Noname manuscript No.
(will be inserted by the editor)

Simulation-only experiments to mimic the effects of the reality gap in the automatic design of robot swarms

Antoine Ligot · Mauro Birattari

the date of receipt and acceptance should be inserted later

Abstract The reality gap—the discrepancy between reality and simulation—is a critical issue in the off-line automatic design of control software for robot swarms, as well as for single robots. It is understood that the reality gap manifests itself as a drop in performance: when control software generated in simulation is ported to physical robots, the performance observed is often disappointing compared with the one obtained in simulation. In this paper, we investigate whether, to observe the effects of the reality gap, it is necessary to assume that the control software is designed in a context that is simpler than the one in which it is evaluated. In a first experiment, we show that a performance drop may be observed also in an artificial, simulation-only reality gap: control software is generated on the basis of a simulation model and assessed on a second one. We will call this second model a *pseudo-reality.* We selected the simulation model to be used as a pseudo-reality by trial and error, so as to qualitatively replicate previously published observations made in experiments with physical robots. The results show that a performance drop occurs even if we can exclude that pseudo-reality is more complex than the simulation model used for the design. In a second experiment, we eliminate the trial-and-error selection of the first experiment by evaluating control software across multiple pseudo-realities, which are sampled around the original simulation model used for the design. The results of the second experiment confirm those of the first one and show that they do not depend on the specific pseudo-reality we previously selected by trial and error. Moreover, they suggest that one could use multiple pseudo-realities to evaluate automatic design methods and, from this simulation-only evaluation, infer their robustness to the reality gap.

The experiments were conceived by the two authors and performed by AL. The article was drafted by AL and revised by the two authors. The research was directed by MB.

IRIDIA, Université libre de Bruxelles, Belgium
E-mail: mbiro@ulb.ac.be

1 Introduction

Inspired by swarm intelligence principles (Dorigo and Birattari 2007), swarm robotics is an engineering discipline whose ultimate goal is to design large groups of robots that coordinate autonomously (Beni 2004; Şahin 2004; Hamann 2018). In a swarm, robots have local sensing and communication capabilities, and do not rely on a robot leader or external infrastructures. The collective behavior emerges from the interaction between the neighboring individuals, and between the individuals and the environment. Because of its decentralized nature, a robot swarm cannot be programmed as a whole: only the individuals can be. Conceiving the individual behavior such that the desired global behavior emerges is a difficult endeavor. To address it, various manual and automatic approaches have been proposed in the literature (Brambilla et al. 2013).

Manual approaches typically rely on the skill and ingenuity of a human designer. Methods exist that can guide the designer to achieve specific global behaviors (Hamann and Wörn 2008; Berman et al. 2011; Brambilla et al. 2015; Reina et al. 2015). However, no generally applicable methodology exists, yet. Automatic approaches implement a sort of design by optimization: by searching for an appropriate instance of the control software of the individual robots, an optimization algorithm maximizes a mission-dependent measure of the collective performance of the swarm (Francesca and Birattari 2016). Automatic design methods can be further classified as either on-line or off-line.

In on-line methods, the design process takes place after the robots are deployed in the target environment (Lee and Arkin 2003; Watson et al. 2002; König and Mostaghim 2009; Bredeche et al. 2012; Haasdijk et al. 2014; Silva et al. 2015). A distributed optimization process, running on the robots themselves, iteratively improves the control software on the basis of their performance, while the robots perform their mission in the target environment. Online methods suffer from a number of drawbacks (Francesca and Birattari 2016), among which: (i) they can handle only a relatively small search space, and as a consequence, might be appropriate for optimizing a few parameters but not for fully designing the behavior of the robots; (ii) their application is limited to the case in which the robots can evaluate their collective, instantaneous performance; and (iii) when testing suboptimal instances of control software (or values of the parameters to be optimized), they may cause damages to the robots and/or to the environment. Although promising for a number of specific applications, we do not deem them as the ultimate solution to the automatic design problem in swarm robotics.

