commonly used in the manual design of robot swarms due to their modularity and readability. Probabilistic finite state machines are composed of *states* and *transitions*. In AutoMoDe, states are chosen among a set of preexisting *constituent behaviors* and transitions are defined on the basis of a set of preexisting *conditions*. In the following, we will collectively refer to constituent behaviors and conditions as *modules*. AutoMoDe automatically searches for the best combination of modules to perform a given task.

Each constituent behavior is an activity that the robot can perform. Constituent behaviors have a set of parameters that regulate their internal functioning. Parameters allow AutoMoDe to fine-tune constituent behaviors and fit different situations. Different instances of the same constituent behavior can be obtained by assigning different values to the parameters and can coexist in the same probabilistic finite state machine.

Conditions are used to trigger transitions from a constituent behavior to another one in response to a particular event. Similarly to constituent behaviors, conditions can be fine-tuned through a set of parameters and can be instantiated multiple times in the same probabilistic finite state machine.

The output of AutoMoDe is thus a combination of specific instances of modules where the parameters and the topology of the connections are optimized for the task at hand. AutoMoDe explores a search space that is represented by all the possible probabilistic finite state machines that can be obtained by instantiating and combining the given modules. Within AutoMoDe, the exploration of the search space can be performed using a wide range of optimization algorithms.

In AutoMoDe, the fact that the control software is obtained by selecting, assembling, and fine-tuning some given modules introduces a bias and reduces the representational power: the control software produced is *a priori* constrained to belong to the space of the finite

state machines that can be composed out of the given modules. This limits the possibility to fine-tune the dynamics of the robot–robot and robot–environment interaction. As we will show with the experimental results presented in Sect. 6, if the set of modules is appropriately defined, the bias that is introduced reduces the variance and increases the generalization capabilities of the obtained control software, without hampering its effectiveness.

The specialization of AutoMoDe

AutoMoDe is a general framework that needs to be specialized: (i) AutoMoDe has to be adapted to the given robotic platform, and (ii) the optimization process has to be defined. In the following, we will refer to the person that performs the specialization of AutoMoDe for a specific platform as the *expert*.

The expert specializes AutoMoDe for a specific platform on the basis of a *reference model*, an abstraction of the robotic platform that specifies in formal terms its characteristics and capabilities. The reference model defines the way in which we think of the robots and the way in which we intend the interaction of a robot with the environment and with other robots. In particular, the reference model defines an interface between the hardware layer and the logic layer represented by the control software. As an example, the reference model of a platform featuring an ambient light sensor could include the capability to distinguish between night and day. This capability could take the form of a Boolean variable that is updated by the hardware every, say, 100 ms and that can be read by the control software.

The reference model implicitly defines the class $\mathbb{T}$ of tasks that can be performed using a swarm composed of instances of the given robotic platform. For example, a task that requires separating blue objects from green objects cannot be performed by robots that are unable to distinguish green from blue. On the basis of the reference model, the expert must produce the set of modules that will be used by the specialization of AutoMoDe that is intended to produce control software for the corresponding platform. The set of modules implicitly defines the class $\mathbb{T}'$ of tasks that can be performed by a swarm of the given platform whose control software is obtained by assembling them.

In the ideal case, the set of modules perfectly and exhaustively exploits all the capabilities provided by the reference model and $\mathbb{T}' \equiv \mathbb{T}$. In practice, it has to be expected that the set of modules produced by the expert fails to suitably exploit some of the capabilities provided by the reference model, with the result that $\mathbb{T}'$ will be a proper subset of $\mathbb{T}$. In this process of specialization, the experience of the expert plays an important role. Indeed, the expert defines the constituent behaviors and the conditions by taking inspiration from those that have been previously presented in the literature and is guided by her personal understanding of what tasks are relevant in swarm robotics.

It has to be noticed that the specialization of AutoMoDe for a given reference model is task independent and has to be done only once: the same set of modules will be then used to design the control software for any task that one will subsequently wish to tackle with the given platform. It will be clearly unrealistic to expect that a specialization of AutoMoDe for a reference model is able to perform a task $t \notin \mathbb{T}$, where $\mathbb{T}$ is the class of tasks implicitly defined by the given reference model. On the other hand, given the current understanding of swarm robotics, whether any task $t \in \mathbb{T}$ can be performed via the set of modules produced by the expert—that is, whether $\mathbb{T}' \equiv \mathbb{T}$—is an empirical question.

Concerning the definition of the optimization process, a number of elements have to be selected including: the optimization algorithm to span the space of possible control software; a way to initialize the optimization algorithm; possible constraints on the finite state machine to be produced—e.g., the maximum number of states and of outgoing transitions for each

state; and a way to assess the performance of a candidate control software. We foresee that, to assess the performance of control software candidates, the optimization process will typically rely on computer-based simulations. The specialization of AutoMoDe for a specific robotic platform involves therefore the selection of an appropriate simulator of the robotic platform at hand.

## 4.1 Proof of concept: AutoMoDe-`Vanilla`

AutoMoDe-`Vanilla` is a proof-of-concept specialization of AutoMoDe. Our goal in this paper is not to define the ultimate automatic design method, but to show that the core ideas of AutoMoDe are valid. For this reason, AutoMoDe-`Vanilla` is unsophisticated in many respects such as the way in which probabilistic finite state machines are represented and optimized. We will explore more sophisticated instances of AutoMoDe in future research.

### *4.1.1 Robot platform and reference model*

AutoMoDe-`Vanilla` is specialized for a swarm of *e-puck* robots extended with the Overo Gumstick, the ground sensor, and the *range-and-bearing* board—see Figure 1 for a picture of the platform. The e-puck robot is a small wheeled robot designed for research and education (Mondada et al. 2009a,b). It is equipped with eight IR transceivers that can be used as light and proximity sensors. The IR transceivers are distributed around the body of the robot—for details on the position of the sensors, see Mondada et al. (2009b). The Overo Gumstick is a single-board computer that allows the e-puck to run Linux. The ground sensor comprises three IR transceivers positioned in the front of the robot and pointed downward to measure the reflectivity of the ground. The range-and-bearing board (Gutiérrez et al. 2009) comprises 12 IR emitters and 12 receivers equally distributed along the perimeter of the board and


**Fig. 1** The e-puck robot

**Table 1** Reference model—$prox_i$ is the reading of the $i$-th proximity sensor and $\angle q_i$ is the angle at which the $i$-th proximity sensor is positioned with respect to the head of the robot; $light_i$ is the reading of the $i$-th light sensor and $\angle q_i$ is the angle at which the $i$-th light sensor is positioned with respect to the head of the robot; $gnd_i$ is the reading of the $i$-th ground sensor; $n$ is the number of robots in the neighborhood; $r_m$ and $\angle b_m$ are respectively the range and bearing of the $m$-th neighbor; $v_l$ and $v_r$ are respectively the speed of the left and right wheel; and $\bar{v}$ is the maximum speed of the robot. Sensors and actuators are updated with a period of 100 ms

| Sensors/actuators | Variables |
|---|---|
| Proximity | $prox_i \in [0, 1]$, $\angle q_i$, with $i \in \{1, 2, \ldots, 8\}$ |
| Light | $light_i \in [0, 1]$, $\angle q_i$, with $i \in \{1, 2, \ldots, 8\}$ |
| Ground | $gnd_i \in \{0, 0.5, 1\}$, with $i \in \{1, 2, 3\}$ |
| Range and bearing | $n \in \mathbb{N}$ and $r_m$, $\angle b_m$, with $m \in \{1, 2, \ldots, n\}$ |
| Wheels | $v_l, v_r \in [-\bar{v}, \bar{v}]$, with $\bar{v} = 0.16\,\text{m/s}$ |

Period of the control cycle: 100 ms

pointed radially and outwards, on the horizontal plane. The range-and-bearing board allows the e-puck to reliably send and receive messages within a range of about 0.7 m. When an e-puck receives a message via the range-and-bearing board, it also obtains information about the relative position of the sender.

