Swarm Intell
DOI 10.1007/s11721-015-0107-9

AutoMoDe-Chocolate: automatic design of control software for robot swarms

Gianpiero Francesca[1] · Manuele Brambilla[1] · Arne Brutschy[1] · Lorenzo Garattoni[1] · Roman Miletitch[1] · Gaëtan Podevijn[1] · Andreagiovanni Reina[1] · Touraj Soleymani[1,2] · Mattia Salvaro[1,3] · Carlo Pinciroli[1,4] · Franco Mascia[1] · Vito Trianni[5] · Mauro Birattari[1]

Received: 10 November 2014 / Accepted: 12 May 2015

Abstract We present two empirical studies on the design of control software for robot swarms. In Study A, `Vanilla` and `EvoStick`, two previously published automatic design methods, are compared with human designers. The comparison is performed on five swarm robotics tasks that are different from those on which `Vanilla` and `EvoStick` have been previously tested. The results show that, under the experimental conditions considered, `Vanilla` performs better than `EvoStick`, but it is not able to outperform human designers. The results indicate that `Vanilla`'s weak element is the optimization algorithm employed

The main contributors to this research are G. Francesca and M. Birattari. AutoMoDe and Vanilla were conceived and developed by G. Francesca, M. Brambilla, A. Brutschy, V. Trianni, and M. Birattari. Chocolate was conceived by G. Francesca and M. Birattari. F. Mascia contributed to the implementation. L. Garattoni, R. Miletitch, G. Podevijn, A. Reina, and T. Soleymani acted as human experts: They defined the tasks on which the experimental analysis is performed and they developed control software via `C-Human` and `U-Human`. M. Salvaro developed and operated the software we used to track the robots and to compute the value of the objective functions. C. Pinciroli contributed his experience with the ARGoS simulator. The analysis of the results has been performed by G. Francesca, F. Mascia, and M. Birattari. Most of the manuscript has been drafted by G. Francesca and M. Birattari. M. Brambilla drafted the state of the art on manual design methods and V. Trianni the one on automatic design methods. L. Garattoni, R. Miletitch, G. Podevijn, A. Reina, and T. Soleymani drafted the paragraphs that describe the tasks. All authors read and commented the manuscript. The final editing has been performed by M. Birattari and G. Francesca, with notable contributions from C. Pinciroli and A. Brutschy. The research has been conceived and directed by M. Birattari.

✉ Gianpiero Francesca
gianpiero.francesca@ulb.ac.be

✉ Mauro Birattari
mbiro@ulb.ac.be

1 IRIDIA, Université libre de Bruxelles, Brussels, Belgium

2 Present Address: ITR, Technische Universität München, Munich, Germany

3 Alma Mater Studiorum – Università di Bologna, Bologna, Italy

4 Present Address: MIST, École Polytechnique de Montréal, Montreal, Canada

5 ISTC-CNR, Rome, Italy

Published online: 02 June 2015

to search the space of candidate designs. To improve over Vanilla and with the final goal of obtaining an automatic design method that performs better than human designers, we introduce Chocolate, which differs from Vanilla only in the fact that it adopts a more powerful optimization algorithm. In Study B, we perform an assessment of Chocolate. The results show that, under the experimental conditions considered, Chocolate outperforms both Vanilla and the human designers. Chocolate is the first automatic design method for robot swarms that, at least under specific experimental conditions, is shown to outperform a human designer.

Keywords Swarm robotics · Automatic design · AutoMoDe

1 Introduction

In this paper, we present two empirical studies on the design of control software for robot swarms in which we compare automatic and manual design methods. Moreover, we introduce AutoMoDe-Chocolate (hereafter Chocolate), the first automatic design method for robot swarms that, at least under specific experimental conditions, has outperformed a human designer.

Designing control software for robot swarms is challenging due to the complex relation existing between the individual behavior of each robot and the resulting swarm-level properties: Requirements are naturally expressed at the collective level by stating the characteristics of the desired collective behavior that the swarm should exhibit. Nonetheless, the designer must eventually define what the individual robots should do, so that the desired collective behavior is achieved (Dorigo et al. 2014).

Presently, no general approach exists to derive the individual behavior of the robots from a desired collective behavior—see Sect. 2 for some promising preliminary steps in this direction. Typically, control software for robot swarms is designed manually by trial and error: The designer implements, tests, and modifies the behavior of the individual robots until the desired collective behavior is obtained. Manual design by trial and error completely relies on the intuition and skill of the designer.

Automatic design is an appealing alternative to the manual design process described above. In an automatic method, the design problem is cast into an optimization problem: The solution space comprises instances of control software that conform to a predefined parametric architecture. An optimization algorithm is employed to search the solution space, which amounts to tuning the free parameters of the architecture. Several studies have shown that effective control software for robot swarms can be produced via an optimization process—see Sect. 2. However, these studies owe their success to task-specific expedients, while the problem of creating a general-purpose design method remains open (Trianni and Nolfi 2011).

The focus of our research is on developing a general-purpose, automatic design method capable of producing control software for robot swarms. By "general-purpose" method we mean a method that proves to be effective for a sufficiently large class of tasks, without requiring task-specific modifications.

In our research, we consider the simplest life-cycle model for a robot swarm: specification–development–deployment. In particular, we consider the case in which the three phases can be instantiated as follows.

– Specification: The task to be performed by the swarm is defined and a performance measure is specified.

– Development: The control software of the robots is designed and implemented with the support of computer-based simulations.
– Deployment: The control software is uploaded onto the robots and the swarm is deployed to perform the assigned task.

More complex life cycles can be conceived. Nonetheless, as the engineering of robot swarms is at its dawn, we think that focusing on a basic life cycle is more appropriate and avoids unnecessary complications. Stated in the terms of the specification–development–deployment life cycle, the goal of our research is to define a method that is able to perform the development step in an automatic way. Once the requirements are given in the form of an objective function to maximize or minimize, the method must be able to produce the desired control software without any human intervention. To define an effective design method, a critical issue that one has to address concerns the transition between development and deployment. In this transition, the control software is required to overcome the so-called *reality gap* (Jakobi et al. 1995)—the unavoidable difference between the models used in the computer-based simulations of the development step and the real world to be faced in the deployment step.

In Francesca et al. (2014c), we introduced AutoMoDe, as a first step in the definition of general-purpose automatic design methods. AutoMoDe is an approach in which control software is automatically designed in the form of a probabilistic finite state machine, by combining and fine-tuning preexisting parametric modules. Moreover, we defined AutoMoDe-`Vanilla` (hereafter `Vanilla`), a first method that complies with the AutoMoDe approach. More precisely, `Vanilla` is a specialization of AutoMoDe for a version of the e-puck robot (Mondada et al. 2009). We compared `Vanilla` with `EvoStick` (Francesca et al. 2014c), a design method that uses an evolutionary algorithm to optimize a neural network. The comparison was based on two classic swarm robotics tasks: aggregation and foraging. The results show that on both tasks `Vanilla` outperforms `EvoStick`.

In this paper, our aim is to (i) perform an objective comparison of some automatic and manual design methods and (ii) present the first automatic design method that is shown to outperform human designers, at least under specific experimental conditions.

We compare `Vanilla`, `EvoStick`, and two manual design methods that we call `U-Human` and `C-Human`. In `U-Human`, the human designer is *unconstrained* and implements the control software without any restriction on the structure of the design. `U-Human` closely mimics the way in which most control software for robot swarms is currently produced (Brambilla et al. 2013). In `C-Human`, the human designer is *constrained* to implement control software by combining the same parametric modules on which `Vanilla` operates. A detailed description of the four methods under analysis is given in Sect. 3. We perform the study on five tasks defined by researchers that, at the moment of defining the tasks, were neither aware of the functioning of `Vanilla` and `EvoStick`, nor informed on which design methods were included in the study. This ensures that the definition of the tasks is neutral and that no a priori advantage is granted to any method. The versions of `Vanilla` and `EvoStick` that we adopt are exactly the same that were described in Francesca et al. (2014c). In other words, `Vanilla` and `EvoStick` were developed before the five tasks were defined and did not undergo any modification to adapt them to these five tasks. This is consistent with our quest for a true general-purpose automatic design method.

In the following, we refer to this study as Study A. Study A answers two questions: (i) Whether `Vanilla` performs better than `EvoStick` also on the new tasks, and (ii) whether `Vanilla` performs better than the manual design methods `U-Human` and `C-Human`. Study A is reported in Sect. 4. The results show that also on the five new tasks, `Vanilla` outperforms `EvoStick`. Moreover, `Vanilla` outperforms `U-Human`. However, `Vanil-`

la performs worse than C-Human. As Vanilla and C-Human operate on the same set of modules, the difference in performance is to be ascribed to the mechanism adopted by Vanilla to combine and fine-tune the modules: the optimization algorithm.

In the light of this conclusion, in Sect. 5 we introduce Chocolate, an improved version of Vanilla that is based on a more effective optimization algorithm. To assess Chocolate, we perform a second empirical study: Study B. The goal of this study is to confirm the following working hypotheses: (i) by adopting a more advanced optimization algorithm, Chocolate improves over Vanilla; and, most importantly, (ii) the improvement is such that, under the experimental conditions considered, Chocolate outperforms C-Human. Study B is reported in Sect. 6. The results confirm both working hypotheses.

The research presented in this paper advances the state of the art in the automatic design of robot swarms in two main respects: (i) We introduce Chocolate, the first automatic design method for robot swarms that is shown to outperform a human designer, albeit under specific experimental conditions. (ii) We present the first comparison of automatic and manual methods for the design of control software for robot swarms. This is the first comparison performed on multiple tasks without any task-specific modification of the methods under analysis. The experimental protocol we adopt is a contribution per se and can easily be extended/adapted to any study that aims to compare automatic and manual design methods. The empirical studies presented in the paper are unprecedented in the domain of the automatic design of control software for robot swarms—they comprise 350 runs with a swarm of 20 robots; a total of five methods are tested on five tasks. This paper is an extended version of Francesca et al. (2014b), which was presented at ANTS 2014. The results of Study A were already contained in Francesca et al. (2014b), while the ones of Study B are presented here for the first time.

2 Related work

In this section, we discuss studies that propose or support principled manual design methods and automatic or semiautomatic design methods for swarm robotics. We focus on studies that prove the viability of the proposed method through robot experiments—as we do in this paper—although we mention also some promising ideas that have not been yet validated via robot experiments.