In off-line methods, the design process relies on simulation to reproduce relevant features of the target environment in which the robot will be eventually deployed (Francesca and Birattari 2016). Because the assessment of performance is much faster and cheaper in simulation than on physical robots, off-line methods can explore a larger space of possible instances of control software in a relatively short time. In addition, simulation provides a Godeye view of the swarm, which allows off-line methods to evaluate any possible performance metric. Finally, damaging robots or the environment is irrelevant when they are simulated entities.

However, contrary to on-line methods, off-line methods are faced with the so-called *reality gap*: the difference between simulation and reality, which might be subtle but is unavoidable (Brooks 1992; Jakobi et al. 1995). Due to the reality gap, it is likely that robot swarms do not display the same behavior in simulation and in reality (Floreano et al. 2008). Several approaches have been proposed to cross the reality gap satisfactorily—see Section 2. However, none of them has been studied in details, no extensive comparison has been produced, and the reality gap remains an important issue for off-line methods (Francesca and Birattari 2016; Silva et al. 2016).

According to the domain literature, the reality gap manifests itself in the form of a *performance drop* when control software designed in simulation is ported to physical robots (Floreano et al. 2008; Francesca et al. 2014). It is also understood that the performance drop is a relative problem: instances of control software produced by different methods or under different conditions may be affected to different degrees. This can lead to a phenomenon that we call *rank inversion*: an instance of control software outperforms another one in simulation, but is outperformed by the latter when they are evaluated on physical robots (Francesca et al. 2014; Birattari et al. 2016). On the other hand, what remains an open issue is the true nature of the reality gap. Often, the effects of the reality gap are explained by saying that the optimization process exploits inaccuracies of the simulation models to produce unrealistic behaviors that achieve high performance. Indeed, simulators often neglect some complex physical phenomena, which make them inaccurate but fast. This because accurate simulations would be too time consuming— possibly even more than experiments with real robots—which would lead to prohibitively long optimization processes (Nolfi et al. 1994; Koos et al. 2013).

In the literature, the effects of the reality gap have only been observed when control software has been developed in a relatively simpler setting for then being tested in a more complex one. This holds true for the most common case in which control software is developed in simulation and then tested on real robots. It also holds true for a single but relevant case in which control software has been developed using a first simulator and then tested using a second one. Indeed, to the best of our knowledge, Koos et al. (2013) have been the only ones to consider an artificial, simulation-only reality gap, which they used to investigate the properties of a method they proposed to handle the reality gap—see Section 2. Their artificial reality gap relies on a simplified simulator to be used in the design, and an accurate one to be used for the evaluation.

In the present paper, we investigate whether, to observe the effects of the reality gap, it is necessary to assume that the control software is evaluated in a context that is more complex than the one in which it is designed. We shall call this assumption the *complexity assumption*. Through our investigation, we bring empirical evidence that the effects of the reality gap appear even in cases in which we can exclude that the evaluation is performed in a context that is more complex than the one in which control software is designed. Our results indicate that performance drops should be ascribed to a sort of overfitting of the conditions experienced in the design phase, *regardless of the fact that these conditions are more or less complex than those faced in the evaluation phase.* The core device that enables the research we present in the paper is an artificial, simulation-only reality gap: control software is designed on the basis of a simulation model M_x and evaluated on a second simulation model M_y, which we shall call a *pseudo-reality.*

In a first experiment, we create an artificial, simulation-only reality gap and we address the following research question:

Question Q1: Are performance drop and rank inversion to be necessarily ascribed to the fact that control software is evaluated in a context that is more complex than the one in which it is designed?

