AutoMoDe: a modular approach to the automatic off-line design and fine-tuning of control software for robot swarms[*]

Mauro Birattari, Antoine Ligot, and Gianpiero Francesca

Abstract Although swarm robotics is widely recognized as a promising approach to coordinating large groups of robots, a general methodology for designing collective behaviors for robot swarms is still missing. Automatic off-line design is an appealing solution but it is prone to the so called reality gap, which is the reason for performance drops when control software developed in simulation is deployed on real robots. We present here our research on AutoMoDe, a novel approach to the automatic off-line design of robot swarms, which is based on the principle of modularity. AutoMoDe produces control software for robot swarms by selecting, combining, instantiating, and fine-tuning predefined parametric modules that represent low-level behaviors defined in a mission-agnostic way. By restricting the generation of control software to the instances that can be produced with the given modules, we effectively inject a bias in the design process and we consequently reduce its variance. As confirmed by the empirical studies realized so far, this reduces the risk of overfitting simulation models and improves the chances of crossing the reality gap successfully.

Mauro Birattari
Université libre de Bruxelles, Brussels, Belgium, e-mail: mbiro@ulb.ac.be

Antoine Ligot
Université libre de Bruxelles, Brussels, Belgium, e-mail: aligot@ulb.ac.be

Gianpiero Francesca
Toyota Motor Europe, Brussels, Belgium, e-mail: gianpiero.francesca@toyota-europe.com

[*] The three authors equally contributed to the realization of this chapters and should be considered as co-first authors. The research presented has been conceived and directed by MB; the core experiments were performed by GF; the manuscript was drafted by AL, revised by MB, and optimized by the three authors. The research has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 681872). MB acknowledges support from the Belgian *Fonds de la Recherche Scientifique*–FNRS, of which he is a Research Director.

1 An introduction to swarm robotics and its design problem

Swarm robotics [26] is an engineering discipline that found inspiration in swarm intelligence [25], and that is now recognized as a promising approach to design large groups of autonomous robots [86]. Although this discipline has attained a notable position in the scientific literature [68, 82, 35, 73, 87, 52, 85], it has yet to be applied to a real world scenario. It is the difficulty of reliably generating a desired collective behavior, and more specifically the lack of a general methodology to do so, that hinders the application of the principles of swarm robotics to the real world [16]. In a swarm, robots are completely autonomous and act on the basis of the principle of locality: they take decisions based solely on local information collected through their own sensors, or on information communicated by their neighboring peers. Any collective behavior displayed by a swarm is the result of interactions between robots, and between robots and the environment. The design problem in swarm robotics is particularly challenging as it is not feasible to directly program the desired collective behavior of a swarm: only the individual robot behaviors can be specified. Obtaining the desired collective behavior requires therefore to master the complex "what, where, when, and how" of the many robot-robot and robot-environment interactions that characterize the operation of a swarm.

Traditional multi-robot systems and software engineering techniques [20, 24, 14, 70], which rely on the formal derivation of the individual behaviors from specifications expressed at the collective level, cannot be applied to swarm robotics, at least in the general case, due to the aforementioned issues. A few principled manual design methods have been proposed [41, 46, 7, 5, 15, 66, 56, 63], but their application is limited to specific classes of missions due to their working hypotheses and constraints. Therefore, experts in swarm robotics usually proceed by trial and error to obtain the desired collective behaviors

A promising alternative to manually designing the control software exists: the adoption of *optimization-based* design methods. With such methods, the design problem becomes an optimization problem: an optimization algorithm explores the search space composed of all possible individual behaviors, with the objective of finding one that maximizes a performance measure expressed at the collective level. The optimization-based approach regroups different categories of design methods. In the domain literature, the commonly adopted classification distinguishes between on-line and off-line methods [16, 30, 17]. A second classification, orthogonal to the on-line/off-line one, distinguishes between semi-automatic and (fully-)automatic design methods [10].

In on-line methods, the design process is distributed and operates on the physical robots while they perform their mission [81, 51, 47, 18, 40, 72]. Although promising for specific circumstances—for example, to adjust the parameters of control software—on-line methods do not appear to be the ultimate solution to the design problem in swarm robotics as their application is limited to cases in which the robots are able to evaluate their collective performance [30]. In off-line methods, the design process is performed before the swarm is deployed in the target environment and relies on computer-based simulations. As the assessment of control software is performed in simulation, which allows the computation of any desired performance measure, off-line methods can potentially be applied to any mission. However, they are faced with a problem that does not affect on-line methods: the so-called *reality gap* [19, 45], which refers to the difference between simulation and reality. Due to the reality gap, one should expect to be deceived by the performance of automatically generated control software when it is deployed on physical robots, as it is likely to drop in comparison with the one obtained in simulation [29].

In semi-automatic design, a human designer utilizes an optimization algorithm as a tool that they operate using their intuition and previous experience. Typically, the designer iterates through a series of steps, which include: the execution of the optimization process, the evaluation and analysis of the behavior produced using simulation and/or physical robot experiments, and the modification of the optimization process. The designer modifies the optimization process so that, on basis of the evaluation and according to their experience, more performing control software will be produced in the following execution. The elements of the design process that are often modified include the parameters of the optimization algorithm, the characteristics of the control software architecture, or the performance measure to be optimized. This three-step procedure is repeated until the control software produced satisfies the designer and/or they feel that it cannot be improved any further. Many studies have shown that semi-automatic design is an effective way to design robot swarms [65, 22, 4, 58, 75, 44, 78, 79, 80, 83, 27, 38, 28, 77]. The drawback of this approach is that it involves an expert designer that must have a good understanding of the optimization process and of the mission at hand. Moreover, because the results obtained are to be partially credited to the expert's ingenuity, they are often hardly reproducible.

In automatic design, one expects from a method to reliably produce control software for a whole class of missions without the need to apply modifications [10]. The optimization process is therefore performed in a fully automatic way, without the need of any per-mission human intervention.

In the rest of this chapter, we take a close look at a novel approach to the off-line automatic design: AutoMoDe, short for *automatic modular design* [33]. The main characteristic of this approach is the modularity of the instances of control software it generates: they are obtained by an optimization algorithm that automatically selects, combines, and fine-tunes predefined modules. These modules include actions to be performed by an individual robot, and conditions on the environment perceived or the internal state of the robot that determine whether the robot should transition from one action to another. A number of automatic design methods have been proposed so far that belong in the AutoMoDe approach. In most of them, as we will see in the body of the chapter, the optimization algorithm adopted is either F-race [11, 8] or Iterated F-race [3, 12, 57]. These two algorithms, which were originally developed to automatically design and/or fine-tune metaheuristics, appear to be an ideal choice to handle the high uncertainty that characterizes the operation of a robot swarm. In Section 2, we elaborate on the reasoning and on the working hypothesis that led to the definition of AutoMoDe; in Section 3, we describe the different instantiations of AutoMoDe and discuss their achievements; and in

Section 4, we report additional experiments that were conducted to corroborate the working hypothesis discussed in Section 2. In Section 5, we summarize the main points made in this chapter and we highlight future research directions.