In this paper, the reference model that we adopt for the platform described above is given in Table 1. According to this reference model, the control software has a control cycle of 100 ms. At each control step, the control software makes decisions based on the variables $prox_i$, $light_i$, $gnd_i$, $n$, $r_m$, and $\angle b_m$, which abstract the proximity, light, ground sensors and range-and-bearing readings. Similarly, the control software can set the variables $v_l$ and $v_r$, which abstract the actuators that operate on the wheels. Specifically, $prox_i$ can assume values in the range $[0, 1]$ and it is equal to 0 when the $i$-th proximity sensor does not perceive obstacles within a 0.03 m range, while it is equal to 1 when the obstacle is closer than 0.01 m; $light_i$ can assume values in $[0, 1]$ and it is equal to 0 if the $i$-th light sensor perceives only the ambient light, while it is equal to 1 when the sensor saturates;[1] $gnd_i$ can assume only three values and it is equal to 0, 0.5 or 1 when the $i$-th ground sensor detects, respectively, a black, a gray, or a white floor; $n$ is the number of robots in the neighborhood, as perceived via the range-and-bearing board; $r_m$ and $\angle b_m$ are the range and bearing of each robot $m$ in the neighborhood. The values $v_l$ and $v_r$ define the speed of the wheels and are constrained in $[-\bar{v}, \bar{v}]$, with $\bar{v} = 0.16\,\text{m/s}$ being the maximum speed of the e-puck.

#### *4.1.2 Module set*

In AutoMoDe-`Vanilla`, the set of modules comprises six constituent behaviors and six conditions. Some of the modules have tunable parameters that are optimized by AutoMoDe-`Vanilla`. AutoMoDe-`Vanilla` selects, combines, and fine-tunes these modules to produce a finite state machine. This finite state machine operates with a period of 100 ms, which is the same period at which sensors and actuators are updated, as specified by the reference model given in Table 1. At each control cycle, the constituent behavior associated with the current state is performed. Subsequently, each outgoing transition of the current state is considered and the corresponding condition is evaluated to decide whether the transition is

[1] The reference model assumes that, besides the ambient light, at most one light source is present in the environment.

enabled or not. In case no transition is enabled, no state transition occurs. If at least a transition is enabled, one of them is randomly selected and the current state is updated accordingly.

In the following, we describe the modules of AutoMoDe-`Vanilla`. We adopt the convention that tunable parameters are denoted by letters of the Greek alphabet.

**Constituent behaviors**

*Exploration:* the robot moves straight. If any of the proximity sensors positioned in front senses an obstacle, that is, if $prox_i \geq 0.1$ for any $i \in \{1, 2, 7, 8\}$, the robot turns on itself for a random number of control cycles chosen in $\{0, 1, \ldots, \tau\}$, where $\tau$ is an integer parameter in $\{1, 2, \ldots, 100\}$. The robot turns away from the direction faced by the proximity sensor that returned the highest value.

*Stop:* the robot stays still.

*Phototaxis:* the robot moves towards the light source, if perceived; otherwise, it moves straight. Obstacle avoidance is embedded: the robot follows the vector $\mathbf{w} = \mathbf{w}' - k\mathbf{w}_o$, where $k$ is a hard-coded parameter whose value has been *a priori* fixed to 5 and $\mathbf{w}'$ and $\mathbf{w}_o$ are vectors defined as:

$$\begin{aligned} \mathbf{w}' &= \begin{cases} \mathbf{w}_l = \sum_{i=1}^{8} \left(light_i,\ \angle q_i\right), & \text{if light is perceived,} \\ (1,\ \angle 0), & \text{otherwise;} \end{cases} \\ \mathbf{w}_o &= \sum_{i=1}^{8} \left(prox_i,\ \angle q_i\right), \end{aligned} \tag{1}$$

where $\angle q_i$ is the angle at which sensor $i$ is positioned with respect to the head of the robot.

*Anti-phototaxis:* the robot moves away from the light source, if perceived; otherwise, it moves straight. Obstacle avoidance is embedded: the robot follows the vector $\mathbf{w} = \mathbf{w}' - k\mathbf{w}_o$, where

$$\mathbf{w}' = \begin{cases} -\mathbf{w}_l, & \text{if light is perceived,} \\ (1,\ \angle 0), & \text{otherwise;} \end{cases}$$

and $k$, $\mathbf{w}_l$, and $\mathbf{w}_o$ are defined in phototaxis.

*Attraction:* the robot uses the range-and-bearing board to go in the direction of the robots in neighborhood, if any; otherwise, it moves straight. Obstacle avoidance is embedded: the robot follows the vector $\mathbf{w} = \mathbf{w}' - k\mathbf{w}_o$, where

$$\mathbf{w}' = \begin{cases} \mathbf{w}_{r\&b} = \sum_{m=1}^{n} \left(\frac{\alpha}{r_m},\ \angle b_m\right), & \text{if robots are perceived,} \\ (1,\ \angle 0), & \text{otherwise;} \end{cases} \tag{2}$$

and $\alpha$ is a real-valued parameter in $[1, 5]$, and where $\mathbf{w}_o$ and $k$ are defined in phototaxis.

*Repulsion:* the robot moves away from the other robots in its neighborhood, if any; otherwise, it moves straight. Obstacle avoidance is embedded: the robot follows the vector $\mathbf{w} = \mathbf{w}' - k\mathbf{w}_o$, where

$$\mathbf{w}' = \begin{cases} -\mathbf{w}_{r\&b}, & \text{if robots are perceived,} \\ (1,\ \angle 0), & \text{otherwise;} \end{cases}$$

and $\mathbf{w}_{r\&b}$ is defined in attraction, while $\mathbf{w}_o$ and $k$ are defined in phototaxis.

**Conditions**

*Black-floor:* if $gnd_i = 0$, for any $i \in \{1, 2, 3\}$, the transition is enabled with probability $\beta$, where $\beta$ is a parameter.

*Gray-floor:* same as black-floor but the prerequisite is that $gnd_i = 0.5$, for any $i \in \{1, 2, 3\}$.

*White-floor:* same as black-floor but the prerequisite is that $gnd_i = 1$, for any $i \in \{1, 2, 3\}$.

*Neighbor-count:* the transition is enabled with probability:

$$z(n) = \frac{1}{1 + e^{\eta(\xi - n)}}, \tag{3}$$

where $n$ is the number of robots in the neighborhood, $\eta \in [0, 20]$ is a real-valued parameter and $\xi \in \{0, 1, \ldots, 10\}$ is an integer parameter. The transition is enabled with probability 0.5 if $n = \xi$. The parameter $\eta$ regulates the steepness of the function $z(n)$ at $n = \xi$.

*Inverted-neighbor-count:* the transition is enabled with probability $1 - z(n)$, where $z(n)$ is defined in Eq. 3.

*Fixed-probability:* the transition is enabled with probability $\beta$, where $\beta$ is a parameter.