Principled manual design methods. The core issue with the trial-and-error approach is that it does not explicitly address the problem of deriving the individual behavior from the desired collective one. Some works have proposed ideas to address this issue. Unfortunately, most of them rely on strong assumptions and are not of general applicability as they have been conceived for specific tasks. The following is a brief overview of some of the most promising ideas. For a comprehensive review, we refer the reader to Brambilla et al. (2013).

Martinoli et al. (1999) used rate equations to model a collective clustering behavior and to guide the implementation of the control software of the individual robots. The method was assessed both in simulation and with up to ten Khepera robots (Mondada et al. 1993). Lerman et al. (2001) and Martinoli et al. (2004) applied rate equations to a cooperative stick pulling task. The control software produced was tested with up to six Kheperas. Lerman and Galstyan (2002) used rate equations to model a foraging behavior under the effect of interference.

Kazadi et al. (2007) used a method based on artificial vector fields to develop a pattern formation behavior. The method is illustrated with simulations and appears to be limited

to spatially organizing behaviors. Hsieh et al. (2007) proposed an approach based on artificial potentials to obtain control software for coordinated motion along predefined orbital trajectories. The authors provided convergence proofs and simulated experiments. Similarly, Sartoretti et al. (2014) proposed an approach based on stochastic differential equations driven by white Gaussian noise to tackle coordinated motion. In this case, the orbital trajectory is derived via collective consensus among the robots of the swarm. The approach has been validated with a swarm of eight e-puck robots.

Hamann and Wörn (2008) used Langevin equations to model the behavior of the individual robots, and analytically derived a Fokker-Planck equation that models the collective behavior of the swarm. Berman et al. (2011) adopted a similar approach based on a set of advection-diffusion-reaction partial differential equations to design control software for task allocation. None of the two approaches has been assessed in robot experiments yet.

Lopes et al. (2014) introduced an approach based on supervisory control theory. The approach has been demonstrated by designing a segregation behavior. The assessment has been performed with a swarm of 26 e-pucks and one of 42 kilobots (Rubenstein et al. 2014). The main drawback of this approach is that it requires extensive domain knowledge.

Brambilla et al. (2014) introduced an approach based on prescriptive modeling and model checking. The approach has been demonstrated by designing control software for two tasks: aggregation and foraging. The assessment has been performed with swarms of up to 20 e-pucks. Also in this case, the approach requires extensive domain knowledge.

Automatic and semiautomatic design methods. The automatic design of control software for robot swarms has been pursued mainly within the evolutionary robotics domain, in which the standard methodologies employed in the single robot case have been extended toward multi-robot systems (Trianni 2008). Following the main tradition of evolutionary robotics, several studies demonstrated the possibility of designing control software in the form of a neural network (Quinn et al. 2003; Baldassarre et al. 2007; Trianni and Nolfi 2009; Hauert et al. 2008; Groß and Dorigo 2009; Izzo et al. 2014). Research studies often sway between providing an engineering solution and modeling biological systems (Trianni 2014). Notwithstanding the large number of robot swarms successfully designed via an evolutionary process, an engineering methodology for the application of evolutionary robotics is still unavailable (Trianni and Nolfi 2011). In the following, we discuss three techniques that have been proposed as contributions to the definition of an engineering methodology in evolutionary robotics (for a recent review, see Doncieux and Mouret 2014).

Multi-objectivization has been proposed as a general way to guide the evolutionary search in rugged fitness landscapes and avoid bootstrap problems, both problems severely affecting the evolution of control software for robot swarms (Trianni and López-Ibáñez 2014). However, no test with robots has been performed to date, and multi-objective evolution could be affected by the reality gap problem as much as single-objective evolution. The possibility to deal with the reality gap by adding a specific objective—as demonstrated by Koos et al. (2013)—makes the approach promising in the general case.

Novelty search has been proposed by Lehman and Stanley (2011) as a technique for promoting diversity among possible behaviors and improving the exploration of the search space. Gomes et al. (2013) extended the technique to swarm robotics and provided also a solution that combines novelty search and fitness-based techniques through scalarization of the respective scores. No quantitative results with robots have been provided to date.

Hierarchical decomposition has been proposed by Duarte et al. (2014b) as an approach to scale in task complexity. The design problem is tackled by decomposing the control software into modules that are either evolved or manually developed. The hierarchical decomposition is

performed manually by the designer and is task-specific. The approach has been successfully extended to multi-robot systems (Duarte et al. 2014a), but no test with robots has been performed yet. The transfer to reality of the control software in the single robot case (Duarte et al. 2014b) makes the proposed technique promising also for robot swarms.

A number of studies on online adaptation in multi-robot systems are related to evolutionary robotics. In these studies, population-based optimization algorithms are implemented in a decentralized way exploiting the robots as computational nodes. Watson et al. (2002) introduced embodied evolution as a technique to distribute an evolutionary algorithm over a group of robots. Since its introduction, several studies have tested the feasibility of the approach, proposing algorithms for open-ended and task-dependent evolution (Bredeche et al. 2012; Haasdijk et al. 2014). König and Mostaghim (2009) proposed the usage of finite state machines within embodied evolution. However, the problems studied are rather simple and no test with robots has been performed.

Further studies about online adaptation depart from the implementation of distributed evolutionary algorithms. Winfield and Erbas (2011) exploited an imitation-based algorithm to explore the idea of cultural evolution within robot swarms. Pugh and Martinoli (2009) implemented a distributed version of particle swarm optimization, and Di Mario and Martinoli (2014) extended the approach to a hybrid simulate-and-transfer setup.

Related to automatic design are those studies in swarm robotics in which the control architecture is fixed and only a small set of parameters is tuned. Hecker et al. (2012) used a genetic algorithm to optimize the parameters of a finite state machine for a cooperative foraging task. Gauci et al. (2014a) used evolutionary strategies to optimize the six parameters of the control software for an object clustering task. The same authors used exhaustive search to tune the parameters of similar control software for self-organized aggregation (Gauci et al. 2014b). In these studies, the distinction between manual design with some parameter tuning and a truly automatic design is somewhat blurred.

Empirical assessments of automatic design methods for robot swarms that have been conducted with swarms of a reasonably large size are rare in the literature. To the best of our knowledge, the only automatic design methods that have been empirically tested in experiments involving swarms of at least ten robots are the ones presented by Gauci et al. (2014a, b) and Francesca et al. (2014c). Those presented by Francesca et al. (2014c), that is, `Vanilla` and `EvoStick`, are the only automatic design methods that have been tested on more than one task without undergoing any task-specific modification. No automatic design method for robot swarms has been so far compared against a human designer in a controlled experiment.

3 Four design methods for a swarm of e-pucks

In this section, we describe four methods that design control software for a swarm of e-puck robots: `Vanilla`, `EvoStick`, `C-Human`, and `U-Human`. To be precise, these methods operate with a subset of the capabilities of the e-puck platform that are formally described by the reference model introduced by Francesca et al. (2014c). In this paper, we call this reference model RM1.

The e-puck maneuvers by actuating its two wheels, which constitute a differential steering system (Mondada et al. 2009). The version of the e-puck adopted in our research is shown in Fig. 1. This version of the e-puck is equipped with eight infrared transceivers, three ground sensors, and a range-and-bearing board. The infrared transceivers are placed around the body of the e-puck and are used as light and proximity sensors. The ground sensors are placed under the front of the e-puck and measure the reflectance of the floor, which allows the

Fig. 1 Front and side view of an e-puck robot and a tag used to localize the robot via a ceiling-mounted camera

Table 1 Reference model RM1

Input variable	Values	Description
$prox_{i\in\{1,2,\ldots,8\}}$	[0, 1]	Reading of proximity sensor i
$light_{i\in\{1,2,\ldots,8\}}$	[0, 1]	Reading of light sensor i
$gnd_{j\in\{1,2,3\}}$	{black, gray, white}	Reading of ground sensor j
n	$\{0,\ldots,20\}$	Number of neighboring e-pucks
$r_{m\in\{1,2,\ldots,n\}}$	[0, 0.70] m	Distance of neighbor m
$\angle b_{m\in\{1,2,\ldots,n\}}$	$[0, 2\pi]$ rad	Angle of neighbor m
Output variable	**Values**	**Description**
$v_{k\in\{l,r\}}$	$[-0.16, 0.16]$ m/s	Target linear wheel velocity

Period of the control cycle: 100 ms

e-puck to distinguish at least three levels of gray. The range-and-bearing board (Gutiérrez et al. 2009) allows the e-puck to perceive the presence of other e-pucks in a 0.70 m range. For each perceived e-puck, the range-and-bearing board computes the distance (range) and the relative angle (bearing).[1]

Reference model RM1 is a formalization of the capabilities of the e-puck that are described above: RM1 abstracts sensors and actuators by defining the input and the output variables that are made available to the control software at each control step. Sensors are defined as input variables: The control software can only read them. Actuators are defined as output variables: The control software can only write them. Input and output variables are updated with a period of 100 ms. The reference model RM1 is summarized in Table 1. According to RM1, the reading of a proximity sensor i is stored in the variable $prox_i$, which ranges between 0 and 1. When sensor i does not perceive any obstacle in a 0.03 m range, $prox_i = 0$; while when sensor i perceives an obstacle closer than 0.01 m, $prox_i = 1$. Similarly, the reading of a light sensor i is stored in the variable $light_i$, which ranges between 0, when no light source

[1] The range-and-bearing board also allows the e-pucks to exchange messages. However, this functionality is not included in RM1.

is perceived, and 1, when the sensor i saturates. The readings of the three ground sensors are stored in the variables gnd_1, gnd_2, and gnd_3. These variables can take three different values: black, gray, and white. The e-puck uses the range-and-bearing board to perceive other e-pucks in its neighborhood. The variable n stores the number of the neighboring e-pucks. For each neighboring e-puck $m \in \{1, 2, \ldots, n\}$, the variables r_{m} and $\angle b_{\mathrm{m}}$ indicate the range and the bearing, respectively. The wheel actuators are operated by the control software through the variables v_l and v_r, in which the control software writes the target linear velocity for the left and right wheel, respectively. The linear wheel velocity ranges between -0.16 m/s and 0.16 m/s.

In the rest of this section, we describe the four design methods: `Vanilla`, `EvoStick`, `U-Human`, and `C-Human`.

3.1 `Vanilla`

`Vanilla` produces robot control software by assembling preexisting modules into a probabilistic finite state machine. The modules operate on the variables defined in RM1. Modules might have parameters that regulate their internal functioning. The parameters, along with the topology of the probabilistic finite state machine, are optimized in order to maximize a task-dependent performance measure.