2 From neuro-evolutionary robotics to AutoMoDe

In swarm robotics, the most popular optimization-based approach for designing control software is neuro-evolutionary robotics [55, 29, 75, 76]. In this approach, robots are controlled by a neural network: sensor readings are fed to the neural network as inputs, whereas the robot actuator values are dictated by the network's output. An evolutionary algorithm [2] is used to search for the best possible configuration of the neural network. Thanks to the high flexibility and representational power of neural networks, the neuro-evolutionary approach can produce extremely varied and diverse behaviors [42, 61, 13]. The neuro-evolutionary approach has been successfully used to generate control software for various swarm robotics missions [65, 22, 44, 78]. However, when applied to the off-line case, neuro-evolutionary robotics presents a major limitation: it is unable to cross the reality gap reliably [71], that is, control software developed in simulation does not typically perform satisfactorily when ported to reality.

We conjecture that the inability of evolutionary robotics to cross the reality gap reliably is indeed a side effect of the high representational power of neural networks [33]. Our working hypothesis here is that the reality gap problem faced in the off-line automatic design of robot swarms is somehow reminiscent of the generalization problem faced in machine learning. In supervised learning, a fundamental result is the so-called *bias/variance tradeoff*, which states that the prediction error of an approximator can be decomposed into a bias and a variance component [37, 84]. It is known that these two components are correlated to the representational power of the approximator: a large representational power implies low bias and high variance. It is therefore our contention that, in the context of the automatic off-line design of robot swarms, the high representational power of neural networks can be counterproductive and be the cause of a sort of *overfitting* to the simulation environment, which then hinders performance in the real world.

Based on this reasoning, Francesca et al. introduced a novel approach to the automatic design of robot swarms: *AutoMoDe* [33]. In AutoMoDe, robots are controlled by a modular software architecture (e.g., a finite state machine) automatically generated by assembling predefined modules. Compared to the neural networks used in neuro-evolutionary robotics, the control software generated by AutoMoDe features a lower representational power. In AutoMoDe, the representational power is reduced by injecting bias in the generation of control software: the control software that can be produced is restricted to what can be obtained by assembling some predefined modules. AutoMoDe is a general framework that needs to be *specialized* to a specific robotic platform. To define a specialization of AutoMoDe, an expert needs to: (i) provide a set of modules for the given robotic platform, (ii) select an optimization algorithm, and (iii) define a control software architecture into which the modules must be assembled by the optimization algorithm. The optimization algorithm explores a search space composed of all possible instantiations and combinations of the available modules. It is important to stress that the modules are defined by the expert in a mission-agnostic way. In other terms, the modules are defined on the basis of the capabilities of the robots for then be used to generate control software for any possible mission of interest. The modules are defined once and for all: they are not supposed to be manually modified or tweaked to accommodate the needs of a specific mission for which control software must be designed. The fact that the produced control software is *a priori* constrained to be a combination of predefined modules introduces a bias and reduces the representational power: it limits the possibility to fine-tune the robot-robot and robot-environment interactions. However, assuming that the expert implements the modules correctly, a specialization of AutoMoDe should be able to generate a sufficient variety of behaviors so as to allow the swarm to perform the possible missions of interest.

The definition and implementation of the modules is critical to the success of a specialization of AutoMoDe. To better understand the goals and the implications of the definition of a specialization of AutoMoDe, the following conceptual representation of the issue can be helpful. A specialization of AutoMoDe—like any automatic design method—is implemented for a robotic platform on the basis of an abstraction of its characteristics and capabilities: what we call a *reference model*. Implicitly, a reference model defines also the class of missions that can be performed by a swarm of robots described by the reference model itself. For example, a mission that requires sorting objects according to their color cannot be performed by robots that are unable to distinguish colors. More formally, consider the class M_I comprising the behaviors that accomplish collective missions that are of interest (within a specific context). Consider also the class B_{RM} comprising all the behaviors that can be produced by a robot swarm whose individuals conform to the given reference model. The intersection between M_I and B_{RM} gives the class M of behaviors that accomplish collective missions that are of interest and that can be produced by robots conforming to the given reference model. A specialization of AutoMoDe and specifically the definition of the modules—like the implementation of any automatic design method—implicitly defines the class B' of the behaviors that can be produced by a swarm of robots whose control software is generated by the specialization of AutoMoDe itself—or by any other automatic design method one might consider. Clearly, B' is a subset of B_{RM}: a specialization of AutoMoDe cannot generate behaviors for a robotic platform that allow it to perform missions requiring capabilities (e.g., sensors and actuators) it does not have—and neither can any other automatic design method. What is important to notice is that the relationship between the classes B_{RM} and B' defines the representational power: the larger B', the higher the representational power. However, what is crucial is M', the intersection between the classes B' and M, which represents the behaviors that can be generated and that solve collective missions of interest. In an ideal case, a specialization of AutoMoDe—or more generally an automatic design method—can produce all the behaviors that solve collective missions of interest conforming to the reference

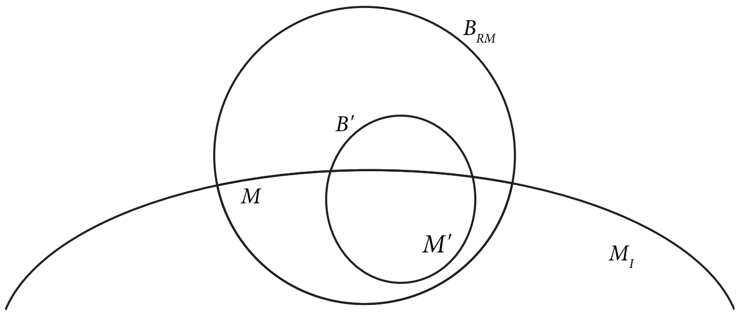

Fig. 1 Graphical representation of the sets of behaviors that are relevant when discussing the capabilities of an automatic design method. Note that these sets are purely conceptual and do not need to be explicitly defined in a closed form. M_I is the set of behaviors allowing a swarm to perform collective missions that are of interest (within a specific context). B_{RM} is the set of behaviors that can be achieved by a swarm of robots conforming to the reference model at hand. M is the set of behaviors that can be achieved by a swarm of robots conforming to the reference model and that accomplish collective missions of interest. B' is the set of behaviors that can be produced by a specialization of AutoMoDe—or more generally by a given design method. M' is the set of behaviors that can be produced by the specialization of AutoMoDe—or by a given design method—and that accomplish collective missions of interest.

model, that is $M \equiv M'$, and no behaviors that are irrelevant, that is $M \equiv M' \equiv B'$. Figure 1 gives a graphical representation of the different classes.

3 The specializations of AutoMoDe

Multiple specializations of AutoMoDe have been defined so far, all generating control software for an extended version of the e-puck robot [59, 36]. In the following, we only give a brief overview of these specializations and of the results obtained. We refer the reader to the original publications for the details.