#### *4.1.3 Optimization process*

Concerning the optimization algorithm, AutoMoDe-`Vanilla` adopts F-Race (Birattari et al. 2002; Birattari 2009), a racing algorithm originally developed for tuning metaheuristics. In particular, we use the implementation provided by the *irace* package (López-Ibáñez et al. 2011) for R (R Development Core Team 2008). F-Race has been designed to handle stochasticity in the evaluation of candidates: in the case of swarm robotics, the performance of a control software candidate is highly stochastic and the ability of F-Race to handle stochasticity appears to be appropriate in this context. Moreover, F-Race is an extremely simple algorithm and in the context of this paper we wish to keep the focus on the control architecture rather than on the optimization process.

Within the optimization process, control software candidates are evaluated via a computer-based simulation performed using ARGoS (Pinciroli et al. 2012), a multi-engine simulator of swarm robotics systems. In particular, we use ARGoS' 2D dynamic physics engine to model the robots and the environment.

F-Race iteratively evaluates a set of control software candidates, all generated randomly at the beginning of the optimization process, and discards the candidates that have a low expected performance, until a stopping criterion is met.

At iteration $i$, the candidates that have not been discarded in the previous $i-1$ iterations are evaluated on a test case. A test case is characterized by the specific initial condition—e.g., the initial position of the robots in the arena. To evaluate a control software candidate on a test case, F-Race performs a simulation run. After all candidates are evaluated, F-Race discards those candidates whose expected performance, as estimated on the $i$ test cases considered so far, is statistically dominated by at least another candidate. The surviving candidates enter iteration $i+1$. The process stops either when a single candidate remains or when a predefined maximum number of evaluations has been performed. The maximum number of evaluations is the available *design budget* and is part of the specifications of an automatic design problem. It measures the computational resources available to produce the desired design.

In order to limit the complexity of the control software produced and therefore its representational power, we limit the number of states and conditions included: AutoMoDe-`Vanilla` can generate probabilistic finite state machines with up to four states, where each state can

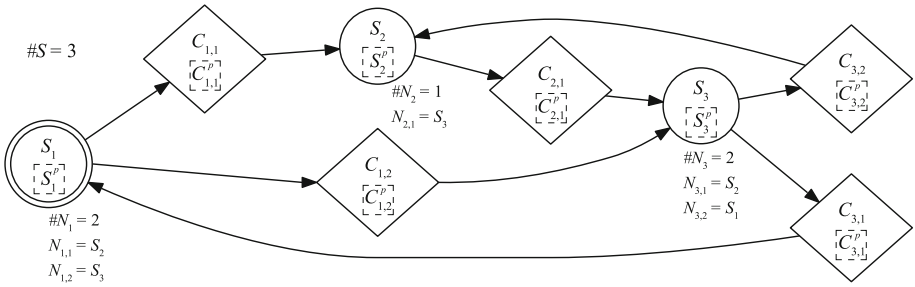


**Fig. 2** Example of a finite state machine generated by AutoMoDe-`Vanilla`. The notation adopted is the one defined in Eq. 4

have up to four outgoing transitions. The candidates are generated at the beginning of the optimization process using the built-in sampling procedure provided by the *irace* package. This procedure samples the space defined as:

$$\langle \#S,\ S_i,\ S_i^p,\ \#N_i,\ N_{i,j_i},\ C_{i,j_i},\ C_{i,j_i}^p \rangle \begin{array}{l} i = 1, 2 \ldots, \#S \\ j_i = 1, 2, \ldots, \#N_i \end{array} \tag{4}$$

where $\#S \in \{1, 2 \ldots, 4\}$ is the number of states of the probabilistic finite state machine; $S_i \in \{1, 2 \ldots, 6\}$ is the constituent behavior of state $i$;[2] $S_i^p$ are the parameters of the constituent behavior $S_i$, if any; $\#N_i \in \{1, 2, \ldots, 4\}$ is the outdegree of state $i$, that is, the number of transitions outgoing state $i$; $j_i$ is an index spanning the $\#N_i$ successors of state $i$; $N_{i,j_i} \in \{1, 2, \ldots, \#S\}$ is the $j_i$-th successor of state $i$; $C_{i,j_i} \in \{1, 2, \ldots, 6\}$ is the condition associated to the transition that connects state $i$ to its $j_i$-th successor $N_{i,j_i}$; and $C_{i,j_i}^p$ are the parameters of the condition $C_{i,j_i}$, if any. Figure 2 provide an illustrative example of a finite state machine sampled from the space defined in Eq. 4.

The cardinality of the initial set of candidates is one sixth of the available design budget.

AutoMoDe-`Vanilla` implements the best practice commonly followed in automatic design to obtain control software that has the highest chance to overcome the reality gap: A simulated uniform noise of 5 % is added on the proximity, ground and light sensors and on the wheels actuator (Jakobi et al. 1995). The noise of the range-and-bearing board follows a model defined using empirical data. Moreover, the initial position and orientation of the robots at each iteration of the F-Race algorithm are randomly set by sampling from a uniform distribution.

## 5 Experimental setup

To assess the capabilities of AutoMoDe-`Vanilla`, we conduct a series of experiments in which AutoMoDe-`Vanilla` is used to automatically design the control software for robot swarms that are intended to perform two different tasks: *aggregation* and *foraging*. We selected these two tasks because they are common benchmarks in swarm robotics and it appears that they can be performed by a swarm of robots characterized by the reference model presented in Sect. 4.1.

The experiments presented in this paper adhere to a hands-off experimental protocol: we do not allow any human intervention in the automatic design process. The aim of the

[2] State $i = 1$ is the initial state of the probabilistic finite state machine.

experiments is to assess the expected performance of AutoMoDe-Vanilla in designing control software for a robot swarm.[3] We run three sets of experiments that differ in the design budget, that is, the total number of simulation runs that AutoMoDe-Vanilla can use to design the control software. The three design budgets are: 10000, 50000, and 200000 simulation runs. For each design budget, we execute 20 independent runs of AutoMoDe-Vanilla and we therefore obtain 20 instances of control software; we then assess the performance of these instances on the robots by performing a single run of each of them.[4]

The experimental protocol we adopt provides for a number of elements that reduce the intervention of the human experimenter during the evaluation of the control software produced by AutoMoDe-Vanilla: the control software obtained in simulation is automatically cross-compiled by ARGoS and copied on the e-pucks without any modification. The initial position and orientation of the robots is obtained by running the constituent behavior exploration—see Sect. 4.1—for a random number of seconds in $\{1, 2, \ldots, 20\}$. The performance of the robots is automatically computed by a tracking system (Stranieri et al. 2013) on the basis of data gathered via a ceiling camera.

With the aim of quantifying the effects of the reality gap, we perform a further independent assessment in simulation of the instances of control software produced by AutoMoDe-Vanilla. Also in simulation, each instance is assessed by performing a single run.

### 5.1 A yardstick: EvoStick

To put AutoMoDe-Vanilla into perspective, we define another automatic design method and we perform a comparison. This second method, which we call EvoStick, is based on evolutionary robotics and implements the current best practice in the automatic design of robot swarms. EvoStick is the same method that we have already successfully used in previous experiments (Francesca et al. 2012).

EvoStick is based on the same reference model that we have defined in Table 1 and that is adopted by AutoMoDe-Vanilla. Each robot is controlled by a fully connected, feed-forward neural network whose control cycle has a period of 100 ms, as specified by the reference model. The neural network has 24 inputs, 2 outputs and no hidden units. The inputs are based on the capabilities defined by the reference model: 8 proximity sensors $prox_i \in [0, 1]$, $i \in \{1, 2, \ldots, 8\}$; 8 light sensors $light_i \in [0, 1]$, $i \in \{1, 2, \ldots, 8\}$; 3 ground sensors $gnd_i \in \{0, 0.5, 1\}$, $i \in \{1, 2, 3\}$; and 5 aggregated inputs from the range-and-bearing board. The aggregated inputs from the range-and-bearing board are:

[3] Because AutoMoDe-Vanilla is stochastic, reporting and discussing its expected performance appears to be the appropriate choice (Birattari and Dorigo 2007).