We address this question with a procedure that has the logical structure of a *reductio ad absurdum*: we show that, in the light of empirical results we produce, the complexity assumption leads to a contradiction—notably, that one model should be more *and* less complex than an other, at once. The procedure comprises two stages. In a first stage, we reproduce, with simulation-only experiments, the results of Francesca et al. (2014): they observed a rank inversion when comparing control software generated by two off-line design methods. In their experiments, the authors generated control software on the basis of a model, which we shall call M_A, and assessed its performance in reality. In our simulation-only experiment, we also generate control software on the basis of the same model M_A, but we use a second model, which we shall call M_B, as a replacement of reality, a pseudo-reality. Here, we choose M_B by trial and error so that, when used as pseudo-reality to evaluate control software generated on M_A, a rank inversion occurs between the same off-line design methods studied by Francesca et al. (2014). As designing on M_A and evaluating on M_B produces performance drop and rank inversion, if we were to accept the complexity assumption, we would conclude from this first stage that M_B is more complex than M_A. In a second stage, we invert the roles of the two models: we automatically design control software on M_B and we evaluate it on the pseudo-reality M_A. Also in this second stage, we observe performance drop and rank inversion that are qualitatively similar to those reported by Francesca et al. (2014). If we were to accept the complexity assumption, we would conclude that M_A is more complex than M_B. The clear contradiction between the conclusions of the first and second stage disproves the complexity assumption: it is not necessary to assume that the effects of the reality gap manifest because control software designed in simulation is evaluated in a more complex (pseudo-)reality.

The results of the first experiment also suggest that one could use an artificial, simulation-only reality gap to conceive a methodology to predict the robustness of design methods to the reality gap. This methodology would be able to predict which of the design methods under analysis is more likely to produce control software that will suffer from a performance drop; whether the observation of a rank inversion is to be expected; and, eventually, which of the design methods under analysis is more likely to generate control software that successfully performs a given real-world mission. The trial-and-error approach we use in the first experiment to create a pseudo-reality is clearly inappropriate to be adopted in the framework of the methodology we have in mind. The approach is indeed labor intensive and not easily reproducible. To support the methodology, we need a way to generate an appropriate pseudo-reality in an automatic and reliable way.

In a second experiment, we move a step in the direction of creating a pseudo-reality automatically. We address the following research question:

Question Q2: Can the results of the first experiment be observed with other pseudo-realities or are they an artifact of the specific pseudo-reality we employ there?

We study this question by reproducing the previous experiment, but this time using multiple evaluation models—and therefore, multiple pseudo-realities—to assess control software generated on the basis of model M_A. The different pseudo-realities are uniformly sampled around model M_A. The results of this second experiment are qualitatively similar to those observed in the first one. This suggests that a methodology for assessing the intrinsic robustness of design methods can be devised based on multiple randomly generated pseudo-realities. By intrinsic robustness we informally refer to the general ability of a design method to produce control software that transfers seamlessly from any (reasonable) model to reality, as opposite to the ability to produce control software that transfers from a specific model to reality.

The following is the structure of the paper. In Section 2, we discuss previously proposed approaches to handle the reality gap. In Section 3, we describe materials and methods adopted in two aforementioned experiments. In Sections 4 and 5, we present the two experiments and we report their results. In Section 6, we conclude the paper by illustrating our vision on how simulation-only experiments could be used to assess the intrinsic robustness of automatic design methods.

2 Related work

A number of approaches have been proposed to handle the reality gap and reduce the difference between the performance of control software assessed in simulation and in reality. However, none of these approaches has been studied in details; as noticed by Koos et al. (2013), some of them are not described with sufficient detail to be precisely reproduced; no extensive comparison has been performed; and none of them appears to be the ultimate solution to the reality gap, which remains a major issue in the off-line automatic design of control software, as pointed out by Francesca and Birattari (2016) and Silva et al. (2016) among others.

In this section, we describe their functioning and propose a taxonomy, which is summarized in Table 1. The majority of the approaches have been

Table 1 Taxonomy of the approaches to handle the reality gap. We list here the most relevant works for each group of the taxonomy. These works are described in Sections 2.1 to 2.4.

Focus on / To	Simulation models	Design methods
Reduce differences between simulation and reality	Miglino et al. (1995) Jakobi et al. (1995) Bongard and Lipson (2004) Zagal et al. (2004)	Koos et al. (2013)
Enhance robustness of control software	Jakobi (1997, 1998) Boeing and Braunl (2012)	Floreano and Mondada (1996) Urzelai and Floreano (2000) Floreano and Urzelai (2001) Francesca et al. (2014, 2015)

proposed in the context of automatic design of behaviors for single robots. However, they are general enough to be applied to multi-robot systems and robot swarms.