The very first specialization of AutoMoDe was named `Vanilla` [33] and served as a proof-of-concept to assess the core ideas of AutoMoDe. `Vanilla` selects, combines, and fine-tunes predefined modules into control software in the form of a probabilistic finite state machine. In the finite state machine produced by `Vanilla`, the states are associated with low-level behavior modules, whereas the edges or transitions are associated with condition modules. `Vanilla` is based on a set of six low-level behaviors and a set of six conditions. The low-level behaviors are **stop**, which prevents the robot from moving; **exploration**, which makes the robot walk randomly; **attraction** and **repulsion**, which makes the robot move towards or away from its peers, respectively; and **phototaxis** and **anti-phototaxis**, which makes the robot move towards or away from a source of light, respectively. The conditions are **fixed-probability**, which is true with a fixed probability; **white-**, **gray-**, and **black-floor**, which are true when the floor situated below the robot is white, gray, or black, respectively; **neighbor-count**, which is true when a sufficient number of neighboring robots are perceived; and **inverted-neighbor-count**, which is true when sufficiently few neighboring robots are perceived. These modules might have free parameters that are tuned by the optimization algorithm within the optimization process. At every control cycle, the low-level behavior associated with the current state is executed, and the conditions associated to the outgoing edges of the current state are evaluated. The robot executes the same low-level behavior as long as the conditions are evaluated as false. If at least one condition is evaluated as true, one of them is randomly selected, the corresponding transition takes place, and the current state is updated accordingly.

`Vanilla` produces finite state machines with up to four states (or low-level behaviors) to which up to four edges (or conditions) can be connected. To search the space of the possible finite state machine that can be produced by assembling the aforementioned modules and by fine tuning their parameters, `Vanilla` adopts F-race [11, 8], an optimization algorithm originally developed to configure metaheuristics. F-race has been designed to handle stochasticity, it is therefore appropriate to use it in the context of swarm robotics as the performance of an instance of control software varies with the operating conditions (e.g., the initial position and orientation of the robots) and with the contingencies encountered by the swarm. An execution of F-race is reminiscent of a race: a number of candidate solutions are incrementally tested and discarded from the race, should they display low performance. The initial set of candidate solutions are generated randomly at the beginning of the optimization process. At each step of the F-race algorithm, using simulation, the surviving candidate solutions (that is, those that have not been discarded in the previous iterations) are evaluated on a test case which is characterized by the initial position and orientation of the robots, and (possibly) the configuration of the working environment. Based on the evaluations of all previous steps, the candidate solutions whose expected performance is statistically dominated by at least another one are discarded from the race and are not evaluated any further in the following steps.

`Vanilla` has been studied in two experiments. In the first one, it was compared, on two missions, against an implementation of the neuro-evolutionary approach called `EvoStick` [33]. In simulation, `EvoStick` was able to find control software that displayed more sophisticated behaviors than those found by `Vanilla`. However, the control software generated by `EvoStick` was unable to reproduce these behaviors once ported on the physical robots, and suffered an important performance drop. On the contrary, the control software generated by `Vanilla` transitioned smoothly from simulation to reality and much lesser performance drop was observed. As a result, what we call a *rank inversion* was observed in both missions: `EvoStick` outperformed `Vanilla` in simulation, but `Vanilla` outperformed `EvoStick` in reality.

In the second experiment, the control software produced by `Vanilla` and `EvoStick` was compared against the one produced by five human experts [32]. The experimental protocol devised by Francesca et al. was an original contribution to swarm robotics on its own: each of the five experts proposed a mission for which

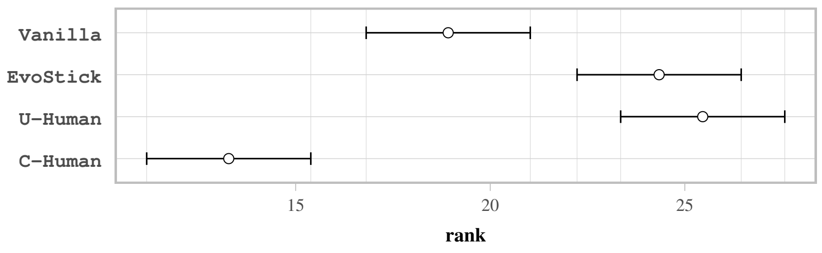

Fig. 2 Outcome of the Friedman test performed on the results of the second experiment involving Vanilla [33]. For each of the methods, the plot represents the average rank and its 95% confidence interval evaluated on five missions in robot experiments. The lower the rank, the better. EvoStick is an implementation of the neuro-evolutionary approach; U-Human and C-Human are manual methods. The performance of two methods is to be considered as significantly different if their respective confidence interval do not overlap [23].

control software had to be produced. The five experts did not have any knowledge of the internal functioning of Vanilla, therefore, they could not have favored or disfavored Vanilla with their choice of a mission. Each expert was then asked to manually produce control software for two missions (other than the one they proposed). For one of them, the expert was left free to design the control software as they deemed appropriate—we call this approach U-Human. For the other one, the experts was constrained to use the modules and control architecture of Vanilla—we call this approach C-Human. In other words, in C-Human, the expert plays the role of Vanilla's optimization algorithm. All in all, for each of the five missions, the authors obtained control software produced by U-Human, C-Human, and by Vanilla and EvoStick.

The results of this second experiment confirm those of the first one: the control software produced by Vanilla outperformed the one produced by EvoStick when the comparison was performed in reality. Compared to the manual methods, Vanilla outperformed U-Human, but was outperformed by C-Human. The results of this second experiment are reported in Figure 2. Because the only difference between Vanilla and C-Human lies in the way the space of possible solutions is explored—the former uses the F-race optimization algorithm, the latter relies on an implicit search performed by a human—Francesca et al. argued that Vanilla could be improved by adopting a better optimization algorithm [32]. This conclusion gave birth to a second specialization of AutoMoDe: Chocolate [31].

Chocolate uses an improved version of the optimization algorithm adopted in Vanilla: Iterated F-race [3, 12, 57]. Iterated F-race has shown to outperform F-race when used to configure meta-heuristics [12]. In Iterated F-race, multiple executions of the F-race algorithm are iteratively conducted. In the first iteration, the initial set of candidates is obtained by uniformly sampling the space of possible solutions. In the following ones, the initial set of candidates are obtained by sampling the space of solutions according to a distribution that gives higher priority to the solutions that are close to the surviving solutions of the previous iteration. We

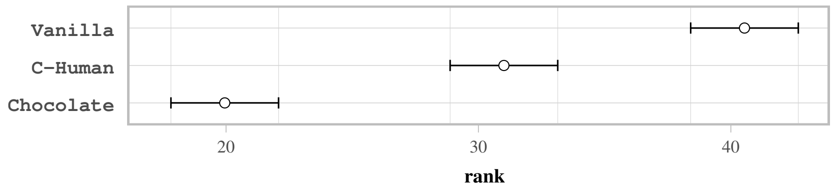

Fig. 3 Outcome of the Friedman test performed on the results of the experiment comparing Vanilla, Chocolate and C-Human [31]. For each of the methods, the plot represent the average rank and its 95% confidence interval, evaluated on the same five missions as in Figure 2. The lower the rank, the better.

refer the reader to [57] for a detailed description of Iterated F-race. Chocolate was compared with Vanilla and C-Human on the basis of the same five missions proposed by the human experts for the previous experiment. The results, depicted in Figure 3, show that Chocolate improves over Vanilla, and the improvement is such that Chocolate also outperforms C-Human. The convincing results of Chocolate have motivated the creation and study of other further specifications of AutoMoDe: Gianduja [43], Maple [49], and Waffle [69].