[4] The reader might wonder why, in order to estimate the expected performance of AutoMoDe-Vanilla on each design budget, we repeat the design process 20 times and we test the resulting design on the robots only once. Due to time constraints, we have decided to run 20 robot experiments per design budget. Having fixed to 20 the total number of robot experiments, one might be tempted to consider alternative protocols: repeat the design 20 times and evaluate each resulting design 1 time; repeat the design 10 times and evaluate each resulting design 2 times; repeat the design 5 times and evaluate each resulting design 4 times; or even performing the design once and evaluate the result 20 times. Although all these protocols would produce an unbiased estimate of the expected performance of AutoMoDe-Vanilla, the one implemented in this study is the one that minimizes the variance of the estimate. A similar issue has been formally studied in the context of the assessment of stochastic optimization algorithms (Birattari 2004, 2009).

**Table 2** `EvoStick`—partition of the available design budget

| Budget | Evolutionary algorithm | (Iterations) | Post-evaluation | (Per individual) |
|---|---|---|---|---|
| 10,000 | 8,000 | (8) | 2,000 | (20) |
| 50,000 | 40,000 | (40) | 10,000 | (100) |
| 200,000 | 150,000 | (150) | 50,000 | (500) |

$\tilde{z}(n) = 1 - 2/(1 + e^n)$, where $n$ is the number of the robots perceived; and the scalar projections of $\mathbf{w}_{r\&b} = \sum_{m=1}^{n} (1/r_m, \angle b_m)$ on the four unit vectors that point at 45°, 135°, 225°, and 315° with respect to the head of the robot. Please notice that $\tilde{z}(n)$ is defined as the function $z(n)$ adopted in AutoMoDe-`Vanilla` (Eq. 3), with $\eta = 1$, $\xi = 0$, and scaled and shifted to be constrained in $[-1, 1]$; $\mathbf{w}_{r\&b}$ is defined as the function of the same name adopted in AutoMoDe-`Vanilla` (Eq. 2), with $\alpha = 1$. The activation of the output neurons is computed as the weighted sum of all input units plus a bias term, filtered through a standard logistic function. The outputs of the neural network are scaled in $[-v_m, v_m]$, with $v_m = 0.16$ m/s, as specified by the reference model, and are used to set the speed of the two wheels.

The neural network is characterized by a set of 50 parameters. Each parameter is a real value in $[-5, 5]$. These parameters are optimized via a standard evolutionary algorithm. The cardinality of the population is 100. The initial population is randomly generated. At each iteration, each individual in the population is evaluated through 10 simulations performed using ARGoS' 2D dynamic physics engine. The following population is generated via elitism and mutation. The elite composed of the 20 best individuals is included unchanged. The rest of the population is obtained from the elite via mutation: parameters are modified by adding a random value drawn from a normal distribution with mean 0 and variance 1. The evolutionary algorithm stops after a predefined number of iterations. The final population is re-evaluated a number of times to obtain a better performance estimation, and the individual with the highest mean performance is selected. The available design budget is partitioned in two parts: one for the evolutionary algorithm and one for the post-evaluation. See Table 2 for the details.

`EvoStick` implements exactly the same precautions adopted in AutoMoDe-`Vanilla` to reduce the risk of obtaining control software that does not overcome the reality gap—see Sect. 4.1.3. In the definition of `EvoStick`, we do not employ any further technique to overcome the reality gap. The adoption of other techniques is not *a priori* justified, and would possibly become apparent only *a posteriori*, when looking at the results obtained on each specific task to be performed. If additional techniques to overcome the reality gap are adopted on a per-task basis and *a posteriori* on the base of the results obtained, the overall design process would end up into a human-driven trial-and-error search and this would defeat the purpose of our experimental protocol and of our research as a whole.

We assess the performance of `EvoStick` with the same criteria and under the same experimental conditions that we adopt for the assessment of AutoMoDe-`Vanilla`. In particular, the two methods use the same simulator, design control software for the same robotic platform under the same reference model, and optimize the same objective function under the same environmental conditions. Concerning the robot experiments, to limit spurious effects of battery level and of other possible unforeseen ambiental contingencies, the order of the experiments is randomly generated and runs of instances of control software produced by AutoMoDe-`Vanilla` and by `EvoStick` are interleaved.

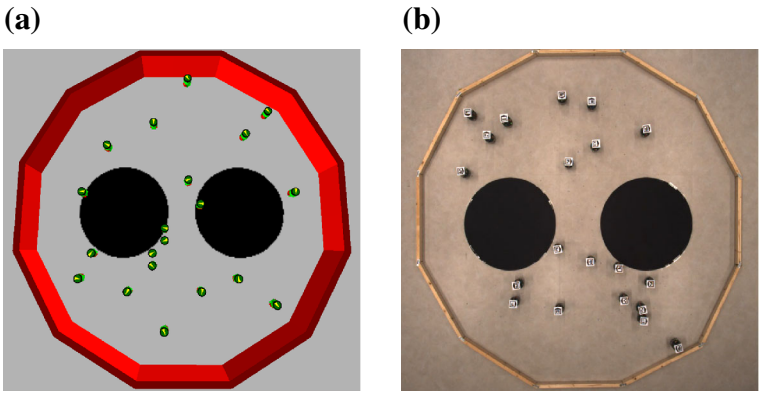


**Fig. 3** Arena for the aggregation task. **a** Simulated arena and 20 e-pucks; **b** Real arena and 20 e-pucks

## 5.2 Tasks

In all the experiments that we perform on the aggregation and the foraging tasks, the swarm comprises 20 e-puck robots. The available time to carry out the task is 250 s. The robots operate in a dodecagonal arena of 4.91 $m^2$, surrounded by walls. For future reference, we define a coordinate system with origin in the center of the arena and $x$ axis parallel to one of the sides. Coordinates in this system are given in meters.

### *5.2.1 Aggregation*

In the aggregation task, the swarm has to cluster in one of the two black areas of the arena's floor. The aggregation task is the same analyzed in (Francesca et al. 2012). Figure 3 shows the arena for the aggregation task in both simulation and reality. The floor of the arena is gray and there are two black circular areas on the floor, namely $a$ and $b$. The areas have the same radius of 0.35 m and are centered in $(0.6, 0)$ and $(-0.6, 0)$.

At the beginning of each run, the 20 e-puck robots are randomly distributed in the arena. The objective function is $F_{aggregation} = \max(N_a, N_b)/N$, where $N_a$ and $N_b$ are the number of robots that are in the black areas $a$ or $b$ at the end of the simulation, and $N = 20$ is the size of the swarm. This objective function equals 1 if all the robots are aggregated in one of the two areas.