`Vanilla` assembles and tunes the parameters of two kinds of modules: *behaviors* and *transitions*. A behavior is an activity that the robot can perform, while a transition is a criterion to regulate the change of behavior in response to a particular condition or event experienced by the robot. In practice, a behavior is a parametric procedure that sets the output variables defined in RM1 on the basis of the value of (a subset of) the input variables. A transition is a parametric procedure that returns either true or false on the basis of the value of (a subset of) the input variables. In the parlance of probabilistic finite state machines, states and edges are instances of behaviors and transitions, respectively. More precisely, a state (edge) is an instance of a behavior (transition) in which the parameters, if any, are given a valid value. Different states (edges) might be instances of the same behavior (transition), possibly with different values of the parameters.

An execution of the control software is a series of control steps of 100 ms each. At any given control step, the probabilistic finite state machine is in one and only one state, which we refer to as the active state. The instance of the behavior associated with the active state is executed, that is, output variables are set, on the basis of the input variables, as prescribed by the behavior. Subsequently, if at least one outgoing transition returns true, the control software changes state: One transition among the ones that returned true is randomly selected and the state pointed by the selected transition becomes the active state for the following control step. If no transition returns true, the active state remains unchanged. The execution of the control software then moves on to the following control step.

In `Vanilla`, twelve modules are available for being assembled into a probabilistic finite state machine: six behaviors and six transitions. The six behaviors are: exploration, stop, phototaxis, anti-phototaxis, attraction, and repulsion. With the exception of stop, these behaviors include an obstacle avoidance mechanism. The six transitions are: black-floor, gray-floor, white-floor, neighbor-count, inverted-neighbor-count, and fixed-probability. We refer the reader to `Vanilla`'s original paper for a detailed description of the modules (Francesca et al. 2014c).

The finite state machine and the parameters of the modules are obtained via an optimization process. The space of feasible solutions searched by `Vanilla` is the space of the probabilistic finite state machines that comprise up to four states and up to four outgoing edges from each

state; the behaviors and the transitions to be associated with states and edges, respectively, are sampled with replacement from the available modules.

The goal of the optimization is to maximize the expected value of a task-specific performance measure; where the expectation is taken with respect to the initial conditions and the contingencies of task execution. Each different initial condition starting from which the task has to be performed is reproduced through a different test case on which solutions are evaluated. The contingencies of task execution are accounted for through realistic computer-based simulations that reproduce sensor and actuator noise. Specifically, a solution is evaluated on a test case by means of ARGoS (Pinciroli et al. 2012), a physics-based simulator of swarm robotics systems that includes a detailed model of the e-puck robot.

The optimization algorithm adopted by `Vanilla` is F-Race (Birattari et al. 2002; Birattari 2009). In F-Race, a set of candidate solutions are sequentially evaluated over different test cases in a process that is reminiscent of a race. The aim of the process is to select the best candidate solution. The set of candidate solutions is sampled in a uniformly random way from the space of feasible solutions. The F-Race algorithm comprises a series of steps. At each step, a different test case is sampled and is used to evaluate the candidate solution. At the end of each step, a Friedman test is performed on the basis of the results obtained by the candidate solutions on the test cases sampled so far. All candidate solutions that appear to perform significantly worse than at least another one are dropped from the set of candidate solutions and are not evaluated further in the subsequent steps. The process stops either when a single candidate remains or when a predefined budget of evaluations has been spent. By discarding as early as possible the candidates that are statistically dominated by at least another candidate, the evaluation process implemented by the F-Race algorithm allows for a rational and efficient use of the available evaluation budget (Birattari 2009).

The implementation of F-Race that is adopted in `Vanilla` is the one provided by the *irace* package (López-Ibáñez et al. 2011) for R (R Core Team 2014). `Vanilla` uses the default parameters of F-Race provided by the *irace* package and samples the design space using the built-in sampling procedure of *irace*.

3.2 `EvoStick`

`EvoStick` is an automatic design method that implements a typical evolutionary robotics setup. We introduced `EvoStick` in Francesca et al. (2014c) with the goal of defining a yardstick against which we could compare `Vanilla`. By giving this automatic design method a name, we wish to define a specific and fully characterized evolutionary robotics setup for robots conforming to RM1, in which all parameters are given a precise and immutable value.

`EvoStick` generates control software in the form of a fully connected, feed-forward neural network without hidden nodes. Inputs and outputs of the network are defined on the basis of the variables given in RM1. The neural network has 24 inputs and 2 outputs. The inputs are 8 proximity sensors, 8 light sensors, 3 ground sensors, and 5 values computed from the messages received by the range-and-bearing board. The two outputs act on the two wheels by setting the target speed velocity. The neural network is described by 50 parameters. Each parameter is a real value in the range $[-5, 5]$. In `EvoStick`, the parameters of the neural network are encoded in a real-valued vector and are optimized via an evolutionary algorithm that involves mutation and elitism. At the beginning, a population of 100 neural networks is randomly generated. Each neural network of the population is evaluated via 10 runs in simulation using ARGoS. To constitute the new population of neural networks, elitism and mutation are applied. The elite—that is, the 20 best performing neural networks—are included unmodified in the new population. The remaining 80 neural networks of the new

population are obtained by mutating the individuals of the elite. Mutations are performed by adding a random value drawn from a normal distribution (with mean 0 and variance 1) to each parameter of the neural network. The evolutionary algorithm stops when a predefined number of iterations is reached. The final population is then evaluated again in order to select the best neural network, that is, the one with the highest mean performance.

`EvoStick` is similar to some previously published methods in a number of respects. For example, `EvoStick` shares the control architecture, the encoding of the parameters, and the optimization algorithm (albeit with different parameters) with the evolutionary robotics method described in Francesca et al. (2012). The two methods differ in the inputs that are fed to the neural network. `EvoStick` is also similar to the methods proposed by Ampatzis et al. (2009) and Tuci et al. (2011). The three methods share the encoding of the parameters and the optimization algorithm (albeit with different parameters). The main difference is the structure of the control architecture: The methods proposed by Ampatzis et al. (2009) and Tuci et al. (2011) adopt a neural network that includes hidden nodes. To the best of our knowledge, `EvoStick` is the only evolutionary robotics method that has been tested on more than one task without undergoing any task-specific modification (Francesca et al. 2014c).

3.3 `U-Human`

`U-Human` is a manual design method in which a human designer implements the control software in the way (s)he deems appropriate, without any kind of restriction regarding the design to produce. The designer realizes a trial-and-error process: The control software is iteratively improved and tested until the desired behavior is obtained. Within this process, the designer assesses the quality of the control software by computing the value of the objective function and by observing the resulting behavior via the simulator's visual interface. As in the case of `Vanilla` and `EvoStick`, during the development of the control software, the designer is allowed to perform tests in simulation using ARGoS, but is not allowed to perform tests with the robots. In the implementation of the control software, the designer is free to access all the resources (s)he deems appropriate including the internet and her/his own previously developed code.

The control software is implemented as a C++ class that operates on the input and output variables defined in RM1. These variables are manipulated by the control software via an API. The designer is provided with a complete programming and simulation environment based on ARGoS. Moreover, the designer is provided with the description of the task to be solved, a control software skeleton to be used as a starting point, the task-specific objective function to be optimized, and all the scripts that initialize ARGoS for the task at hand. The control software skeleton is an empty C++ class that complies with the specification of a valid control software for ARGoS. In other terms, the skeleton is a valid control software that compiles and runs correctly but that leaves the robot motionless in its initial position. The designer is required to fill in the skeleton with the appropriate logic for solving the given task. To reduce the burden on the designer, the skeleton contains commented instructions to access the variables of RM1 via the API. The task-specific objective function computes the performance of the swarm within the simulation. It is implemented via *loop functions*, which in ARGoS parlance are callback functions executed at each step of the simulation (Pinciroli et al. 2012). The objective function is computed automatically by the simulation environment in a way that is completely transparent to the designer. To ease the assessment of the control software being implemented, a utility script is provided. The script compiles the control software, starts ARGoS, generates the simulated arena, runs and visualizes the

simulation, and prints the value of the objective function. The designer is allowed to use debugging tools including gdb[2] and valgrind.[3]

3.4 C-Human

C-Human is a manual method in which the human designer is constrained to use Vanilla's control architecture and modules. In other words, the human designer takes the role of Vanilla's optimization algorithm and searches the same design space searched by Vanilla. As in Vanilla, the human is constrained to create finite state machines comprised of at most four states, each with at most four outgoing transitions—see Sect. 3.1 for the details on the restrictions on the finite state machines produced by Vanilla. As in U-Human, in C-Human the designer iteratively improves the control software in a trial-and-error process that comprises implementation phases interleaved with testing via simulation. The only difference between U-Human and C-Human is that in the case of C-Human, the designer implements the control software by combining the modules of Vanilla and setting their parameters, rather than directly writing C++ source code. To allow the designer to implement the control software in this fashion, a user interface is provided. The user interface allows the designer to specify the probabilistic finite state machine using a simple finite language. The user interface also graphically visualizes the probabilistic finite state machine specified by the designer. An example of a statement in this language is given in Fig. 2, together with the graphical visualization produced by the user interface. The user interface also starts ARGoS, generates the simulated arena, runs and visualizes the simulation, and prints the value of the objective function.

4 Study A: comparison of four design methods for RM1

The goal of this study is to compare the design methods described in Sect. 3.

4.1 Experimental protocol

In both studies proposed in the paper, a central role is played by five researchers, hereinafter referred to as experts.[4] The experts are Ph.D. candidates with about 2 years of experience in the domain of swarm robotics. They have previously worked with the e-puck platform or with similar platforms. They are familiar with the ARGoS simulator and programming environment.[5] Within the protocol, each expert plays a threefold role: (i) define a task, (ii) solve a task via U-Human, and (iii) solve a task via C-Human. The tasks solved by an expert via U-Human and C-Human are different from each other and from the one proposed by the expert himself. Experts are not allowed to exchange information throughout the duration of the empirical study. The roles of each expert is summarized in Table 2.

[2] https://www.gnu.org/software/gdb/.

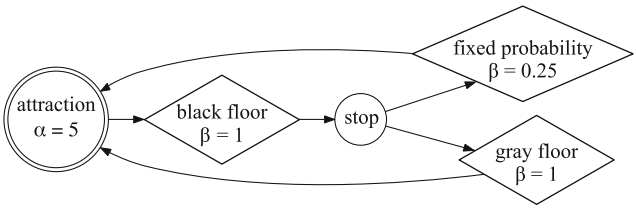
[3] http://valgrind.org/.