With Gianduja, Hasselmann et al. [43] studied the emergence of a communication protocol between robots in the automatic design of control software for robot swarms. The authors considered a reference model that comprises an extra capability with respect to the reference model on the basis of which Chocolate was developed: the robots are allowed to locally broadcast and receive a message. This message is *a priori* meaningless, and the authors therefore studied whether an automatic design process is able to assign to the message an appropriate semantics on a per-mission basis. The authors extended the sets of behavioral and conditional modules of Vanilla (and Chocolate) so as to leverage the enhanced reference model. Two new behavioral modules, **attraction-to-message** and **repulsion-from-message**, were included: the robot moves towards or away from the peers that are broadcasting a message, respectively. Also two new conditional modules, **message-count** and **inverted-message-count** were included: a transition is enabled if the number of messages received is larger or smaller than a parameter, respectively. Except for the new modules, Gianduja is defined as Chocolate: like Chocolate, it adopts Iterated F-race as an optimization algorithm, and it generates control software in the form of probabilistic finite state machines. The authors tested Gianduja on three missions, and for each of them, the design process produced control software that leveraged the communication capability of the robots in a meaningful way.

With Maple, Kuckling et al. [49] explored the use of another control architecture for the automatic design of robot swarms: behavior trees. Indeed, instead of combining the predefined modules into a probabilistic finite state machine—as it is done in Vanilla, Chocolate, and Gianduja—Maple combines them into a behavior tree. The goal of the authors was not to create a new automatic design method that outperforms `Chocolate`, but rather to study the impact of the control architecture on the performance and on the robustness to the reality gap. To do so, the authors developed `Maple` so that it differs from `Chocolate` only in the control architecture: the two methods share the same sets of predefined modules, they use the same optimization algorithm, and they produce control software that comprise a maximum of four behavioral modules. The authors compared the performance of `Chocolate`, `EvoStick`, and `Maple` on two missions. The results show that `Chocolate` and `Maple` performed similarly, and that they both displayed a smaller performance drop with respect to `EvoStick` when the control software produced was ported to the robots. These results confirm the conjecture that the robustness of AutoMoDe to the reality gap originates from its modular nature, and indicate that the architecture into which the modules are combined is a secondary issue.

With `Waffle`, Salman et al. [69] generalized the functionality of AutoMoDe to not only generate control software, but also to concurrently configure the hardware of the swarm itself. Concerning the generation of control software, `Waffle` is identical to `Chocolate`. The novelty is the configuration of the robot swarm: `Waffle` automatically optimizes the hardware configuration of the individual robots and the number of robots comprised in the swarm. In their study, to prove their concept, the authors restricted the hardware configuration of the individual robots to the selection of a local communication module out of a set of hypothetical but realistic candidates. These candidates were defined as variants of an existing infra-red communication module [39]. Some of the candidates are more and some are less capable than the existing module in terms of their transmission range and communication reliability. Those that are more capable are also more expensive and draw more current per unit time, and vice versa. The authors studied the performance of `Waffle` under two types of economic constraints: constraints on the monetary budget available, and/or on the battery capacity of the individual robots. Without these constraints, the design problem is trivial: the design process always selects the largest possible swarm composed of the most capable robots. The constraints makes the design problem more interesting and realistic: indeed, every real-world design problem involves trade-offs of an economic nature. Due to the monetary constraint, the automatic design process must choose between: (a) having a swarm composed of many but relatively incapable robots; and (b) few capable ones. Due to the power constraint, it must choose between: (a) robots with capable hardware that, due to their high power consumption, can operate for a relatively short time; and (b) robots with less capable hardware that, due to their low power consumption, can operate for a relatively long time. Salman et al. tested `Waffle` on three missions, and, for each mission, performed the design process under nine levels of the aforementioned economic constraint. The results show that the optimal hardware configuration and behavior of the swarm depends on the mission to be accomplished and on the constraint imposed. They also show that the principles of the automatic modular design can be successfully applied to the concurrent design of hardware and control software of robot swarms.

Two other specializations of AutoMoDe were developed: `IcePop` [50] and `Coconut` [74]. With `IcePop`, Kuckling et al. [50] investigated the use of yet another optimization algorithm: simulated annealing [34]. With `Coconut`, Spaey et al. [74] extended the set of behavior modules to include different exploration schemes, and investigated their influence on the performance of the swarms that can then be generated.

4 Further corroboration of our working hypothesis

The results of the different specializations of AutoMoDe corroborate our original working hypothesis: the reality gap problem bears similarity with the generalization problem faced in machine learning and reducing the complexity of control architecture by injecting bias produces control software with increased robustness to the reality gap. In this section, we present other studies that further corroborate our insight on the subject.

In supervised machine learning, the phenomenon of overfitting—or alternatively, overtraining—occurs when the learning process is protracted beyond an ideal threshold, which causes the performance on the training set and on the test set to diverge [1, 64]. Indeed, past an optimal level of the training effort, the approximator overspecializes to the examples contained in the training set, which impairs its ability to generalize to a test set. In an experiment, Birattari et al. [9] observed a phenomenon that can be seen as the off-line automatic design counterpart of the overtraining issue faced in supervised machine leaning. They called the phenomenon they observed *overdesign*. In this experiment, the authors evaluated both in simulation and in reality the performance of the best instances of control software produced by an neuro-evolutionary design method at different levels of the design effort, that is, after an increasing number of iterations of the evolutionary algorithm. The results reported in Figure 4 show that, past an optimal design effort, the performance that the automatically generated swarm obtains in reality diverges from the one it obtains in simulation.

In addition to corroborating our hypothesis on the similarity between the reality gap problem and the generalization problem in machine learning, these results illustrate the relative nature of the occurrence of the effects of the reality gap. Indeed, not all instances of control software are equally affected by the same reality gap. In this specific case, some instances, notably those produced at the beginning of the design process, display a smaller performance drop than those generated later on in the design process. This can lead to the observation of what the authors call a rank inversion: an instance A of control software outperforms an instance B in simulation, but B outperform A in reality. In the results of Figure 4, a rank inversion occurred between instances of control software generated by the same design method, but a similar phenomenon has also been observed when comparing different design methods [33, 31].