### *5.2.2 Foraging*

In the foraging task, the swarm has to retrieve as many objects as possible from two sources and deposit them in the nest. Because the e-puck platform has no grasping capabilities, we abstract the actions of retrieving and depositing objects: We reckon that an e-puck has retrieved an object when it enters a source, which is represented by a black circle on the ground. Similarly, we reckon that an e-puck has deposited the object it is carrying when it enters the nest, which is represented by a white area. Our foraging task is inspired by the one presented in (Liu et al. 2007). Figure 4 shows the arena for the foraging task in both simulation and reality. The two black areas have a radius of 0.15 m and are centered in $(0.75, 0)$ and $(-0.75, 0)$. Moreover, a light source is positioned behind the nest area, in $(0, 1.25)$ at 0.75 m

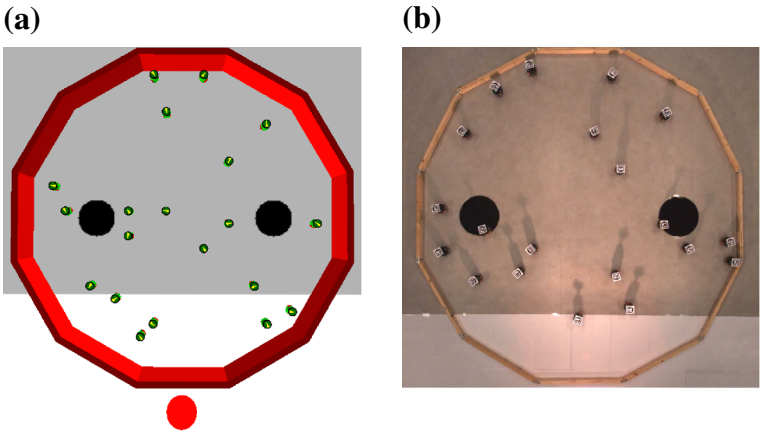


**Fig. 4** Arena for the foraging task. The *red circle* at the bottom of the simulated arena represents the light source. **a** Simulated arena and 20 e-pucks; **b** Real arena and 20 e-pucks

from the ground. The objective function is $F_{\text{foraging}} = N_{\text{o}}$, where $N_{\text{o}}$ is the total number of objects retrieved and deposited.

## 6 Results

We analyze the results of the experiments from two points of view: first, we estimate the performance of the control software produced by AutoMoDe-`Vanilla`, using the performance of the one produced by `EvoStick` as a yardstick; second, we compare the performance of the control software produced by AutoMoDe-`Vanilla` in simulation and on the robots to evaluate the impact of the reality gap. Also in this case, we use `EvoStick` as a yardstick.

Moreover, for each task we provide a behavioral analysis of the swarms designed by AutoMoDe-`Vanilla` and by `EvoStick`. In this analysis, we also highlight the main differences between the behaviors observed in simulation and those observed in reality.

The complete set of experimental data and video recordings of all the robot experiments is available online (Francesca et al. 2013).

### 6.1 Aggregation

Figure 5 shows the performance of AutoMoDe-`Vanilla` and of `EvoStick` in simulation and on the robots.

In all three sets of experiments, AutoMoDe-`Vanilla` designs robot swarms that perform better than those designed by `EvoStick`: for each design budget, the difference in performance between AutoMoDe-`Vanilla` and `EvoStick` is statistically significant according to the Wilcoxon test, with 95 % confidence.

A visual inspection of the plots shows that AutoMoDe-`Vanilla` and `EvoStick` have similar performance in simulation. For what concerns the reality gap, in the case of `EvoStick` there is a large difference in performance between simulation and reality. The control software obtained by `EvoStick`, even though it yields good results in simulation, is not able to reliably produce aggregation on the robots. On the contrary, in the case of AutoMoDe-`Vanilla` the difference between simulation and reality is small. A statistical

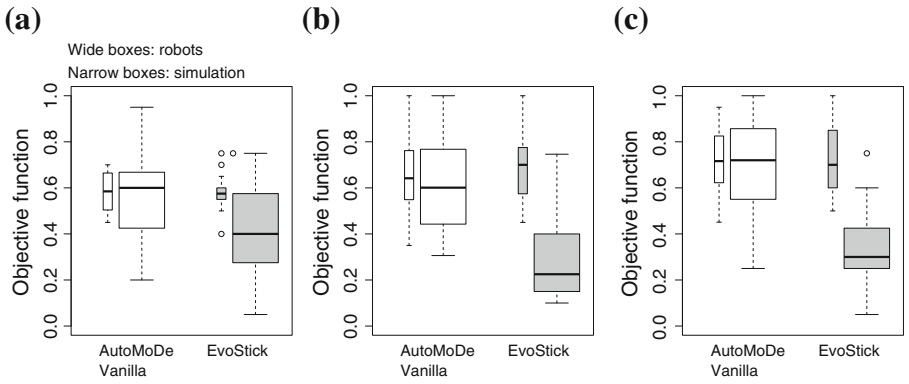


**Fig. 5** Aggregation—performance of the control software obtained using different design budgets. The plot shows, for AutoMoDe-`Vanilla` and `EvoStick`, the performance of the 20 instances of the control software (one for each independent run) both in simulation (*narrow boxes*) and on the robots (*wide boxes*). A box comprises observations between the first and third quartile; the *black horizontal line* represents the median of the observations; the top whisker extends either to the largest observation or to 3/2 of the upper quartile (whatever is smaller); the bottom whisker is defined similarly; observations falling outside the extension of whiskers (if any) are outliers and are represented as *circles*. **a** 10,000 design budget; **b** 50,000 design budget; **c** 200,000 design budget

**Table 3** Aggregation—estimated mismatch between simulation and reality, and corresponding confidence intervals according to the Wilcoxon test

| Budget | | Estimated mismatch | 95 % confidence interval | |
|---|---|---|---|---|
| 10000 | AutoMoDe-`Vanilla` | 0.01 | −0.06 | 0.08 |
| | `EvoStick` | 0.19 | 0.07 | 0.30 |
| 50000 | AutoMoDe-`Vanilla` | 0.03 | −0.12 | 0.17 |
| | `EvoStick` | 0.40 | 0.30 | 0.50 |
| 200000 | AutoMoDe-`Vanilla` | 0.01 | −0.11 | 0.12 |
| | `EvoStick` | 0.40 | 0.30 | 0.50 |

analysis based on the Wilcoxon test is reported in Table 3. The data confirms that the mismatch between simulation and reality is lower in AutoMoDe-`Vanilla` than in `EvoStick`. The difference between the mismatch observed for AutoMoDe-`Vanilla` and for `EvoStick` is significant with a confidence level of at least 95 %. The table indicates that the performance difference between simulation and reality observed in `EvoStick` increases with the design budget from 10000 to 50000, and then saturates. These observations could be explained as a result of overfitting: the larger the design budget, the longer the fine-tuning of the control software, and consequently the larger the risk of overfitting, up to saturation.

In all 60 runs, across the three budget levels, AutoMoDe-`Vanilla` has used the whole available design budget. As a result, AutoMoDe-`Vanilla` and `EvoStick` have always run the same number of simulated experiments.

### *6.1.1 Behavioral analysis*

In this section, we describe the behavior of the control software designed by AutoMoDe-`Vanilla` and `EvoStick` for aggregation.