[4] With the goal of establishing accountability and credit, the five experts are included among the authors of this paper.

[5] We think that PhD candidates are ideal subjects for this study. Indeed, it is our understanding that a large share of the robot swarms described in the domain literature have been programmed by PhD candidates. See Francesca et al. (2014a) for data extracted from the publication record of our research laboratory.

```
--nstates 2
--s0 attraction --alpha0 5 --n0 1
  --n0x0 1 --c0x0 black-floor --beta0x0 1
--s1 stop  --n1 2
  --n1x0 0 --c1x0 fixed-probability --beta1x0 0.25
  --n1x1 0 --c1x1 gray-floor --beta1x1 1
```

(a)



(b)

Fig. 2 Example of a probabilistic finite state machine specified in the simple finite language adopted in `C-Human` (*a*) and its graphical visualization (*b*). *a* A finite state machine described by a statement in the language adopted by `C-Human`. The probabilistic finite state machine comprises 2 states. State 0 is "attraction," with parameter $\alpha = 5$, and has outdegree 1: Edge 0 is connected to state 1, the condition for the transition is "black-floor," with parameter $\beta = 1$. State 1 is "stop" and has outdegree 2: Edge 0 is connected to state 0, the transition is activated with fixed-probability 0.25; edge 1 is connected to state 0, the condition for the transition is "gray-floor," with parameter $\beta = 1$. *b* The resulting probabilistic finite state machine

Table 2 Role of the experts, anonymously indicated here by the labels E1– E5

Task	Defined by	`U-Human`	`C-Human`
SCA – shelter with constrained access	E1	E5	E4
LCN – largest covering network	E2	E1	E5
CFA – coverage with forbidden areas	E3	E2	E1
SPC – surface and perimeter coverage	E4	E3	E2
AAC – aggregation with ambient cues	E5	E4	E3

For each row, the column "task" gives the name of the task; the column "defined by" identifies the expert that has defined the task; the columns "`U-Human`" and "`C-Human`" identify the experts that have solved the task acting as `U-Human` and `C-Human`, respectively. The tasks defined by the experts are described in Sect. 4.1.2

4.1.1 Definition of the tasks

In the definition of the tasks, the experts are kept unaware of the design methods included in the empirical study, in order to avoid any influence in the experts' choices that could favor one method over the others. Experts are asked to define tasks that, according to their judgment, could be performed by a swarm of 20 robots conforming to RM1. The experts are given a set of constraints that the tasks must satisfy: The time available to the robots for performing a task is $T = 120$ s. The robots operate in a dodecagonal area of 4.91 m^2 surrounded by walls. The floor of the arena is gray. Up to three circular or rectangular patches may be present on the floor. The patches may be either white or black. The diameter of the circular patches

and the sides of the rectangular patches cannot exceed 0.6 m. The environmental setup may include a light source placed outside the south side of the arena. Up to 5 obstacles may be present in the arena. Obstacles are wooden cuboids of size 0.05 m × 0.05 m × L, where L is in the range [0.05, 0.80] m.

As part of the task definition, the experts are asked to define the task-specific performance measure that will be used to assess task execution. The performance measure should be computable on the basis of the position and orientation of the robots, evaluated every 100 ms.

The procedure through which an expert defines a task can be interpreted as a sampling according to an unknown distribution defined over the space of tasks that can be performed by a swarm of 20 robots conforming to RM1 and that satisfy the given environmental constraints. The tasks that are relevant to our study can be defined in terms of the sampling procedure: The higher the probability that a task is sampled, the higher the relevance of the task.

4.1.2 Description of the tasks defined by the experts

The following are the tasks defined by the experts according to the procedure given in Sect. 4.1.1. Overhead shots of the arenas are given in Fig. 3.

SCA–shelter with constrained access. The arena contains a rectangular white region of 0.15 m × 0.6 m. This region is closed on three sides by obstacles: Only the south side is open for the robots to enter. In the arena, there are also two black circular patches, positioned aside the white region. The two circular patches have the same diameter of 0.6 m. The setup also includes a light source placed on the south side of the arena. The task for the robots is to aggregate on the white region: the shelter. The robots can use the light source and the black circular patches to orientate themselves. The performance measure is defined in terms of an objective function to maximize: $F_{\text{SCA}} = \sum_{t=1}^{T} N(t)$, where $N(t)$ is the number of robots in the shelter at time t and T is the time available to the robots for performing the task.

LCN–largest covering network. The arena does not contain any obstacle, floor patch, or light source. The robots are required to create a connected network that covers the largest area possible. Each robot covers a circular area of 0.35 m radius. Two robots are considered to be connected if their distance is less than 0.25 m. The performance measure is defined in terms of an objective function to maximize: $F_{\text{LCN}} = A_{C(T)}$, where $C(T)$ is the largest network of connected robots at the end T of the time available for performing the task and $A_{C(T)}$ is the area covered by $C(T)$.

CFA–coverage with forbidden areas. The arena contains three circular black regions, each with a diameter of 0.6 m. The robots are required to cover the arena, avoiding the forbidden areas denoted by the black-floor. The performance measure is defined in terms of an objective function to minimize: $F_{CFA} = E[d(T)]$, where $E[d(T)]$ is the expected distance, at the end T of the time available for performing the task, between a generic point of the arena and the closest robot that is not in the forbidden area. This objective function is measured in meters.

SPC–surface and perimeter coverage. The arena contains a circular black region with a diameter of 0.6 m and a square white region with sides of 0.6 m. The robots are required to aggregate on the perimeter of the black circle and to cover the area of the white square. The performance measure is defined in terms of an objective function to minimize: $F_{SPC} = E[d_a(T)]/c_a + E[d_p(T)]/c_p$, where $E[d_a(T)]$ is the expected distance, at the end T of the time available for performing the task, between a generic point in the square region and the closest robot that is in the square region and $E[d_p(T)]$ is the expected distance between a

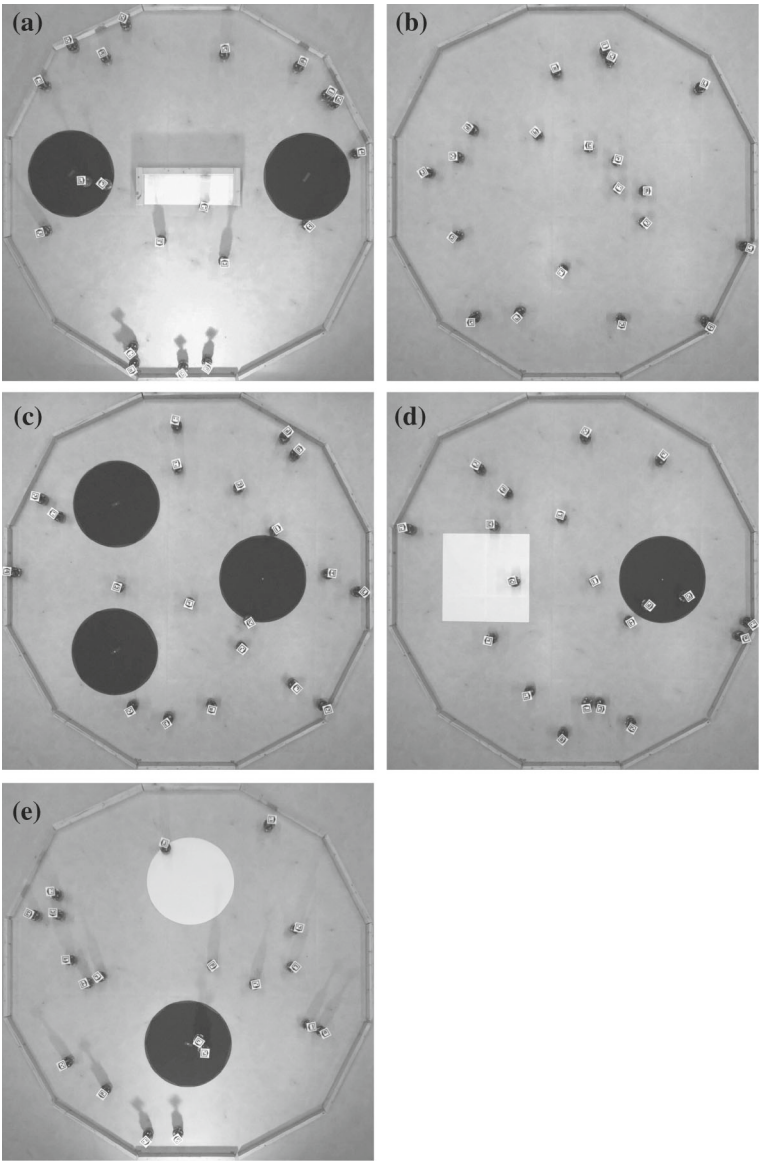


Fig. 3 Overhead shots of the arenas used for the five tasks defined by the experts. The pictures show also the 20 e-puck robots. *a* SCA – shelter with constrained access: robots must aggregate in the white region, the shelter. *b* LCN – largest covering network: robots must create a connected network that covers the largest area possible. *c* CFA – coverage with forbidden areas: robots must cover all the arena except the forbidden *black* regions. *d* surface and perimeter coverage: robots must cover the area of the *white square* and the perimeter of the *black circle*. *e* AAC – aggregation with ambient cues: robots must aggregate on the *black circle*

generic point on the circumference of the circular region and the closest robot that intersects the circumference. $c_a = 0.08$ and $c_b = 0.06$ are scaling factors that correspond to the values of $E[d_a]$ and $E[d_p]$, respectively, under the ideal condition in which 9 robots are regularly

and equally spaced on the surface of the white square and 9 on the perimeter of the black circle. See Francesca et al. (2014a) for more details. If no robot is on the surface of the square region and/or on the perimeter of the circular region, $E[d_a(T)]$ and/or $E[d_p(T)]$ are undefined and we thus assign an arbitrarily large value to F_{SPC}. We consider this a major failure.

AAC–aggregation with ambient cues. The arena contains two circular regions, one black and one white, each with a diameter of 0.6 m. The black region is placed closer to the light source, which is on the south side of the arena. The robots have to aggregate on the black region and can use the light and the white region to orientate themselves. The performance measure is defined in terms of an objective function to maximize: $F_{\text{AAC}} = \sum_{t=1}^{T} N(t)$, where $N(t)$ is the number of robots on the black region at time t.