The phenomena of overdesign and of rank inversion raise two fundamental questions that are core to the off-line automatic design of robot swarms: what design method will produce control software that will yield the best performance once up-

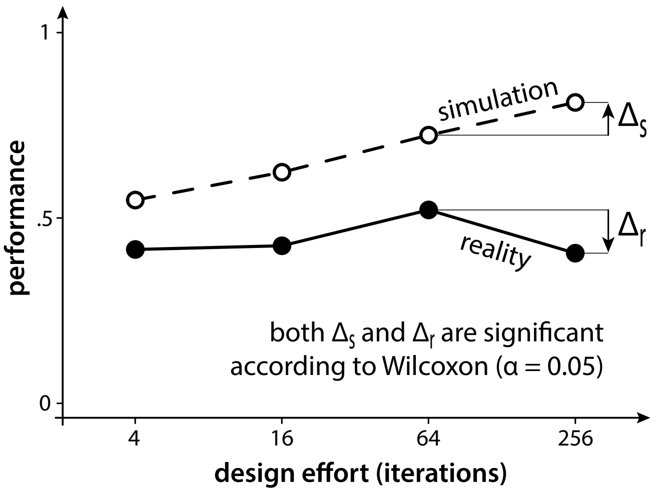

Fig. 4 Results of an experiment in which the best artificial neural networks produced after 4, 16, 64, and 256 iterations of an evolutionary algorithm were evaluated in simulation and in reality. We refer the reader to the original publication for the details on experimental protocol and results [9].

loaded to the robots? When should the design process be stopped? The answers to these questions likely vary from mission to mission and, at the moment, they can be obtained only via robot experiments, which are expensive, time consuming, and not always feasible [10]. What would be highly desirable to have is a simulation-only procedure that could provide reliable estimations of the real-world performance of control software and of its ability to cross the reality gap. Such a procedure could be used to implement an early stopping mechanism [60, 21, 64] so as to prevent overdesign, and also to select among different automatic design methods the one to use.

To the best of our knowledge, Koos et al. [48] were the first to use an artificial, simulation-only reality gap in the context of the automatic design of robot swarms. The authors did so to perform experiments to further assess the performance of a design method they proposed. Ligot and Birattari [53] formally introduced the notion of *pseudo-reality*: a simulation model, different from the one used during the design, used to evaluate the intrinsic robustness of control software to the reality gap. In that paper, a pseudo-reality was generated by hand so as to replicate previously observed performance drop and rank inversion. With the notion of pseudo-reality, the authors investigated the conditions under which the effects of the reality gap manifest. The results show that the effects of the reality gap manifest themselves regardless of whether the evaluation context is more or less complex than the context in which the control software has been designed. These results also substantiate our contention that the effects of the reality gap are due to the fact that control software overfits the context in which it has been designed, and hence that design methods overfit the context in which they operate. In further experiments, Ligot and Birattari [54] obtained similar results with multiple simulation models used as pseudo-reality, all being uniformly sampled around the model used during the design. We foresee that the notion of pseudo-reality can be leveraged in the definition of a simulation-only protocol that would be able to tell, with reasonable confidence, which automatic design method to apply to the problem at hand, and when is the most appropriate time to stop the design process.

5 Discussion on future research directions

Swarm robotics is an appealing way of realizing a group of robots that should operate in environments where the risk of damaging or losing robots is high and establishing a reliable communication between robots and between them and a supervisory entity is unfeasible. Ideal applications of swarm robotics include surveillance, search and rescue, demining, and underwater or space exploration. Yet, the real-world application of swarm robotics is not imminent, which is essentially due to a lack of a general methodology to reliably program the individual robots so that the desired collective behavior emerges from their interactions.

In this chapter, we focused on what appears to be a promising approach to solve the complex problem of designing a robot swarm: automatic off-line design. In particular, we highlighted the results recently obtained with AutoMoDe, an automatic method that generates control software for robot swarms by assembling predefined software modules. AutoMoDe was specifically conceived to address the main challenge in the automatic off-line approach: produce control software that is robust to the so-called reality gap, that is, that attains in reality a performance that is comparable to the one displayed in the simulations of the design phase. The core idea of AutoMoDe is to restrain the representational power of the control architecture. By doing so, it reduces the risk that the control software produced overfits the peculiarities of the simulations and fails subsequently to generalize to reality. Although AutoMoDe produced satisfactory results, work remains to be done to gain a full understanding of the reality gap problem and conceive a reliable automatic design method. A number of specializations of AutoMoDe have been proposed so far that differ one from the other in some of their components, including the optimization algorithm and the architecture in which modules are assembled. The space of the possible alternatives is vast and the research conducted so far has only scratched the surface. Indeed, the use of two architectures has been investigated so far: probabilistic finite state machines and behavior trees. Other alternatives are worth being explored, such as executable Petri nets [62] and Boolean networks [67, 6]. Similarly, only three optimization algorithms have been investigated so far and a few are under analysis. Yet, many more exist or could be conceived specifically for being adopted in the context of AutoMoDe. Finally, we presented some results in the concurrent development of control software and configuration of the hardware. Although preliminary, these results are promising and clearly indicate that the principles of the automatic modular design of AutoMoDe are not restricted to the generation of control software but have a more general applicability.