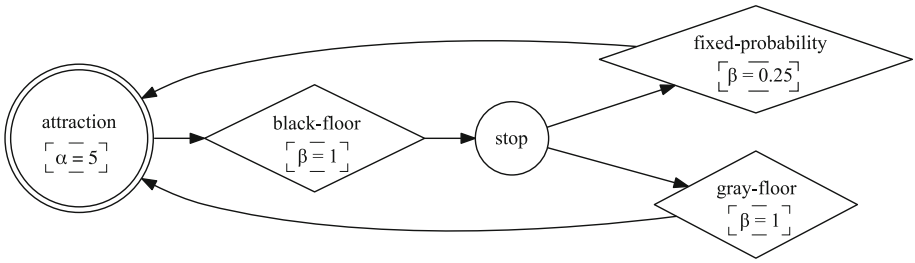


**Fig. 6** Aggregation—an instance of control software designed by AutoMoDe-Vanilla. The initial state is represented by the double-line circle. The robot performs attraction and moves toward the other robots. When it detects the black floor, it stops. In the stop state it checks for its transitions. It changes state when it detects the gray floor. It also starts moving, with a 0.25 probability, independently from the floor color

AutoMoDe-Vanilla. A feature of AutoMoDe is that the obtained control software is a probabilistic state machine, which is human readable. The 60 instances of the control software designed by AutoMoDe-Vanilla for the aggregation task have, with minor differences, the same structure. Figure 6 shows a representative instance. Each robot starts in the attraction state, that is, it moves toward other robots. With probability 1, a robot changes state to stop when it senses that the floor is black. In the stop state, the robot does not move. The robot then changes the state to attraction with a 0.25 probability or when it perceives the gray floor. This last event can happen because the robot is pushed outside the black area by other robots. The resulting collective behavior can be described as follows: Initially, robots tend to move in the direction of their neighbors and tend to cluster. Robots that enter a black area stop for some time and act as an attraction point for their neighbors. After a while, all robots are either in a black area or in its proximity. Eventually, most of the robots are attracted inside the black area where the majority of the robots are located. The behaviors observed in simulation and in reality are similar.

EvoStick. Because neural network are not human readable, the only way to analyze the control software obtained by EvoStick is to instantiate it on the robots and observe the resulting behavior. The 60 instances of control software obtained by EvoStick show behaviors that are qualitatively similar to one another. When a robot is in the gray area, it moves following a circular trajectory. The radius of this trajectory decreases when the number of robots perceived increases. This movement allows the robots to create aggregates. When a robot enters a black area, the radius of its trajectory becomes very small, to the point that the robot almost rotates on the spot. In this condition, the robot leaves the black area only because pushed out by other robots. The robots that are in the black areas attract other robots. From our observations, it appears that the quality of the resulting collective behavior strongly depends on where the first aggregates are created: if these aggregates are far from the black areas, the robots are not able to find the black areas. In simulation, the behavior is qualitatively similar but the circular trajectories have a larger radius with respect to the ones observed in reality.

## 6.2 Foraging

Figure 7 shows the performance achieved on the foraging task by AutoMoDe-Vanilla and by EvoStick, both in simulation and on the robots. The obtained results show the same trend observed in the aggregation task.

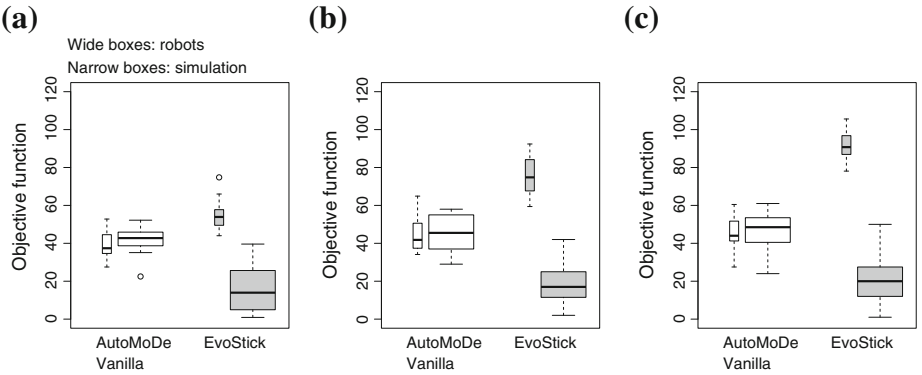


**Fig. 7** Foraging—performance of the control software obtained using different design budgets. The plot shows, for AutoMoDe-`Vanilla` and `EvoStick`, the performance of the 20 instances of the control software (one for each independent run) both in simulation (*narrow boxes*) and on the robots (*wide boxes*). See the caption of Fig. 5 for an explanation of the graphical conventions adopted in the plot. **a** 10,000 design budget; **b** 50,000 design budget; **c** 200,000 design budget

**Table 4** Foraging—estimated mismatch between simulation and reality, and corresponding confidence intervals according to the Wilcoxon test

| budget | | estimated mismatch | 95 % confidence interval | |
|---|---|---|---|---|
| 10000 | AutoMoDe-`Vanilla` | −3 | −7 | 2 |
| | `EvoStick` | 39 | 31 | 47 |
| 50000 | AutoMoDe-`Vanilla` | −2 | −6 | 3 |
| | `EvoStick` | 57 | 51 | 64 |
| 200000 | AutoMoDe-`Vanilla` | −1 | −6 | 4 |
| | `EvoStick` | 70 | 64 | 79 |

In all three experiments, AutoMoDe-`Vanilla` designs robot swarms that perform better than those designed by `EvoStick`. Differences are all significant according to the Wilcoxon test, with 95 % confidence.

Concerning the reality gap, it is interesting to note that `EvoStick` shows signs of overfitting: the mismatch between simulation and reality is large and increases with the design budget. On the contrary, AutoMoDe-`Vanilla` is able to design control software that is robust to the reality gap. The statistical analysis reported in Table 4 confirms these observations: In all the experiments, the difference between the mismatch observed for AutoMoDe-`Vanilla` and for `EvoStick` is significant with a confidence level of at least 95 %. In the case of AutoMoDe-`Vanilla`, the simulation slightly underestimates the performance of the robots. This is shown by the fact that the expected difference is slightly negative. On the contrary, in the case of `EvoStick` the simulation greatly overestimates the performance of the robots.

In all 60 runs, across the three budget levels, AutoMoDe-`Vanilla` has used the whole available design budget. As a result, AutoMoDe-`Vanilla` and `EvoStick` have always run the same number of simulated experiments.

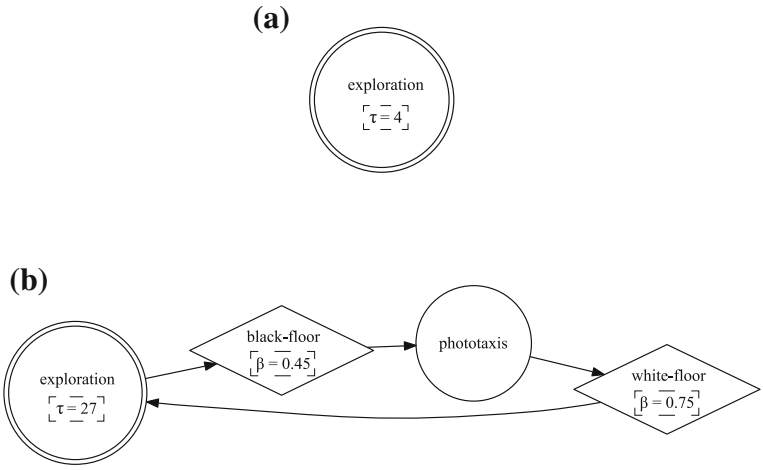


**Fig. 8** Foraging—the two classes of control software designed by AutoMoDe-Vanilla. The initial state is indicated by a *double-line circle*. **a** First class: an example; **b** Second class: an example

### *6.2.1 Behavioral analysis*

In this section, we describe the behavior of the control software designed by AutoMoDe-Vanilla and EvoStick for foraging.

AutoMoDe-Vanilla. The 60 instances of the control software obtained by AutoMoDe-Vanilla can be grouped into two classes. The instances in the first class feature only exploration. See Fig. 8a for an example. By performing exploration, the robots periodically enter the black and the white areas incrementing the value of the objective function. Instances of the first class of control software are frequent when the lowest design budget is used (14 instances out of 20), while they are rare for the other design budgets (3 and 0 instances in the case of design budget 50000s and 200000s, respectively). The instances of the second class feature an alternation between exploration and phototaxis. See Fig. 8b for an example. A robot uses exploration to search for the black areas. When it finds a black area, it switches to phototaxis to return to the white area. When it reaches the white area, it resumes exploration. The behaviors observed in simulation and in reality are similar.