4.1.3 Design methods under analysis and experimental setup

We compare `Vanilla`, `EvoStick`, `U-Human`, and `C-Human`. These four design methods are tested under the same conditions:

- Same platform: All methods target the same robotic platform: the specific version of the e-puck formally defined by RM1.
- Same simulator: All methods employ ARGoS as a simulation software to evaluate design candidates.
- Same performance measures: All methods base the evaluation of a design candidate on the same task-specific performance measures.
- Same resources: To design the control software, the four methods are given a similar amount of time, with a slight advantage to human designers. `U-Human` and `C-Human` are given 4 h per task. Time starts when the human designer receives the description of the task. `Vanilla` and `EvoStick` are given a budget of 200,000 executions of ARGoS per task. `Vanilla` and `EvoStick` are executed on a computer cluster that comprises 400 opteron6272 cores. Under this setting, `Vanilla` and `EvoStick` are able to complete a design session in approximately 2 h and 20 min, wall-clock time.

It is important to notice that simulation plays a different role in automatic and manual design. `Vanilla` and `EvoStick` utilize simulation only to compute the value of the objective function. This value is then used by the optimization algorithm to steer the search process. Beside the value of the objective function, no other piece of information is retained from the simulation. A graphical visualization of the simulation is not needed and is not therefore performed. When a graphical visualization is not performed, ARGoS is much faster than real time. As a consequence, the main benefit that `Vanilla` and `EvoStick` obtain from not requiring a graphical visualization is the fact that the objective function can be computed in a relatively short amount of time.

In contrast, human designers greatly benefit from observing the whole evolution of the simulation: The human designer observes the resulting behavior of the swarm and gets insights on how to improve the control software. Arguably, for a human designer visual observation is more informative than the value of the objective function. Both in `U-Human` and `C-Human`, the designer can choose to speed up the visualization with respect to real time or even to disable visualization altogether. By performing simulations with visualization (at some appropriate speedup level), the human designer trades simulation speed for useful information.

4.1.4 Assessment on a swarm of e-pucks

The control software produced by `Vanilla`, `EvoStick`, `U-Human`, and `C-Human` for each task is assessed via test runs with a swarm of 20 e-puck robots.

In this study we adopt a hands-off approach that reduces human intervention to a bare minimum. The control software is directly cross-compiled by the ARGoS simulator, and it is uploaded onto each e-puck of the swarm without any modification. To reduce the risk that the negative effects of battery discharge and other environmental contingencies affect one method more than another, the order of the test runs is randomly generated so that runs with the control software produced by the four design methods are interleaved. The initial position of the e-pucks is generated by placing the e-pucks in known positions and letting them perform a random walk for 20 s. This effectively yields a randomized starting condition for each run.

To compute the task-dependent performance measure we use a tracking system (Stranieri et al. 2013) that gathers data via a ceiling-mounted camera. The tracking system logs position and orientation of each e-puck every 100 ms.

4.1.5 Objective of the study and statistical tools

The objective of the study is to compare the four design methods. We wish to answer two questions: (i) Whether `Vanilla` performs better than `EvoStick` on the tasks proposed by the experts; and (ii) whether `Vanilla` performs better than a human designer, represented here by the `U-Human` and `C-Human` methods.

As discussed in Sect. 4.1.1, the selected tasks can be seen as a sample extracted from a class of tasks. As such, these tasks allow one to draw conclusions that generalize, in a statistical sense, to the class of tasks from which they have been sampled. For this reason, we concentrate our attention on the aggregate performance of the methods over the tasks considered. For the sake of completeness, we report also a boxplot of the per-task performance and the results obtained in simulation by the control software produced by the methods under analysis. Nonetheless, the focus of our study remains the aggregate analysis.

For each task, we perform 40 independent runs: 10 for the control software generated by each of the four methods under analysis. We analyze the results using the Friedman test (Conover 1999), with the task as a blocking factor. As the Friedman test is a rank-based nonparametric test, it does not require scaling the performance measure computed for each of the tasks nor formulating any restrictive hypothesis on the underlying distribution of the different performance measures. This test requires only to convert the objective functions of all tasks into the objective functions of the equivalent minimization problems. Given the rank-based nature of the Friedman test, this operation is trivial: It can be performed via any function that inverts the rank order. Specifically, to obtain a minimization problem from a maximization one, we use as objective function the inverse of the original one. We represent the result of the Friedman test in a graphical way: a plot that shows the expected rank obtained by each design method, together with a 95 % confidence interval. If the confidence intervals of two methods do not overlap, the difference between the expected rank of the two is statistically significant.

Concerning the per-task results of the four design methods, we present five notched box-and-whisker boxplots: one for each task. A notched box-and-whisker boxplot gives a visual representation of a sample. The horizontal thick line denotes the median. The lower and upper sides of the box are called upper and lower hinges and represent the 25th and 75th percentile of the observations, respectively. The upper whisker extends either up to the largest

observation or up to 1.5 times the difference between upper hinge and median—whichever is smaller. The lower whisker is defined analogously. Small circles represent outliers (if any), that is, observations that fall beyond the whiskers. Notches extend to $\pm 1.58\,\mathrm{IQR}/\sqrt{n}$, where IQR is the interquartile range and $n = 10$ is the number of observations. Notches indicate the 95 % confidence interval on the position of the median. If the notches of two boxes do not overlap, the observed difference between the respective medians is significant (Chambers et al. 1983).

In the boxplots, we include also the results obtained in simulation in order to appraise the impact of the reality gap on the four design methods. Results obtained with robots are represented by wide boxes and those obtained in simulation by narrow boxes. As with the assessment performed with the e-puck robots, also in simulation we perform 10 independent runs for the control software instance generated by each of the four design methods under analysis.

4.2 Per-task results

We report in the following the results obtained by the methods under analysis on each of the five tasks. The notched box-and-whisker boxplots are given in Fig. 4. Videos of the test runs and the complete data are available as online supplementary material (Francesca et al. 2014a).

SCA–shelter with constrained access. `C-Human` and `Vanilla` perform better than `U-Human` and `EvoStick`. In particular, `Vanilla` is significantly better than `EvoStick`. `EvoStick` is unable to overcome the reality gap. This holds true also for `U-Human`, but to a far minor extent. `C-Human` and `Vanilla` overcome the reality gap satisfactorily.

LCN–largest covering network. `C-Human` and `EvoStick` perform better than other methods, with `Vanilla` performing significantly better than `U-Human`. `Vanilla` and `U-Human` do not successfully overcome the reality gap.

CFA–coverage with forbidden areas. The performance of the four methods is comparable: Differences are within a range of few centimeters, that is, less than half of the e-puck's diameter. Regarding the reality gap, all methods display differences between simulation and reality, but these differences are small in absolute terms—they are all within few centimeters.

SPC–surface and perimeter coverage. `EvoStick` is visibly unable to overcome the reality gap and performs significantly worse than the other methods. In the boxplot, the four `X` indicate four runs that resulted in a major failure. `Vanilla`, `U-Human`, and `C-Human` perform comparably well.

AAC–aggregation with ambient cues. `Vanilla`'s results are slightly better than those of `U-Human` and `C-Human`, and significantly better than those of `EvoStick`. The four methods greatly differ in their ability to overcome the reality gap: `EvoStick` has the most severe difficulties, followed by `U-Human`; also `Vanilla` and `C-Human` display difficulties, but to a minor extent.

4.3 Aggregate analysis and discussion

The aggregate analysis is given in Fig. 5: `C-Human` performs significantly better than `Vanilla`, and both `C-Human` and `Vanilla` perform significantly better than `EvoStick` and `U-Human`. The comparison between `Vanilla` and `EvoStick` confirms the results obtained by Francesca et al. (2014c).

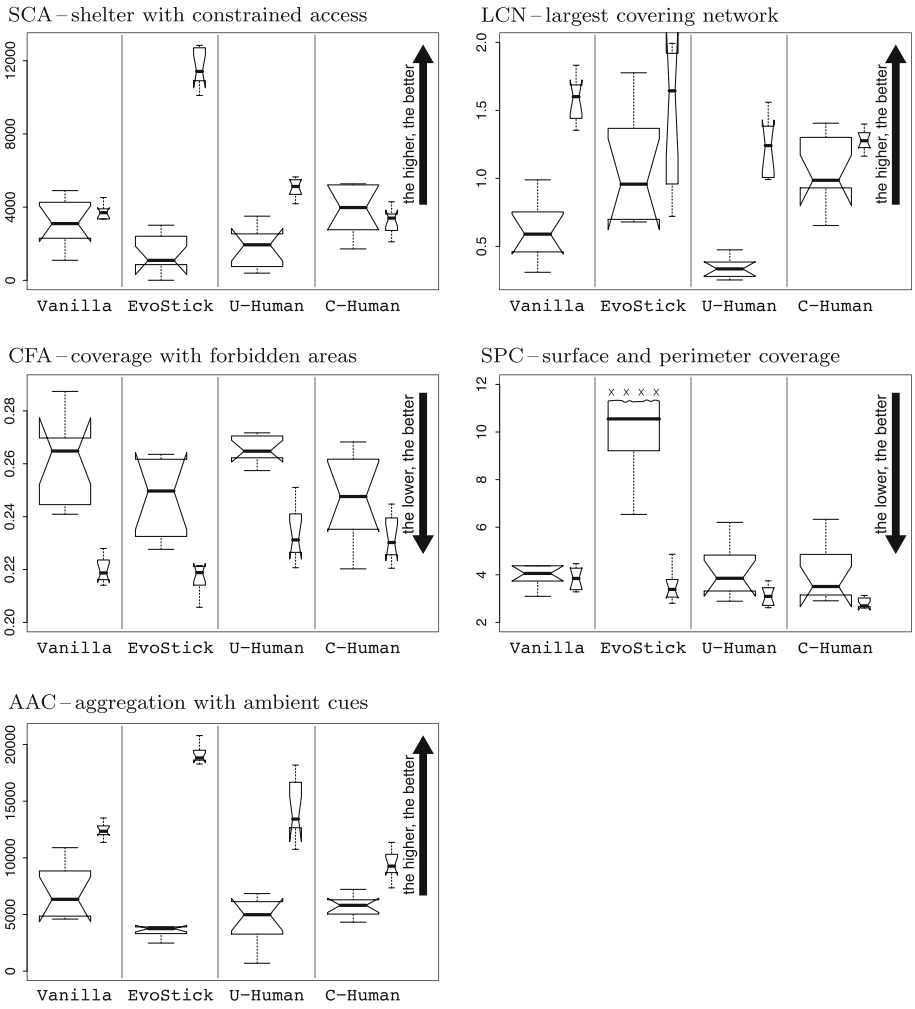


Fig. 4 Study A—notched box-and-whisker boxplots of the results obtained by `Vanilla`, `EvoStick`, `U-Human`, and `C-Human` on the five tasks. In each boxplot, the *vertical axis* reports the values of the task-specific performance measure. *Wide boxes* represent the results obtained with the robots and *narrow boxes* those obtained in simulation. See Sect. 4.1.5 for a guide on how to read notched box-and-whisker boxplots