References

1. Amari, S., Murata, N., Müller, K.R., Finke, M., Yang, H.H.: Statistical theory of overtraining – is cross-validation asymptotically effective? In: D.S. Touretzky, M.C. Mozer, M.E. Hasselmo (eds.) NIPS'95: Advances in Neural Information Processing Systems, pp. 176–182. MIT Press, Cambridge, MA (1996)
2. Bäck, T., Fogel, D.B., Michalewicz, Z.: Handbook of Evolutionary Computation. IOP Publishing Ltd., Bristol, UK (2000)
3. Balaprakash, P., Birattari, M., Stützle, T.: Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement. In: Hybrid Metaheuristics, 4th International Workshop, HM 2007, *LNCS*, vol. 4771, pp. 108–122. Springer, Berlin, Germany (2007)
4. Baldassarre, G., Trianni, V., Bonani, M., Mondada, F., Dorigo, M., Nolfi, S.: Self-organized coordinated motion in groups of physically connected robots. IEEE Trans. Syst. Man Cy. B **37**, 224–39 (2007). DOI 10.1109/TSMCB.2006.881299
5. Beal, J., Dulman, S., Usbeck, K., Viroli, M., Correll, N.: Organizing the aggregate: languages for spatial computing. In: Formal and Practical Aspects of Domain-Specific Languages: Recent Developments, chap. 16, pp. 436–501. IGI Global (2012). DOI 10.4018/978-1-4666-2092-6.ch016
6. Benedettini, S., Villani, M., Roli, A., Serra, R., Manfroni, M., Gagliardi, A., Pinciroli, C., Birattari, M.: Dynamical regimes and learning properties of evolved boolean networks. Neurocomputing **99**, 111–123 (2013). DOI 10.1016/j.neucom.2012.05.023
7. Berman, S., Kumar, V., Nagpal, R.: Design of control policies for spatially inhomogeneous robot swarms with application to commercial pollination. In: IEEE International Conference on Robotics and Automation – ICRA, pp. 378–385. IEEE, Piscataway, NJ (2011). DOI 10.1109/ICRA.2011.5980440
8. Birattari, M.: Tuning Metaheuristics: A Machine Learning Perspective. Springer, Berlin, Germany (2009)
9. Birattari, M., Delhaisse, B., Francesca, G., Kerdoncuff, Y.: Observing the effects of overdesign in the automatic design of control software for robot swarms. In: M. Dorigo, , M. Birattari, X. Li, M. López-Ibáñez, K. Ohkura, C. Pinciroli, T. Stützle (eds.) Swarm Intelligence – ANTS, *LNCS*, vol. 9882, pp. 45–57. Springer, Cham, Switzerland (2016). DOI 10.1007/978-3-319-44427-7_13
10. Birattari, M., Ligot, A., Bozhinoski, D., Brambilla, M., Francesca, G., Garattoni, L., Garzón Ramos, D., Hasselmann, K., Kegeleirs, M., Kuckling, J., Pagnozzi, F., Roli, A., Salman, M., Stützle, T.: Automatic off-line design of robot swarms: A manifesto. Front. Robot. AI **6**, 59 (2019). DOI 10.3389/frobt.2019.00059
11. Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K.: A racing algorithm for configuring metaheuristics. In: W. Langdon, et al. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO, pp. 11–18. Morgan Kaufmann, San Francisco CA (2002)
12. Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T.: F-race and iterated f-race: An overview. In: T. Bartz-Beielstein, M. Chiarandini, L. Paquete, M. Preuss (eds.) Experimental Methods for the Analysis of Optimization Algorithms, pp. 311–336. Springer, Berlin, Germany (2010). DOI 10.1007/978-3-642-02538-9_13
13. Bongard, J.C.: Evolutionary robotics. Commun. ACM **56**(8), 74–83 (2013). DOI 10.1145/2493883
14. Bozhinoski, D., Di Ruscio, D., Malavolta, I., Pelliccione, P., Tivoli, M.: Flyaq: enabling non-expert users to specify and generate missions of autonomous multicopters. In: IEEE/ACM Int. Conf. on Automated Software Engineering – ASE, pp. 801–806. IEEE, Piscataway, NJ (2015). DOI 10.1109/ASE.2015.104

15. Brambilla, M., Brutschy, A., Dorigo, M., Birattari, M.: Property-driven design for swarm robotics: A design method based on prescriptive modeling and model checking. ACM Trans. Auton. Adapt. Syst. **9**(4), 17.1–28 (2015). DOI 10.1145/2700318
16. Brambilla, M., Ferrante, E., Birattari, M., Dorigo, M.: Swarm robotics: a review from the swarm engineering perspective. Swarm Intell. **7**(1), 1–41 (2013). DOI 10.1007/s11721-012-0075-2
17. Bredeche, N., Haasdijk, E., Prieto, A.: Embodied evolution in collective robotics: a review. Frontiers in Robotics and AI **5**, 12 (2018). DOI 10.3389/frobt.2018.00012
18. Bredeche, N., Montanier, J.M., Liu, W., Winfield, A.F.: Environment-driven distributed evolutionary adaptation in a population of autonomous robotic agents. Mathematical and Computer Modelling of Dynamical Systems **18**(1), 101–129 (2012)
19. Brooks, R.: Artificial life and real robots. In: Towards a Practice of Autonomous Systems. Proceedings of the First European Conference on Artificial Life, pp. 3–10. MIT Press, Cambridge, MA (1992)
20. Brugali, D.: Software engineering for experimental robotics, vol. 30. Springer (2007). DOI 10.1007/978-3-540-68951-5
21. Caruana, R., Lawrence, S., Giles, C.L.: Overfitting in neural nets: backpropagation, conjugate gradient, and early stopping. In: T.K. Leen, T.G. Dietterich, V. Tresp (eds.) Advances in Neural Information Processing Systems 13, pp. 402–408. MIT Press (2001)
22. Christensen, A.L., Dorigo, M.: Evolving an integrated phototaxis and hole-avoidance behavior for a swarm-bot. In: Arfiticial Life – ALIFE, pp. 248–254. MIT Press, Cambridge, MA (2006)
23. Conover, W.J.: Practical Nonparametric Statistics, third edn. John Wiley & Sons, New York, NY (1999)
24. Di Ruscio, D., Malavolta, I., Pelliccione, P.: A family of domain-specific languages for specifying civilian missions of multi-robot systems. In: Proceedings of the 1st International Workshop on Model-Driven Robot Software Engineering – MORSE, pp. 13–26 (2014)
25. Dorigo, M., Birattari, M.: Swarm intelligence. Scholarpedia **2**(9), 1462 (2007). DOI 10.4249/scholarpedia.1462
26. Dorigo, M., Birattari, M., Brambilla, M.: Swarm robotics. Scholarpedia **9**(1), 1463 (2014). DOI 10.4249/scholarpedia.1463
27. Ferrante, E., Duéñez Guzmán, E., Turgut, A.E., Wenseleers, T.: Geswarm: Grammatical evolution for the automatic synthesis of collective behaviors in swarm robotics. In: Genetic and Evolutionary Computation – GECCO, pp. 17–24. ACM, New York, NY (2013). DOI 10.1145/2463372.2463385
28. Ferrante, E., Turgut, A., Duéñez-Guzmán, E., Dorigo, M., Wenseleers, T.: Evolution of self-organized task specialization in robot swarms. PLoS Comput. Biol. **11**(8), e1004,273 (2015). DOI 10.1371/journal.pcbi.1004273
29. Floreano, D., Husbands, P., Nolfi, S.: Evolutionary robotics. Handbook of Robotics pp. 1423–1451 (2008)
30. Francesca, G., Birattari, M.: Automatic design of robot swarms: achievements and challenges. Front. Robot. AI **3**(29), 1–9 (2016). DOI 10.3389/frobt.2016.00029
31. Francesca, G., Brambilla, M., Brutschy, A., Garattoni, L., Miletitch, R., Podevijn, G., Reina, A., Soleymani, T., Salvaro, M., Pinciroli, C., Birattari, M.: AutoMoDe-Chocolate: automatic design of control software for robot swarms. Swarm Intell. **9**(2/3), 125–152 (2015). DOI 10.1007/s11721-015-0107-9
32. Francesca, G., Brambilla, M., Brutschy, A., Garattoni, L., Miletitch, R., Podevijn, G., Reina, A., Soleymani, T., Salvaro, M., Pinciroli, C., Trianni, V., Birattari, M.: An experiment in automatic design of robot swarms: AutoMoDe-Vanilla, EvoStick, and human experts. In: Swarm Intelligence – ANTS, *LNCS*, vol. 8667, pp. 25–37. Springer, Berlin, Germany (2014)
33. Francesca, G., Brambilla, M., Brutschy, A., Trianni, V., Birattari, M.: AutoMoDe: a novel approach to the automatic design of control software for robot swarms. Swarm Intell. **8**(2), 89–112 (2014). DOI 10.1007/s11721-014-0092-4
34. Franzin, A., Stützle, T.: Revisiting simulated annealing: A component-based analysis. Computers & Operations Research **104**, 191 – 206 (2019). DOI 10.1016/j.cor.2018.12.015