EvoStick. The 60 instances of the control software obtained by EvoStick show qualitatively similar behaviors. Robots explore the arena following curved trajectories that are perturbed by the presence of other robots, the color of the floor and the intensity of the light. As a result of the perturbations, robots follow the walls and sometimes cross the arena passing on the black areas (the sources) and on the white area (the nest). However, this behavior is strongly affected by interference among robots: frequently, robots create aggregates that dissolve after a while. Concerning the comparison between the simulation and reality, two are the main differences. (i) robots interfere less with each other in simulation than in reality; and (ii) circular trajectories have a larger radius in simulation than in reality.

## 7 Discussion

In the experiments reported in Sect. 6, AutoMoDe-Vanilla scored better than EvoStick at overcoming the reality gap. It should be noted that the two methods have been tested

under the same conditions: same simulator, environment, objective function, design budget, robot platform, and reference model. The reality gap that AutoMoDe-Vanilla and EvoStick had to overcome is therefore the same. Nonetheless, the mismatch between simulation and reality that we observed is significantly higher for the control software generated by EvoStick than for the one generated by AutoMoDe-Vanilla. Following the conjecture we presented in Sect. 3, we ascribe this difference to the different representational power of the control architecture adopted by AutoMoDe-Vanilla and EvoStick. It is true that, besides the control architecture, AutoMoDe-Vanilla and EvoStick also differ in the optimization algorithm adopted. Nonetheless, it is our contention that the impact of the optimization algorithm is negligible in this context and that the control architecture is the main responsible for the differences observed in our experiments. To backup this contention, we performed some exploratory experiments (Francesca et al. 2013) in which we compare EvoStick with another method, FnnStick, in which the control architecture is the same neural network adopted in EvoStick and the optimization process is the one adopted in AutoMoDe-Vanilla. The results of these exploratory experiments show that FnnStick performs slightly worse than EvoStick, which excludes that the differences between the performance of AutoMoDe-Vanilla and EvoStick can be explained by a superior performance of the optimization process adopted in AutoMoDe-Vanilla. We can therefore conclude that our experiments corroborate the conjecture presented in Sect. 3: the high representational power provided by the fine-grained control architecture adopted in EvoStick is not properly exploited and results in solutions that do not properly generalize to the real world. On the contrary, AutoMoDe-Vanilla, with its relatively low representational power, displays better generalization capabilities. In terms of the bias–variance tradeoff (Geman et al. 1992), AutoMoDe-Vanilla has a higher bias towards a relatively restricted class of behaviors—specifically, those that can be obtained by assembling a four-state probabilistic finite state machine starting from the six given constituent behaviors and the six given conditions. As a result, AutoMoDe-Vanilla expectedly features a lower variance compared to EvoStick. Eventually, this results in a superior ability to overcome the reality gap.

## 8 Conclusions and future work

Guided by the concept of *bias–variance tradeoff*, we introduced AutoMoDe: a new approach to the automatic design of control software for robot swarms. AutoMoDe designs control software in the form of probabilistic finite state machines by drawing from a given set of preexisting parametric modules. AutoMoDe automatically selects, combines, and instantiates these modules using an optimization algorithm that aims at maximizing a task-specific measure of performance. By injecting an appropriate *bias* in the form of predefined modules, we can reduce the *variance* of the automatic design process. As a result, the control software produced by AutoMoDe is able to overcome the reality gap.

We presented also AutoMoDe-Vanilla, which is a proof-of-concept instance of AutoMoDe specialized for a specific reference model of the e-puck robot. We carried out an experimental analysis in which we used AutoMoDe-Vanilla to design control software for two different swarm robotics tasks: aggregation and foraging. The experiments were carried out using a hands-off experimental protocol, that is, no human intervention has been allowed in the automatic design process. The results show that AutoMoDe-Vanilla is able to successfully design control software for both tasks. The control software obtained

by AutoMoDe-Vanilla overcomes the reality gap: the performance in simulation and in reality is comparable.

Future work will focus on the empirical characterization of the class of tasks for which AutoMoDe-Vanilla can successfully design control software. We will also focus on the development of other instances of AutoMoDe. We plan on improving over AutoMoDe-Vanilla by exploring the use of different optimization algorithms and more sophisticated ways to encode probabilistic finite state machines.

**Acknowledgments** The research leading to the results presented in this paper has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no. 246939. G. Francesca acknowledges support by the META-X project, an *Action de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium. M. Brambilla, A. Brutschy, and M. Birattari acknowledge support from the Belgian F.R.S.-FNRS. Vito Trianni acknowledges support by the Italian National Research Council (CNR) within the EUROCORES Programme EuroBioSAS of the European Science Foundation. The authors thank the anonymous reviewers for their useful comments.

## References

Baldassarre, G., Trianni, V., Bonani, M., Mondada, F., Dorigo, M., & Nolfi, S. (2007). Self-organised coordinated motion in groups of physically connected robots. *IEEE Transactions on Systems, Man and Cybernetics-Part B*, *37*(1), 224–239.

Berman, S., Kumar V., & Nagpal. R. (2011). Design of control policies for spatially inhomogeneous robot swarms with application to commercial pollination. In *2011 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 378–385). Piscataway, NJ: IEEE Press.

Birattari, M. (2004). On the estimation of the expected performance of a metaheuristic on a class of instances. How many instances, how many runs? Technical report TR/IRIDIA/2004-001, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium.

Birattari, M. (2009). *Tuning metaheuristics: A machine learning perspective*. Berlin, Germany: Springer.

Birattari, M., & Dorigo, M. (2007). How to assess and report the performance of a stochastic algorithm on a benchmark problem: Mean or best result on a number of runs? *Optimization Letters*, *1*(3), 309–311.

Birattari, M., Stützle, T., Paquete, L., & Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002)* (pp. 11–18). San Francisco: Morgan Kaufmann

Bishop, C. (1995). Training with noise is equivalent to Tikhonov regularization. *Neural Computation*, *7*(1), 108–116.

Bongard, J., Zykov, V., & Lipson, H. (2006). Resilient machines through continuous self-modeling. *Science*, *314*(5802), 1118–1121.

Bongard, J. C. (2013). Evolutionary robotics. *Communications of the ACM*, *56*(8), 74–83.

Brambilla, M., Pinciroli, C., Birattari, M., & Dorigo. M. (2012). Property-driven design for swarm robotics. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)* (pp. 139–146). Richland, SC: IFAAMAS.

Brambilla, M., Ferrante, E., Birattari, M., & Dorigo, M. (2013). Swarm robotics: A review from the swarm engineering perspective. *Swarm Intelligence*, *7*(1), 1–41.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, *2*(1), 14–23.

Şahin, E. (2005). Swarm robotics: From sources of inspiration to domains of application. In *Proceedings of the 2004 International Conference on Swarm Robotics, volume 3342 of LNCS,* (pp. 10–20). Berlin, Germany: Springer.

Dietterich, T., & Kong, E. B. (1995). *Machine learning bias, statistical bias, and statistical variance of decision tree algorithms*. Technical report: Department of Computer Science, Oregon State University.

Dorigo, M., Birattari, M., & Brambilla, M. (2014). Swarm robotics. *Scholarpedia*, *9*(1), 1463.