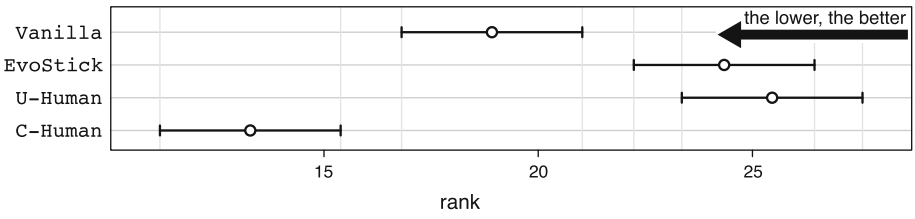


Fig. 5 Study A—Friedman test on aggregate data from the five tasks. `Vanilla` performs significantly better than `EvoStick` and `U-Human`, but significantly worse than `C-Human`. See Sect. 4.1.5 for an explanation of how to read the plot

Concerning manual design, some interesting facts can be observed. The results show that, at least under the specification–development–deployment life cycle, also manual design suffers from the reality gap. It is interesting to notice that C-Human appears to be more effective than U-Human at overcoming the reality gap. This confirms the reasoning based on the notion of bias-variance trade-off (Geman et al. 1992) that we presented in Francesca et al. (2014c): The reality gap problem is an instance of a wider problem related to generalization. Methods that are more constrained and therefore have a reduced representational power also have a reduced variance. As a result, they appear to have better generalization properties.

A second interesting fact is that, under the protocol adopted, human designers produce more effective control software when constrained to use the few and relatively simple predefined modules of Vanilla. This result was unexpected—particularly by the experts themselves—and appears counterintuitive. We were convinced that human designers would have obtained the best results when left free to structure the control software following their intuition and understanding. Also, we expected that the constraint to limit the design activity to assembling 12 predefined modules would have hindered their creativity and would have limited their effectiveness. The results proved us wrong: C-Human clearly outperforms U-Human. Apparently, the aforementioned reduction in the variance that is obtained by introducing constraints more than compensates for the disadvantages that derive from the introduction of a bias.

Concerning the comparison between manual and automatic design, the results show that Vanilla performs significantly better than U-Human but worse than C-Human. This is a promising result but indicates that we have not attained our goal yet: Vanilla cannot be claimed to be more effective than a human designer.

The results obtained by C-Human and Vanilla corroborate the hypothesis that Vanilla's set of modules are generally appropriate for tackling relevant tasks with robots conforming to RM1. The results also highlight a major issue with Vanilla: The optimization algorithm F-Race appears to be unable to fully exploit the potential of the modules. This is clearly indicated by the fact that C-Human, which adopts the same modules adopted by Vanilla, performs significantly better.

5 Chocolate

Chocolate is an improved version of Vanilla. As Vanilla, Chocolate designs control software for RM1. Chocolate differs from Vanilla in a single aspect: the optimization algorithm used to explore the design space. Chocolate adopts Iterated F-Race (Balaprakash et al. 2007; Birattari et al. 2010; López-Ibáñez et al. 2011), an algorithm for automatic configuration originally devised to fine-tune the parameters of metaheuristics. Iterated F-Race is based on F-Race (Birattari et al. 2002; Birattari 2009), the optimization algorithm adopted in Vanilla.

In Iterated F-Race, the optimization process goes through a series of iterations each of which is an execution of the F-Race algorithm. In the first iteration, an initial set of candidate solutions is generated by sampling the space of feasible solutions in a uniformly random way. The initial candidates undergo a first execution of the F-Race algorithm. When the F-Race algorithm terminates, the surviving solutions—that is, the candidate solutions that have not been discarded—are used as a seed to generate a new set of candidate solutions on which the following iteration will operate. The new set of candidates is obtained by sampling the space of feasible solutions according to a distribution that gives a higher probability of

being selected to solutions that are close to the surviving solutions. See López-Ibáñez et al. (2011) for the details. The new set of candidates undergoes a further execution of the F-Race algorithm. The process is iterated and stops when a predefined budget of evaluations have been performed.

The implementation of Iterated F-Race that is adopted in `Chocolate` is the one provided by the *irace* package (López-Ibáñez et al. 2011) for R (R Core Team 2014). `Chocolate` uses the default parameters of *irace* and samples the design space using the built-in sampling procedure of *irace* (López-Ibáñez et al. 2011).

Our working hypotheses are that, by adopting a more effective optimization algorithm, (i) `Chocolate` improves over `Vanilla` and, most importantly, (ii) the improvement is such that `Chocolate` outperforms `C-Human`.

6 Study B: assessment of Chocolate

The goal of this study is to empirically assess `Chocolate` and corroborate the working hypotheses formulated in Sect. 5.

6.1 Experimental protocol

The experimental protocol that we adopt to evaluate `Chocolate` shares its main features with the one defined in Sect. 4.1. The only differences concern (i) the way in which the tasks are defined/selected and (ii) the design methods under analysis.

6.1.1 Tasks and design methods under analysis

The study is performed on the five swarm robotics tasks considered in Sect. 4. We do not perform again the procedure described in Sect. 4.1.1, but we directly adopt the task defined by the experts for Study A and already described in Sect. 4.1.2

We compare `Chocolate`, with `Vanilla`, and `C-Human`. `EvoStick` and `U-Human` have been excluded from this study because they were clearly outperformed by `C-Human` and `Vanilla` in Study A. Concerning `Vanilla` and `C-Human`, we adopt the same control software generated in Study A.

`Chocolate`, `Vanilla`, and `C-Human` share a number of key characteristics: (i) They all produce robot control software in the form of a probabilistic finite state machine; (ii) they operate on the same set of modules; and (iii) they all adopt the same simulator to compare and select candidate designs.

The three design methods under analysis are tested under the same conditions adopted in Study A: same platform, same simulator, same performance measures, same resources. See Sect. 4.1.3 for the details.

6.1.2 Assessment on a swarm of e-pucks

As in Study A, the control software produced by the design methods are assessed via test runs with a swarm of 20 e-puck robots.

Although in Sect. 4 we have already performed an assessment of the control software generated by `Vanilla` and `C-Human` on the five tasks considered, we repeat the assessment here. This allows us to compare the three methods under the exact same conditions. It would be indeed practically impossible to reproduce the same conditions under which Study A

was performed: Robots, batteries, and light sources have been subject to wear and their properties have possibly changed. Moreover, we have updated the firmware of the e-pucks and the software we use to manage the e-pucks and to track their position over time. Finally, the arena has been disassembled and later reassembled in a different position.

We adopt here the same hands-off approach we adopted in Study A: The control software is cross-compiled by ARGoS and it is uploaded onto each e-puck of the swarm without any modification. Moreover, the order of the test runs is randomly generated so that runs with the control software produced by the methods under analysis are interleaved. The initial positions of the e-pucks are randomly generated as in Study A. Also the performance measures are computed via the tracking system as in Study A.

6.1.3 Objective of the study and statistical tools

The objective of the study is to compare the three design methods. In particular, we wish to confirm our working hypotheses: (i) `Chocolate` improves over `Vanilla`; and, most of all, (ii) the improvement is such that `Chocolate` outperforms `C-Human`. As in Study A, we are interested in the aggregate performance of the methods over the five tasks. Therefore, the main statistical tool we use is the Friedman test. For completeness, we report also the per-task notched box-and-whisker boxplots.

6.2 Per-task results

The results obtained by the three methods under analysis on each of the five tasks are represented by the notched box-and-whisker boxplots reported in Fig. 6. Videos of the test runs and the complete data are available as online supplementary material (Francesca et al. 2014a).

SCA – shelter with constrained access. The control software instance designed by `Chocolate` performs better than the ones designed by `Vanilla` and `C-Human`. The differences in performance between `Chocolate` and `C-Human` and between `Chocolate` and `Vanilla` are significant. Moreover, `C-Human` performs significantly better than `Vanilla`. Regarding the difference between simulation and reality, `Chocolate` and `C-Human` appear to overcome the reality gap successfully: They have similar performance in simulation and in reality. On the contrary, `Vanilla` shows a significant mismatch.

LCN – largest covering network. `Chocolate` and `C-Human` have qualitatively the same performance. On the other hand, `Vanilla` performs significantly worse than both `Chocolate` and `C-Human`. For what concerns the effects of the reality gap, all three design methods present a rather noticeable difference between simulation and reality. `C-Human` displays the smallest mismatch. The mismatch between the performance in simulation and reality is possibly due to the fact that, to solve this task, the e-pucks rely on their ability to measure the distance of the neighboring robots. The measurement of the distance is obtained via the range-and-bearing board, which is imprecise and highly dependent on uncontrolled factors such as battery levels and light conditions.

CFA – coverage with forbidden areas. `Chocolate` and `C-Human` have similar performance. `Vanilla` is slightly worse. Differences are small: Medians are all within a range of less than two centimeters. Regarding the reality gap, the three methods present a similar difference between simulation and reality. The flattening of the results with small differences that have no practical implications is possibly due to the imprecision of the distances measured via the range-and-bearing board.