35. Garattoni, L., Birattari, M.: Autonomous task sequencing in a robot swarm. Science Robotics **3**(20) (2018). DOI 10.1126/scirobotics.aat0430
36. Garattoni, L., Francesca, G., Brutschy, A., Pinciroli, C., Birattari, M.: Software infrastructure for e-puck (and TAM). Tech. Rep. TR/IRIDIA/2015-004, IRIDIA, Université libre de Bruxelles, Belgium (2015)
37. Geman, S., Bienenstock, E., Doursat, R.: Neural networks and the bias/variance dilemma. Neural computation **4**(1), 1–58 (1992)
38. Gomes, J., Urbano, P., Christensen, A.: Evolution of swarm robotics systems with novelty search. Swarm Intell. **7**, 115–144 (2013). DOI 10.1007/s11721-013-0081-z
39. Gutiérrez, Á., Campo, A., Dorigo, M., Donate, J., Monasterio-Huelin, F., Magdalena, L.: Open e-puck range & bearing miniaturized board for local communication in swarm robotics. In: K. Kosuge (ed.) IEEE International Conference on Robotics and Automation, ICRA, pp. 3111–3116. IEEE, Piscataway, NJ (2009)
40. Haasdijk, E., Bredeche, N., Eiben, A.: Combining environment-driven adaptation and task-driven optimisation in evolutionary robotics. PloS ONE **9**(6), e98,466 (2014)
41. Hamann, H., Wörn, H.: A framework of space–time continuous models for algorithm design in swarm robotics. Swarm Intell. **2**(2–4), 209–239 (2008). DOI 10.1007/s11721-008-0015-3
42. Harvey, I., Husband, P., Cliff, D., Thompson, A., Jakobi, N.: Evolutionary robotics: the sussex approach. Robotics and Autonomous Systems **20**(2), 205–224 (1997)
43. Hasselmann, K., Robert, F., Birattari, M.: Automatic design of communication-based behaviors for robot swarms. In: M. Dorigo, et al. (eds.) Swarm Intelligence, ANTS, *LNCS*, vol. 11172, pp. 16–29. Springer, Cham, Switzerland (2018)
44. Hauert, S., Zufferey, J.C., Floreano, D.: Evolved swarming without positioning information: an application in aerial communication relay. Auton. Robots **26**(1), 21–32 (2009). DOI 10.1007/s10514-008-9104-9
45. Jakobi, N., Husbands, P., Harvey, I.: Noise and the reality gap: the use of simulation in evolutionary robotics. LNAI **929**, 704–720 (1995). DOI 10.1007/3-540-59496-_337
46. Kazadi, S.: Model independence in swarm robotics. Int. J. Intell. Comput. and Cybern. **2**(4), 672–694 (2009). DOI 10.1108/17563780911005836
47. König, L., Mostaghim, S.: Decentralized evolution of robotic behavior using finite state machines. Int. J. Intell. Comput. and Cybern. **2**(4), 695–723 (2009). DOI 10.1108/17563780911005845
48. Koos, S., Mouret, J.B., Doncieux, S.: The transferability approach: crossing the reality gap in evolutionary robotics. IEEE Trans. Evol. Comput. **17**(1), 122–145 (2013). DOI 10.1109/TEVC.2012.2185849
49. Kuckling, J., Ligot, A., Bozhinoski, D., Birattari, M.: Behavior trees as a control architecture in the automatic modular design of robot swarms. In: Swarm Intelligence – ANTS, *LNCS*, vol. 11172, pp. 30–43. Springer, Cham, Switzerland (2018). DOI 10.1007/978-3-030-00533-7_3
50. Kuckling, J., Ubeda Arriaza, K., Birattari, M.: Simulated annealing as an optimization algorithm in the automatic modular design of robot swarms. In: K. Beuls, B. Bogaerts, G. Bontempi, P. Geurts, N. Harley, B. Lebichot, T. Lenaerts, G. Louppe, P.V. Eecke (eds.) Proceedings of the Reference AI & ML Conference for Belgium, Netherlands & Luxemburg, BNAIC/BENELEARN 2019, *CEUR Workshop Proceedings*, vol. 2491. CEUR-WS.org, Aachen, Germany (2019)
51. Lee, J.B., Arkin, R.C.: Adaptive multi-robot behavior via learning momentum. In: C.S. George Lee (ed.) IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, pp. 2029–2036. IEEE, Piscataway, NJ (2003)
52. Li, S., Batra, R., Brown, D., Chang, H.D., Ranganathan, N., Hoberman, C., Rus, D., Lipson, H.: Particle robotics based on statistical mechanics of loosely coupled components. Nature **567**(7748), 361–365 (2019). DOI 10.1038/s41586-019-1022-9
53. Ligot, A., Birattari, M.: On mimicking the effects of the reality gap with simulation-only experiments. In: Swarm Intelligence – ANTS, *LNCS*, vol. 11172, pp. 109–122. Springer, Cham, Switzerland (2018). DOI 10.1007/978-3-030-00533-7_9