Some approaches aim at reducing the differences between simulation and reality as much as possible (Miglino et al. 1995; Jakobi et al. 1995; Bongard and Lipson 2004; Zagal et al. 2004; Koos et al. 2013), whereas others aim at making control software robust to these differences (Floreano and Mondada 1996; Jakobi 1997, 1998; Boeing and Braunl 2012; Francesca et al. 2014, 2015). The first group of approaches appear to be driven by the hypothesis that the more accurate the simulation, the smoother the transition to reality; the second one, by the hypothesis that a fully accurate simulation is impossible and overfitting is always a risk. The two groups can be further detailed according to which element of the off-line automatic design process is targeted by the approach: either the simulation models (Miglino et al. 1995; Jakobi et al. 1995; Jakobi 1997; Bongard and Lipson 2004; Zagal et al. 2004; Boeing and Braunl 2012) or the design method (Floreano and Mondada 1996; Koos et al. 2013; Francesca et al. 2014, 2015). Among the methods that focus on the simulation models, some aim at increasing their realism—see Section 2.1. Others use them to enhance the robustness of the design process—see Section 2.2. Among the methods that focus on the design process, some aim at conceiving a design process that avoids exploiting features of simulation that do not match reality—see Section 2.3. Others aim at making the design process intrinsically more robust—see Section 2.4.

2.1 Focus on simulation models to reduce differences between simulation and reality

Miglino et al. (1995) proposed guidelines for conceiving simulations models such that they represent reality as accurately as possible. The authors recommend to (i) use real-world data sampled from the robot's sensors and actuators; and (ii) add noise to models to account for imperfect sensing and actuation. Furthermore, if a performance drop between simulation and reality is anyway observed, the authors suggest to continue the design process on physical robots for a few iterations. They demonstrated their protocol by generating control software for a Khepera robot (Mondada et al. 1994) on an obstacle avoidance task. Subsequently, Jakobi et al. (1995) automatically generated control software that behaved almost identically in simulation and in reality. Experiments were performed with a single Khepera robot on two tasks: obstacle avoidance and light seeking. According to the authors, the key to this almost perfect transition from simulation to reality is an appropriate fine-tuning of the levels of noise within the simulation models. Since the publication of the work of Jakobi et al. (1995), incorporating sampled data and fine-tuning the noise levels have become common practice in the conception of simulation models—for a review of the relevant literature, see Silva et al. (2016).

Bongard and Lipson (2004) proposed an approach they called *estimation-exploration*, which consists in the simultaneous evolution of control software and simulation models. The authors generated control software for a quadrupedal robot so that it could walk the longest distance possible. The behavior produced crossed the reality gap successfully. Zagal et al. (2004) proposed a similar approach they named *back to reality*. The method was validated with a Sony AIBO robot on two tasks: gait optimization and ball-kicking (Zagal and Ruiz-Del-Solar 2007). The two aforementioned methods differ in the data used to improve the simulation models: *estimation-exploration* uses samples from the physical robot; whereas *back to reality* uses the performance drop experienced in reality.

2.2 Focus on simulation models to enhance robustness of control software

Jakobi (1997, 1998) proposed the *radical envelope-of-noise* hypothesis: in order to cross the reality gap satisfactorily, random variation should be applied to all aspects of the simulation. In addition, the author suggested to restrict to *minimal simulations*: simulators should reproduce only the elements of reality that are strictly needed to generate the desired behavior. Jakobi demonstrated the validity of his proposal with three experiments involving different robotic platforms and a fourth one in computer vision. He automatically designed behaviors for: (i) a Khepera robot—turn left or right at the end of a corridor, depending on the position of a light; (ii) a gantry robot—recognize shapes; (iii) an octopod robot—walk efficiently and avoid obstacles; and (iv) a computer vision system—track moving objects through a camera.

Although they do not directly cite the work of Jakobi, Peng et al. (2018) and Andrychowicz et al. (2018) reintroduced his idea of applying random variation in the simulation models. They proposed a technique called *domain randomization* in the context of the application of reinforcement learning to robotics. Specifically, they showed that, thanks to this technique, control software developed in simulation can be successfully ported to physical robots: Peng et al. (2018) applied the technique to a robotics arm that was required to push an object; Andrychowicz et al. (2018) to a manipulator that was required to rotate an object.