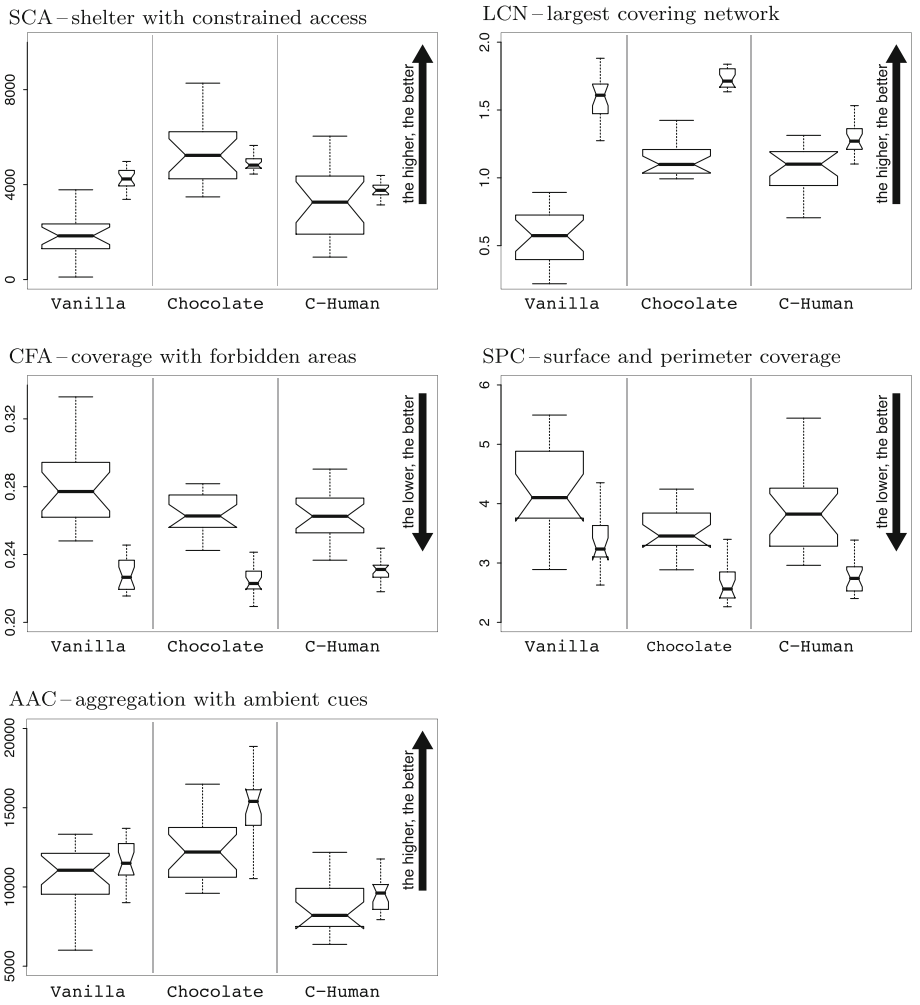


Fig. 6 Study B—notched box-and-whisker boxplots of the results obtained by `Vanilla`, `Chocolate`, and `C-Human` on the five tasks. In each boxplot, the *vertical axis* reports the values of the task-specific performance measure. *Wide boxes* represent the results obtained with the robots and *narrow boxes* those obtained in simulation. See Sect. 4.1.5 for a guide on how to read notched box-and-whisker boxplots

SPC – surface and perimeter coverage. The median performance recorded for `Chocolate` is better than the one recorded for `C-Human` and `Vanilla`. The difference between `Chocolate` and `Vanilla` is significant. Concerning the difference between the performance observed in simulation and on the robots, all three the methods show some mismatch. Like in the case of SCA – shelter with constrained access, this difference is possibly due to the fact that part of the task relies on the imprecise estimation of the distance provided by the range-and-bearing board.

AAC – aggregation with ambient cues. Both `Chocolate` and `Vanilla` perform significantly better than `C-Human`. The median recorded for `Chocolate` is slightly better than the one recorded for `Vanilla`. Concerning the difference between the performance

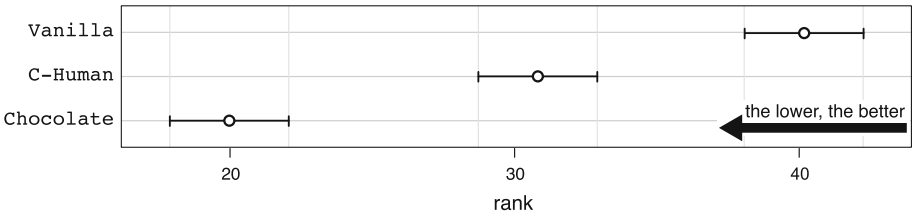


Fig. 7 Study B—Friedman test on aggregate data from the five tasks. `Chocolate` performs significantly better than both `Vanilla` and `C-Human`. See Sect. 4.1.5 for an explanation of how to read the plot

observed in simulation and on the robots, `Vanilla` shows a smaller difference with respect to `Chocolate` and `C-Human`.

6.3 Aggregate analysis and discussion

The results of the aggregate analysis are reported in Fig. 7. These results confirm those presented in Sect. 4: `C-Human` outperforms `Vanilla`. The better performance of `C-Human` over `Vanilla` corroborates the hypothesis formulated in Sect. 4.3: As `C-Human` and `Vanilla` design control software combining the same modules, the failure of `Vanilla` to match `C-Human`'s performance is to be ascribed to `Vanilla`'s optimization algorithm. The hypothesis is confirmed by the fact that `Chocolate`, which adopts a more advanced optimization algorithm, outperforms `Vanilla`. This improvement of `Chocolate` is such that, under the experimental conditions considered in the study, `Chocolate` outperforms also `C-Human`. In other words, under the experimental conditions defined by the protocol presented in Sect. 6.1, `Chocolate` produces better control software than the one produced by a human designer that operates on the same set of modules as `Chocolate`.

The results presented in Sect. 4 show that `C-Human` outperforms `U-Human`, that is, the human designer that produces control software without any restriction on the structure of the control software. Together, the results presented in Sect. 4 and those presented here lead to a stronger statement: Under the experimental conditions defined in Sect. 4.1 and Sect. 6.1, `Chocolate` designs control software that outperforms the one produced by a human designer, whether the human is constrained to use `Chocolate`'s (and `Vanilla`'s) modules or not.

7 Conclusions

In this paper, we presented two empirical studies on the design of control software for robot swarms in which automatic and manual methods are compared. In Study A, we compared two automatic methods—`Vanilla` and `EvoStick`—with two manual methods—`U-Human` and `C-Human`. `Vanilla` produces control software by assembling preexisting modules into a finite state machine. `EvoStick` is a rather standard evolutionary robotics method. The two automatic methods have been already published in Francesca et al. (2014c) and have been applied in this paper without any modification whatsoever. We carried out the comparison on five new tasks, different from those on which `Vanilla` and `EvoStick` had been previously tested. The tasks have been defined by human experts that, at the time of the definition of the tasks, were unaware of the functioning of `Vanilla` and `EvoStick`. The results showed that `Vanilla` produces better control software than `EvoStick`, which confirms the results

previously obtained on other tasks. Moreover, the results showed that Vanilla outperforms U-Human but is outperformed by C-Human. As C-Human is a method in which a human manually synthesizes finite state machines by assembling the same modules on which Vanilla operates, we concluded that the difference in performance between Vanilla and C-Human has to be ascribed to shortcomings in Vanilla's optimization algorithm.

To confirm our hypothesis, we defined a new automatic design method, Chocolate, that differs from Vanilla only in the optimization algorithm adopted. We assessed Chocolate in Study B: Our results show that, under the specific experimental conditions considered in the study, Chocolate outperforms both Vanilla and C-Human. Chocolate is the first automatic design method for robot swarms that is shown to outperform a human designer, albeit under specific experimental conditions.

Beside proposing and assessing Chocolate, in this paper we make an important contribution to the literature on the automatic design methods for robot swarms. We proposed and demonstrated an experimental protocol to compare automatic and manual design methods. The protocol has been developed for use with up to four methods and five experts. Nonetheless, the protocol can be straightforwardly extended to a larger number of methods and experts.

A notable trait of the empirical studies presented in the paper is the fact that all the design methods under analysis adopt the same reference model RM1. It is our contention that the definition of a reference model of the robotic platform at hand is a fundamental and necessary step to enable the fair and meaningful comparison of multiple design methods, whether manual or automatic. We argue that the lack of scientifically sound comparisons between different design methods in the swarm robotics literature is to some extent related to the fact that the important role of reference models has been so far overlooked.

An interesting question that naturally arises concerns the generality of the automatic design methods discussed in the paper. The question should be addressed at two different levels: (i) the generality of Vanilla and Chocolate, as specializations of AutoMoDe to a given reference model; and (ii) the generality of AutoMoDe as an approach. In Francesca et al. (2014c), we have already conceptually framed the notion of specialization of AutoMoDe and the generality of a specialization. In the following, we focus on original remarks that can be made on the basis of the results presented in this paper. Regarding the generality of Vanilla and Chocolate, the results presented in this paper corroborate the hypothesis that underlies the definition of Vanilla (Francesca et al. 2014c): The modules originally proposed for Vanilla and then adopted by Chocolate and C-Human are sufficiently general to produce control software for tasks that can be accomplished by a swarm of robots conforming to RM1. The hypothesis is corroborated by the results because of the way in which the tasks have been defined: As already noted, they can be considered as sampled according to an unknown distribution defined over the space of tasks that can be accomplished by a swarm of robots conforming to RM1. Regarding AutoMoDe, statements on the general applicability of the approach can be made only by specializing AutoMoDe to a large number of different reference models (of the same or different robots) and then assessing these specializations on multiple tasks via studies similar to those presented in this paper. This is clearly a research program that requires a large amount of empirical work and that goes far beyond the possibilities of a single paper.

A similar reasoning applies to the adoption of Iterated F-Race within AutoMoDe. As the results obtained by Iterated F-Race in Chocolate are fully satisfactory, Iterated F-Race will likely be the first optimization algorithm that we will consider in the specialization of AutoMoDe for new reference models. Nonetheless, it is perfectly possible that a new reference model (and therefore a new set of modules) requires adopting another optimization algorithm.

It should be noted that the optimization algorithm to be used is not part of the definition of AutoMoDe, but rather of its specializations. Whether Iterated F-Race scales satisfactorily with the number of modules and whether some characteristics of the modules create particular problems to Iterated F-Race is an empirical question that can be addressed only by specializing AutoMoDe to a large number of reference models that require an increasingly larger set of modules. Also in this case, this research program clearly goes beyond the possibility of a single paper.

A limitation of the study presented in the paper is related to the size and the composition of the group of experts. Five researchers, all affiliated with the same research group, cannot be considered as a sufficiently large and representative sample of the entire population of swarm robotics experts. As a consequence, the results presented in the paper do not generalize to the entire population of swarm robotics experts. These results show only that `Chocolate` was able, under the specific experimental conditions considered in the paper, to outperform a group of qualified domain experts, although these experts are not representative of the whole population. Nonetheless, this paper presents the first empirical study in which an automatic method for the design of control software for robot swarms has been shown to outperform human designers under controlled experimental conditions.

Future work will be devoted to the development of a new specialization of AutoMoDe to target a reference model that is more complex than RM1. In particular, we wish to explore the possibility of specializing AutoMoDe for a reference model that includes communication capabilities. We also plan on assessing the new specialization of AutoMoDe in an empirical study that involves a large number of human experts from different research groups. We will consider also the possibility of having teams of designers working on a task. Finally, we will study the impact of the amount of design time on the quality of the solutions obtained both by humans and by automatic methods.