54. Ligot, A., Birattari, M.: Simulation-only experiments to mimic the effects of the reality gap in the automatic design of robot swarms. Swarm Intell. pp. 1–24 (2019). DOI 10.1007/s11721-019-00175-w
55. Lipson, H.: Evolutionary robotics and open-ended design automation. Biomimetics **17**, 129–155 (2005). DOI 10.1201/9781420037715.ch4
56. Lopes, Y.K., Trenkwalder, S.M., Leal, A.B., Dodd, T.J., Groß, R.: Supervisory control theory applied to swarm robotics. Swarm Intell. **10**(1), 65–97 (2016). DOI 10.1007/s11721-016-0119-0
57. López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., Stützle, T.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives **3**, 43–58 (2016)
58. Marocco, D., Nolfi, S.: Emergence of communication in embodied agents evolved for the ability to solve a collective navigation problem. Connection Science **19**(1), 53–74 (2007). DOI 10.1080/09540090601015067
59. Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J.C., Floreano, D., Martinoli, A.: The e-puck, a robot designed for education in engineering. In: P. Gonçalves, P. Torres, C. Alves (eds.) Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions, pp. 59–65. Instituto Politécnico de Castelo Branco, Portugal (2009)
60. Morgan, N., Bourlard, H.: Generalization and parameter estimation in feedforward nets: some experiments. In: D.S. Touretzky (ed.) Advances in Neural Information Processing Systems 2, NIPS 1990, pp. 630–637. Morgan Kaufmann, San Francisco (1990)
61. Nolfi, S., Floreano, D.: Evolutionary Robotics. MIT Press, Cambridge, MA (2000)
62. Pereira, J.N., Silva, P., Lima, P.U., Martinoli, A.: Formalizing institutions as executable petri nets for distributed robotic systems. In: T. Lenaerts, M. Giacobini, H. Bersini, P.Bourgine, M. Dorigo, R. Doursat (eds.) Advances in Artificial Life, ECAL 2011, pp. 646–653. MIT Press, Cambridge, MA (2011)
63. Pinciroli, C., Beltrame, G.: Buzz: A programming language for robot swarms. IEEE Software **33**, 97–100 (2016). DOI 10.1109/MS.2016.95
64. Prechelt, L.: Early stopping – but when? In: G. Montavon, G.B. Orr, K.R. Müller (eds.) Neural Networks: Tricks of the Trade: Second Edition, *LNCS*, vol. 7000, pp. 53–67. Springer, Berlin, Heidelberg (2012). DOI 10.1007/978-3-642-35289-8_5
65. Quinn, M., Smith, L., Mayley, G., Husbands, P.: Evolving controllers for a homogeneous system of physical robots: structured cooperation with minimal sensors. Philos. Trans. R. Soc. A **361**, 2321–43 (2003). DOI 10.1098/rsta.2003.1258
66. Reina, A., Valentini, G., Fernàndez-Oto, C., Dorigo, M., Trianni, V.: A design pattern for decentralised decision making. PLoS ONE **10**(10), e0140,950 (2015). DOI 10.1371/journal.pone.0140950
67. Roli, A., Manfroni, M., Pinciroli, C., Birattari, M.: On the design of boolean network robots. In: Applications of Evolutionary Computation, *LNCS*, vol. 6624/5, pp. 43–52. Springer, Berlin, Germany (2011). DOI 10.1007/978-3-642-20525-5_5
68. Rubenstein, M., Cornejo, A., Nagpal, R.: Programmable self-assembly in a thousand-robot swarm. Science **345**(6198), 795–799 (2014). DOI 10.1126/science.1254295
69. Salman, M., Ligot, A., Birattari, M.: Concurrent design of control software and configuration of hardware for robot swarms under economic constraints. PeerJ Computer Science **5**, e221 (2019). DOI 10.7717/peerj-cs.221
70. Schlegel, C., Lotz, A., Lutz, M., Stampfer, D., Inglés-Romero, J.F., Vicente-Chicote, C.: Model-driven software systems engineering in robotics: covering the complete life-cycle of a robot. It-Information Technology **57**(2), 85–98 (2015). DOI 10.1515/itit-2014-1069
71. Silva, F., Duarte, M., Correia, L., Oliveira, S., Christensen, A.: Open issues in evolutionary robotics. Evol. Comput. **24**(2), 205–236 (2016). DOI 10.1162/EVCO_a_00172
72. Silva, F., Urbano, P., Correia, L., Christensen, A.L.: odNEAT: An algorithm for decentralised online evolution of robotic controllers. Evol. Comput. **23**(3), 421–449 (2015)

73. Slavkov, I., Carrillo-Zapata, D., Carranza, N., Diego, X., Jansson, F., Kaandorp, J., Hauert, S., Sharpe, J.: Morphogenesis in robot swarms. Science Robotics **3**(25) (2018). DOI 10.1126/scirobotics.aau9178
74. Spaey, G., Kegeleirs, M., Garzón Ramos, D., Birattari, M.: Comparison of different exploration schemes in the automatic modular design of robot swarms. In: K. Beuls, B. Bogaerts, G. Bontempi, P. Geurts, N. Harley, B. Lebichot, T. Lenaerts, G. Louppe, P.V. Eecke (eds.) Proceedings of the Reference AI & ML Conference for Belgium, Netherlands & Luxemburg, BNAIC/BENELEARN 2019, *CEUR Workshop Proceedings*, vol. 2491. CEUR-WS.org, Aachen, Germany (2019)
75. Trianni, V.: Evolutionary Swarm Robotics. Springer, Berlin, Germany (2008)
76. Trianni, V.: Evolutionary robotics: model or design? Front. Robot. AI **1**, 13 (2014). DOI 10.3389/frobt.2014.00013
77. Trianni, V., López-Ibáñez, M.: Advantages of task-specific multi-objective optimisation in evolutionary robotics. PLoS ONE **10**(8), e0136,406 (2015). DOI 10.1371/journal.pone.0136406
78. Trianni, V., Nolfi, S.: Self-organizing sync in a robotic swarm: a dynamical system view. IEEE Trans. Evol. Comput. **13**(4), 722–741 (2009). DOI 10.1109/TEVC.2009.2015577
79. Tuci, E.: An investigation of the evolutionary origin of reciprocal communication using simulated autonomous agents. Biological Cybernetics **101**(3), 183–99 (2009). DOI 10.1007/s00422-009-0329-2
80. Waibel, M., Keller, L., Floreano, D.: Genetic team composition and level of selection in the evolution of multi-agent systems. IEEE Trans. Evol. Comput. **13**, 648–660 (2009). DOI 10.1109/TEVC.2008.2011741
81. Watson, R., G. Ficici, S., Pollack, J.: Embodied evolution: distributing an evolutionary algorithm in a population of robots. Robot. Auton. Syst. **39**, 1–18 (2002). DOI 10.1016/S0921-8890(02)00170-7
82. Werfel, J., Petersen, K., Nagpal, R.: Designing collective behavior in a termite-inspired robot construction team. Science **343**(6172), 754–758 (2014). DOI 10.1126/science.1245842
83. Winfield, A., Erbas, M.: On embodied memetic evolution and the emergence of behavioural traditions in robots. Memetic Computing **3**(4), 261–270 (2011). DOI 10.1007/s12293-011-0063-x
84. Wolpert, D.: On bias plus variance. Neural Computation **9**, 1211–1243 (1997). DOI 10.1162/neco.1997.9.6.1211
85. Xie, H., Sun, M., Fan, X., Lin, Z., Chen, W., Wang, L., Dong, L., He, Q.: Reconfigurable magnetic microrobot swarm: Multimode transformation, locomotion, and manipulation. Science Robotics **4**(28) (2019). DOI 10.1126/scirobotics.aav8006
86. Yang, G.Z., Bellingham, J., E. Dupont, P., Fischer, P., Floridi, L., Full, R., Jacobstein, N., Kumar, V., McNutt, M., Merrifield, R., Nelson, B., Scassellati, B., Taddeo, M., Taylor, R., Veloso, M., Lin Wang, Z., Wood, R.: The grand challenges of Science Robotics. Science Robotics **3**(14), eaar7650 (2018). DOI 10.1126/scirobotics.aar7650
87. Yu, J., Wang, B., Du, X., Wang, Q., Zhang, L.: Ultra-extensible ribbon-like magnetic microswarm. Nature Communications **9**(1) (2018). DOI 10.1038/s41467-018-05749-6