Boeing and Braunl (2012) simultaneously employed two simulators in the design process to automatically generate control software for the Mako robot, an autonomous underwater vehicle. The authors showed that, via an experiment in which the Mako robot has to follow a wall, increasing the variance of the condition experienced in the design process leads to the generation of control software that crosses the reality gap satisfactorily.

2.3 Focus on design methods to reduce differences between simulation and reality

Koos et al. (2013) proposed what they called *the transferability approach*. In this approach a bi-objective algorithm optimizes: (i) a mission-dependent performance metric; and (ii) a measure of disparity between performance in simulation and in reality. The approach aims at constraining the design process to generate control software that only exploits features of the simulator that accurately model reality. The approach uses a model to estimate the difference between performance in simulation and reality. To build and update it, periodic robot evaluations of control software generated by the design process are required. The authors demonstrated their approach in two experiments involving different robotic platforms. In the first one, the navigation experiment of Jakobi (1997) was reproduced with an e-puck robot (Mondada et al. 2009); the transferability approach was compared to noise-based approach of Jakobi (1997, 1998) described in Section 2.2. The instances of control software generated by the transferability approach crossed the reality gap more satisfactorily than those generated following the noise-based approach of Jakobi. In the second experiment, the authors generated control software for a quadrupedal robot so that it could walk as much distance as possible. In addition, the authors also performed simulation-only experiments to further study the properties of the approach. To do so, they created an artificial reality gap between a simple simulator and a more accurate one, with the latter playing the role of reality.

2.4 Focus on design methods to enhance robustness of control software

Floreano and Mondada (1996) deviated from the classical implementation of neuro-evolutionary robotics to propose an approach based on adaptive neurocontrollers called *plastic controllers*. In the approach, the update rule of each neuron and its parameter (learning rate) are selected off-line in simulation. The synaptic weights of the resulting network are then adapted on-line while the robot operates in the target environment. Plastic controllers, evolved to control a Khepera robot in a light switching task, have shown to cross the reality gap satisfactorily (Urzelai and Floreano 2000; Floreano and Urzelai 2001).

Francesca et al. (2014) have also deviated from traditional neuro-evolutionary robotics. They started from the conjecture that neuro-evolutionary robotics is particularly affected by the effects of the reality gap due to the excessive representational power of neural networks. As a result, neuro-evolutionary robotics is likely to overfit the conditions experienced during the design process. Guided by the notion of bias-variance tradeoff (Geman et al. 1992), the authors developed two design methods that produce control software with restricted representational power: they are the result of the combination of predefined low-level behaviors. The two methods—AutoMoDe-`Vanilla` and AutoMoDe-`Chocolate`—have shown to produce control software that crosses the reality gap more satisfactorily than those produced by `EvoStick`, an implementation of the traditional neuro-evolutionary robotics (Francesca et al. 2014, 2015). These results were further confirmed by follow up studies (Kuckling et al. 2018; Hasselmann et al. 2018b).

3 Materials and methods

In this section, we describe the robots and the simulator used to simulate them, the off-line automatic design methods, and the missions considered in our experiments.

3.1 Simulated robots

We consider a version of the e-puck robot whose capabilities have been augmented via several extension modules (Mondada et al. 2009; Garattoni et al. 2015). The e-puck is a small, circular, and two-wheeled modular robot. In the paper, we adopt the following extension modules: (i) the Overo Gumstix, which allows Linux to be run on the robot; (ii) the ground sensor module, which allows the robot to detect the gray-level color of the floor situated under the front part of its body; and (iii) the range-and-bearing module (Gutiérrez et al. 2009), which allows the robot to detect neighboring peers located within an approximate range of 0.7 m, and to estimate their relative position.

To simulate e-puck robots, we use ARGoS (Pinciroli et al. 2012), an open source multi-engine simulator, specifically designed to simulate robot swarms. To model the robot-robot and robot-environment interactions, we use the integrated 2D-dynamics physics engine based on the Chipmunk library. ARGoS is conceived such that control software that runs in the simulator can be directly cross-compiled, ported, and executed on the robots without any modification.