Duarte, M., Oliveira, S., & Christensen, A. L. (2012a). Automatic synthesis of controllers for real robots based on preprogrammed behaviors. In *From animals to animats 12, volume 7426 of LNCS* (pp. 249–258). Berlin, Germany: Springer.

Duarte, M., Oliveira, S., & Christensen, A. L. (2012b). Hierarchical evolution of robotic controllers for complex tasks. In *2012 IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL)* (pp. 1–6). Piscataway, NJ: IEEE Press.

Ferrante, E., Guzmán, E. D., Turgut, A. E., & Wenseleers, T. (2013). GESwarm: Grammatical evolution for the automatic synthesis of collective behaviors in swarm robotics. In *Proceedings of the Fifteenth International Conference on Genetic and Evolutionary Computation Conference Companion* (pp. 17–24). New York: ACM.

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

Floreano, D., Husbands, P., & Nolfi, S. (2008). Evolutionary robotics. *Springer handbook of robotics* (pp. 1423–1451). Berlin, Germany: Springer.

Francesca, G., Brambilla, M., Trianni, V., Dorigo, M., & Birattari. M. (2012). Analysing an evolved robotic behaviour using a biological model of collegial decision making. In *From animals to animats 12, volume 7426 of LNCS* (pp. 381–390) Berlin, Germany: Springer.

Francesca, G., Brambilla, M., Brutschy, A., Trianni, V., & Birattari, M. (2013). AutoMoDe: A novel approach to the automatic design of control software for robot swarms. Supplementary information page at http://iridia.ulb.ac.be/supp/IridiaSupp2013-007/

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, *4*(1), 1–58.

Gutiérrez, Á., Campo, A., Dorigo, M., Donate, J., Monasterio-Huelin, F., & Magdalena, L. (2009). Open E-puck range & bearing miniaturized board for local communication in swarm robotics. In 2009 IEEE International Conference on Robotics and Automation (ICRA) (pp. 3111–3116). Piscataway, NJ: IEEE Press.

Hamann, H., & Wörn, H. (2008). A framework of space-time continuous models for algorithm design in swarm robotics. *Swarm Intelligence*, *2*(2), 209–239.

Harvey, I., Husbands, P., Cliff, D., Thompson, A., & Jakobi, N. (1997). Evolutionary robotics: The Sussex approach. *Robotics and Autonomous Systems*, *20*(2), 205–224.

Hauert, S., Zufferey, J.-C., & Floreano, D. (2008). Evolved swarming without positioning information: An application in aerial communication relay. *Autonomous Robots*, *26*(1), 21–32.

Jakobi, N. (1997). Evolutionary robotics and the radical envelope-of-noise hypothesis. *Adaptive Behavior*, *6*(2), 325–368.

Jakobi, N., Husbands, P., & Harvey, I. (1995). Noise and the reality gap: The use of simulation in evolutionary robotics. In *Advances in Artificial Life (ECAL'95), volume 929 of LNCS* (pp. 704–720) Berlin, Germany: Springer.

Kazadi, S., Lee, J. R., & Lee, J. (2009). Model independence in swarm robotics. *International Journal of Intelligent Computing and Cybernetics*, *2*(4), 672–694.

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

Liu, W., Winfield, A., Sa, J., Chen, J., & Dou, L. (2007). Strategies for energy optimisation in a swarm of foraging robots. *Swarm Robotics, volume 4433 of LNCS* (pp. 14–26). Berlin, Germany: Springer.

López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., & Birattari, M. (2011). The irace package, iterated race for automatic algorithm configuration. Technical report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium.

Maes, P. (1991). The agent network architecture (ANA). *ACM SIGART Bulletin*, *2*(4), 115–120.

Marocco, D., & Nolfi, S. (2007). Emergence of communication in embodied agents evolved for the ability to solve a collective navigation problem. *Connection Science*, *19*(1), 53–74.

Matarić, M. (1997a). Learning social behavior. *Robotics and Autonomous Systems*, *20*(2–4), 191–204.

Matarić, M. (1997b). Reinforcement learning in the multi-robot domain. *Autonomous Robots*, *4*(1), 73–83.

Miglino, O., Lund, H. H., & Nolfi, S. (1995). Evolving mobile robots in simulated and real environments. *Artificial Life*, *2*(4), 417–434.

Mitri, S., Floreano, D., & Keller, L. (2011). Relatedness influences signal reliability in evolving robots. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, *278*(1704), 378–383.

Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J.-C., Floreano, D., & Martinoli, A. (2009a). The e-puck, a robot designed for education in engineering. In *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions* (pp. 59–65). Castelo Branco, Portugal: IPCB: Instituto Politécnico de Castelo Branco.

Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J.-C., Floreano, D., & Martinoli, A. (2009b). E-puck website. URL http://www.e-puck.org/. Accessed Nov 2013.

Nolfi, S. (2002). Power and the limits of reactive agents. *Neurocomputing*, *42*(1), 119–145.

Nolfi, S., & Floreano, D. (2000). *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. Cambridge, MA: MIT Press.
Panait, L., & Luke, S. (2005). Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, *11*(3), 387–434.
Parker, L. E. (1996). L-ALLIANCE: Task-oriented multi-robot learning in behavior-based systems. *Advanced Robotics*, *11*(4), 305–322.
Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., et al. (2012). ARGoS: A modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, *6*(4), 271–295.
Pinville, T., Koos, S., Mouret, J.-B., & Doncieux, S. (2011). How to promote generalisation in evolutionary robotics: The ProGAb approach. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation* (pp. 259–266). New York: ACM.
R Development Core Team. (2008). R: A language and environment for statistical computing. Vienna, Austria: R Foundation for Statistical Computing. URL http://www.R-project.org.
Riano, L., & McGinnity, T. M. (2012). Automatically composing and parameterizing skills by evolving finite state automata. *Robotics and Autonomous Systems*, *60*(4), 639–650.
Stranieri, A., Turgut, A., Francesca, G., Reina, A., Dorigo, M., & Birattari, M. (2013). IRIDIA's arena tracking system. Technical report TR/IRIDIA/2013-013, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium.
Trianni, V., & Dorigo, M. (2006). Self-organisation and communication in groups of simulated and physical robots. *Biological Cybernetics*, *95*(3), 213–231.
Trianni, V., & Nolfi, S. (2009). Self-organising sync in a robotic swarm. A dynamical system view. *IEEE Transactions on Evolutionary Computation*, *13*(4), 722–741.
Trianni, V., & Nolfi, S. (2011). Engineering the evolution of self-organizing behaviors in swarm robotics: A case study. *Artificial Life*, *17*(3), 183–202.
Tuci, E. (2009). An investigation of the evolutionary origin of reciprocal communication using simulated autonomous agents. *Biological Cybernetics*, *101*(3), 183–199.
Urzelai, J., Floreano, D., Dorigo, M., & Colombetti, M. (1998). Incremental robot shaping. *Connection Science*, *10*(3–4), 341–360.
Waibel, M., Keller, L., & Floreano, D. (2009). Genetic team composition and level of selection in the evolution of cooperation. *IEEE Transactions on Evolutionary Computation*, *13*(3), 648–660.
Winfield, A. F. T., & Erbas, M. D. (2011). On embodied memetic evolution and the emergence of behavioural traditions in robots. *Memetic Computing*, *3*(4), 261–270.
Wischmann, S., Floreano, D., & Keller, L. (2012). Historical contingency affects signaling strategies and competitive abilities in evolving populations of simulated robots. *Proceedings of the National Academy of Sciences*, *109*(3), 864–868.