Acknowledgments The authors thank Maria Zampetti and Maxime Bussios for their help with the robots, and Marco Chiarandini for his implementation of the Friedman test. This research was partially funded by the ERC Advanced Grant "E-SWARM" (Grant Agreement No. 246939). It was also partially supported by the COMEX project within the Interuniversity Attraction Poles Program of the Belgian Science Policy Office. Vito Trianni acknowledges support from the Italian CNR. Arne Brutschy, Franco Mascia, and Mauro Birattari acknowledge support from the Belgian F.R.S.–FNRS. The authors thank the anonymous reviewers for their useful comments.

References

Ampatzis, C., Tuci, E., Trianni, V., Christensen, A. L., & Dorigo, M. (2009). Evolving self-assembly in autonomous homogeneous robots: Experiments with two physical robots. *Artificial Life*, *15*(4), 465–484.

Balaprakash, P., Birattari, M., & Stützle, T. (2007). Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement. *Hybrid metaheuristics, HM 2007, LNCS* (Vol. 4771, pp. 108–122). Berlin: Springer.

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. *International conference on robotics and automation, ICRA 2011* (pp. 378–385). Piscataway, NJ: IEEE Press.

Birattari, M. (2009). *Tuning metaheuristics*. Berlin: Springer.

Birattari, M., Stützle, T., Paquete, L., & Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. In W. B. Langdon, et al. (Eds.), *Genetic and evolutionary computation, GECCO* (pp. 11–18). San Francisco, CA: Morgan Kaufmann.

Birattari, M., Yuan, Z., Balaprakash, P., & Stützle, T. (2010). F-race and iterated F-race: An overview. In T. Bartz-Beielstein, et al. (Eds.), *Experimental methods for the analysis of optimization algorithms* (pp. 311–336). Berlin: Springer.

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

Brambilla, M., Brutschy, A., Dorigo, M., & Birattari, M. (2014). Property-driven design for swarm robotics: A design method based on prescriptive modeling and model checking. *ACM Transactions on Autonomous and Adaptive Systems*, *9*(4), 17.1–28.

Bredeche, N., Montanier, J. M., Liu, W., & Winfield, A. F. (2012). Environment-driven distributed evolutionary adaptation in a population of autonomous robotic agents. *Mathematical and Computer Modelling of Dynamical Systems*, *18*(1), 101–129.

Chambers, J. M., Cleveland, W. S., Kleiner, B., & Tukey, P. A. (1983). *Graphical methods for data analysis*. Pacific Grove, CA: Wadsworth & Brooks/Cole.

Conover, W. J. (1999). *Practical nonparametric statistics*. New York: Wiley.

Di Mario, E., & Martinoli, A. (2014). Distributed particle swarm optimization for limited-time adaptation with real robots. *Robotica*, *32*(2), 193–208.

Doncieux, S., & Mouret, J. B. (2014). Beyond black-box optimization: A review of selective pressures for evolutionary robotics. *Evolutionary Intelligence*, *7*(2), 71–93.

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

Duarte, M., Oliveira, S. M., & Christensen, A. L. (2014a). Evolution of hierarchical controllers for multirobot systems. In H. Sayama, et al. (Eds.), *Artificial life 14: Proceedings of the international conference on the synthesis and simulation of living systems* (pp. 657–664). Cambridge, MA: MIT Press.

Duarte, M., Oliveira, S. M., & Christensen, A. L. (2014b). Evolution of hybrid robotic controllers for complex tasks. *Journal of Intelligent & Robotic Systems*, *78*(3–4), 463–484.

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, LNAI* (Vol. 7426, pp. 381–390). Berlin: Springer.

Francesca, G., Brambilla, M., Brutschy, A., Garattoni, L., Miletitch, R., Podevijn, G., et al. (2014a). AutoMoDe-chocolate: Automatic design of control software for robot swarms. Supplementary material. http://iridia.ulb.ac.be/supp/IridiaSupp2014-011.

Francesca, G., Brambilla, M., Brutschy, A., Garattoni, L., Miletitch, R., Podevijn, G., et al. (2014b). An experiment in automatic design of robot swarms. In M. Dorigo, et al. (Eds.), *Swarm intelligence, ANTS 2014, LNCS* (Vol. 8667, pp. 25–37). Berlin: Springer.

Francesca, G., Brambilla, M., Brutschy, A., Trianni, V., & Birattari, M. (2014c). AutoMoDe: A novel approach to the automatic design of control software for robot swarms. *Swarm Intelligence*, *8*(2), 89–112.

Gauci, M., Chen, J., Li, W., Dodd, T. J., & Groß, R. (2014a). Clustering objects with robots that do not compute. In A. Lomuscio, et al. (Eds.), *Autonomous agents and multiagent systems, AAMAS 2014* (pp. 421–428). SC: IFAAMAS, Richland.

Gauci, M., Chen, J., Li, W., Dodd, T. J., & Groß, R. (2014b). Self-organized aggregation without computation. *International Journal of Robotics Research*, *33*(8), 1145–1161.

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

Gomes, J., Urbano, P., & Christensen, A. L. (2013). Evolution of swarm robotics systems with novelty search. *Swarm Intelligence*, *7*(2–3), 115–144.

Groß, R., & Dorigo, M. (2009). Towards group transport by swarms of robots. *International Journal of Bio-Inspired Computation*, *1*(1–2), 1–13.

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

Haasdijk, E., Bredeche, N., & Eiben, A. E. (2014). Combining environment-driven adaptation and task-driven optimisation in evolutionary robotics. *PLoS ONE*, *9*(6), e98,466.

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

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.

Hecker, J. P., Letendre, K., Stolleis, K., Washington, D., & Moses, M. E. (2012) Formica ex machina: Ant swarm foraging from physical to virtual and back again. In Dorigo M, et al. (Eds.), *Swarm intelligence, ANTS 2012, LNCS* (Vol. 7461, pp. 252–259). Berlin: Springer.

Hsieh, M., Loizou, S., & Kumar, V. (2007). Stabilization of multiple robots on stable orbits via local sensing. In *International conference on robotics and automation, ICRA 2007* (pp. 2312–2317). Piscataway, NJ: IEEE Press.

Izzo, D., Simões, L. F., & de Croon, G. C. H. E. (2014). An evolutionary robotics approach for the distributed control of satellite formations. *Evolutionary Intelligence*, *7*(2), 107–118.

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, LNCS* (Vol. 929, pp. 704–720). Berlin: Springer.

Kazadi, S., Lee, J. R., & Lee, J. (2007). Artificial physics, swarm engineering, and the Hamiltonian method. In *World congress on engineering and computer science* (pp. 623–632). Hong Kong: Newswood.

König, L., & Mostaghim, S. (2009). Decentralized evolution of robotic behavior using finite state machines. *International Journal of Intelligent Computing and Cybernetics*, *2*(4), 695–723.

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.

Lehman, J., & Stanley, K. O. (2011). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, *19*(2), 189–223.

Lerman, K., & Galstyan, A. (2002). Mathematical model of foraging in a group of robots: Effect of interference. *Autonomous Robots*, *13*(2), 127–141.

Lerman, K., Galstyan, A., Martinoli, A., & Ijspeert, A. J. (2001). A macroscopic analytical model of collaboration in distributed robotic systems. *Artificial Life*, *7*(4), 375–393.

Lopes, Y., Leal, A., Dodd, T. J., & Groß, R. (2014). Application of supervisory control theory to swarms of e-puck and kilobot robots. In M. Dorigo, et al. (Eds.), *Swarm Intelligence, ANTS 2014, LNCS* (Vol. 8667, pp. 62–73). Berlin: 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, Belgium.

Martinoli, A., Ijspeert, A. J., & Mondada, F. (1999). Understanding collective aggregation mechanisms: From probabilistic modelling to experiments with real robots. *Robotics and Autonomous Systems*, *29*(1), 51–63.

Martinoli, A., Easton, K., & Agassounon, W. (2004). Modeling swarm robotic systems: A case study in collaborative distributed manipulation. *The International Journal of Robotics Research*, *23*(4–5), 415–436.

Mondada, F., Franzi, E., & Ienne, P. (1993). Mobile robot miniaturization: A tool for investigation in control algorithms. In T. Yoshikawa & F. Miyazaki (Eds.), *Experimental robotics III* (pp. 501–513). Berlin, Germany: Springer.

Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., et al. (2009). The e-puck, a robot designed for education in engineering. In *9th Conference on autonomous robot systems and competitions*, Instituto Politécnico de Castelo Branco, Portugal, (pp. 59–65).

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.

Pugh, J., & Martinoli, A. (2009). Distributed scalable multi-robot learning using particle swarm optimization. *Swarm Intelligence*, *3*(3), 203–222.

Quinn, M., Smith, L., Mayley, G., & Husbands, P. (2003). Evolving controllers for a homogeneous system of physical robots: Structured cooperation with minimal sensors. *Philosophical Transactions of the Royal Society of London, Series A: Mathematical, Physical and Engineering Sciences*, *361*(1811), 2321–2343.

R Core Team. (2014). *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing.

Rubenstein, M., Ahler, C., Hoff, N., Cabrera, A., & Nagpal, R. (2014). Kilobot: A low cost robot with scalable operations designed for collective behaviors. *Robotics and Autonomous Systems*, *62*(7), 966–975.

Sartoretti, G., Hongler, M. O., de Oliveira, M. E., & Mondada, F. (2014). Decentralized self-selection of swarm trajectories: From dynamical systems theory to robotic implementation. *Swarm Intelligence*, *8*(4), 329–351.

Stranieri, A., Turgut, A., Salvaro, M., Garattoni, L., Francesca, G., Reina, A., et al. (2013). IRIDIA's arena tracking system. Technical report TR/IRIDIA/2013-013, IRIDIA, Université Libre de Bruxelles, Belgium.

Trianni, V. (2008). *Evolutionary swarm robotics*. Berlin: Springer.

Trianni, V. (2014). Evolutionary robotics: Model or design? *Frontiers in Robotics and AI*, *1*(13), 1–6.

Trianni, V., & López-Ibáñez, M. (2014). Advantages of multi-objective optimisation in evolutionary robotics: Survey and case studies. Technical report TR/IRIDIA/2014-014, IRIDIA, Université Libre de Bruxelles, Belgium.

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., Ferrauto, T., Zeschel, A., Massera, G., & Nolfi, S. (2011). An experiment on behavior generalization and the emergence of linguistic compositionality in evolving robots. *IEEE Transactions on Autonomous Mental Development*, *3*(2), 176–189.

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

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.