We automatically generate control software that has access to a subset of the robot sensors and actuators, each of them abstracted by appropriate variables. These variables are updated at every control cycle (that is, every 100 ms). A formal definition of the sensors and actuators, and the corresponding variables is given by the reference model RM 1.1 (Hasselmann et al. 2018a) of Table 2.

Table 2 Reference model RM1.1 (Hasselmann et al. 2018a) of the extended version of the e-puck robot.

sensor/actuator	variables
proximity	$prox_i \in [0,1]$, with $i \in \{0,1,...,7\}$
light	$light_i \in [0,1]$, with $i \in \{0,1,...,7\}$
ground	$ground_i \in \{white, gray, black\}$, with $i \in \{0,1,2\}$
range-and-bearing	$n \in \{0,1,...,19\}$
	$V_d \in ([0.5,20], [0,2\pi]\,\text{rad})$
wheels	$v_l, v_r \in [-0.12, 0.12]\text{ms}^{-1}$

The control software uses eight infrared sensors situated all around the robot to detect obstacles ($prox_i$) and to measure the ambient light ($light_i$). It uses three ground sensors to detect the gray level of the floor ($ground_i$). It also uses the range-and-bearing module to know the relative position of neighboring robots. The relative range r_m and bearing $\angle b_m$ of the perceived robots are aggregated into a vector $V_d = \sum_{m=1}^{n} (1/(1+r_m), \angle b_m)$. In addition to this vector V_d, the control software has access to the number n of robots perceived. Finally, the control software sets the velocity of the left and right wheels (v_l and v_r), which dictate the displacement of the robot.

In simulation, sensor readings and actuator values are affected by noise that can be fine-tuned via parameters. For the proximity sensor, a uniform white noise is applied to the sensor readings and parameter p_u controls the level of noise: at every control cycle, a value in the range $[-p_u, p_u]$ is uniformly sampled and added to the reading. This also holds for the light and ground sensors. For the range-and-bearing module, the robot fails to estimate the relative position of a neighboring peer with probability p_{fail}. Finally, a Gaussian white noise is applied to the velocities of each wheel and parameter p_g controls the level of noise: every control cycle, for each wheel, a value is sampled according to a Gaussian distribution with mean 0 and standard deviation p_g, and added to the velocity.

3.2 Design methods

In the experiments of the paper, we consider three previously proposed off-line design methods: `EvoStick` (Francesca et al. 2014); and two instances of AutoMoDe, `Vanilla` (Francesca et al. 2014) and `Chocolate` (Francesca et al. 2015). We briefly describe these methods, and refer the reader to the original publications for more details.

`EvoStick` is an implementation of the traditional neuro-evolutionary robotics approach: it generates control software in the form of a neural network, and uses an evolutionary algorithm to optimize its parameters. The neural network is fully connected, feedforward, and comprises 24 input nodes, dedicated to the readings of the robot sensors. Inputs are directly connected to 2 output nodes, dedicated to the actuators. The input and output nodes are defined on the basis of the reference model RM 1.1 of Table 2, that is: 8 input nodes for the proximity sensors $prox_i$, 8 input nodes for the light sensors $light_i$, 3 input nodes for the ground sensors $ground_i$, 5 input nodes for the range-and-bearing module, and 2 output nodes for the velocities of the wheels v_l and v_r. The 5 input nodes dedicated to the range-and-bearing module are organized as follows: 4 for the scalar projections of the vector V_d on four unit vectors, and 1 for the number n of robots perceived. Because the topology of the neural network is fixed, the role of the evolutionary algorithm is to fine-tune the weights of the connections between the input and output nodes. There is a total of 50 weights, each in the range $[-5.0, 5.0]$. The evolutionary algorithm used by `EvoStick` has a population size of 100 individuals and it evaluates each individual 10 times at each iteration.

AutoMoDe is an approach to the automatic design that generates control software by combining predefined modules. In `Vanilla` and `Chocolate`, the control software produced is in the form of a probabilistic finite-state machine. The two methods share the same set of modules: 6 low-level behaviors and 6 transition conditions. All the modules have been conceived on the basis of the reference model RM 1.1 of Table 2, and some of them have parameters that regulate their functioning. The two methods also share the same design space that they explore: it comprises all the probabilistic finite-state machines that can be generated out of the given set of modules, with a maximum of 4 states and a maximum of 4 outgoing edges per state. The methods automatically associate to each state one of the 6 low-level behaviors, and to each edge one of the 6 conditions, which will determine whether the transition between the states connected by the corresponding edge should happen or not. `Vanil-la` and `Chocolate` only differ in the optimization algorithm they employ to select, combine, and fine-tune the modules: the former uses F-race (Birattari et al. 2002; Birattari 2009), the latter uses Iterated F-race (López-Ibáñez et al. 2016).

3.3 Missions

We consider two missions: AGGREGATION and FORAGING. These missions must be performed by a swarm comprising 20 e-puck robots in a dodecagonal arena of $4.91\,\text{m}^2$ within a time of 250 s. At the beginning of an experimental run, we randomly position and orient the robots uniformly in the arena. Figure 1 depicts the simulated arenas for each mission.

3.3.1 AGGREGATION

The robots must aggregate on one of the two black zones, namely a or b. The size and position of the black zones are given in Figure 1. The objective function, to be maximized, is

$$F_{\text{A}} = \max(N_a, N_b)/N,$$

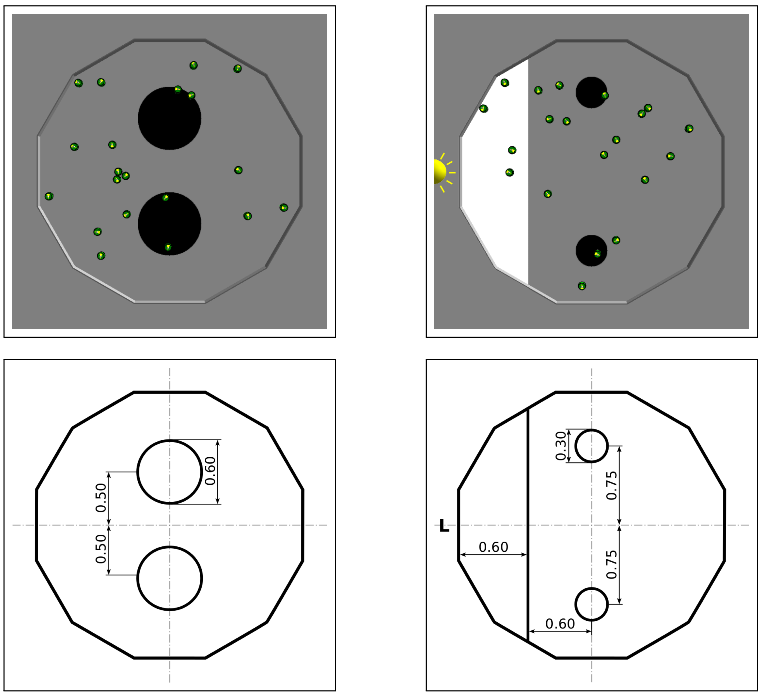

Fig. 1 ARGoS3 representations and technical diagrams of the arenas: AGGREGATION (*left*) and FORAGING (*right*). Measures are expressed in meters. For FORAGING, a light is placed behind the nest so that it is visible from everywhere in the arena.

where N_a and N_b are the number of robots located on the zones a and b, respectively; and N is the total number of robots in the swarm. The objective function is computed at the end of an experimental run.

3.3.2 FORAGING

We consider an idealized version of foraging in which the robots must retrieve to the nest as many items as possible from either of two sources. The food sources are represented by black circular zones, and the nest by a white zone. A robot is considered to have picked up or dropped an item when it enters a black zone or the white zone, respectively. A robot can only carry one object at a time. A light is placed behind the nest at a height of 0.75 m so that it is visible from everywhere in the arena. The size and position of sources and nest are given in Figure 1.

The objective function, to be maximized, is

$$F_{\mathrm{F}} = I,$$

where I is the number of items retrieved.

3.4 Statistics

Throughout the paper, we present the performance of control software with box-and-whiskers boxplots, and statements such as "method 1 is significantly better/worse than method 2" imply that significance has been assessed via a paired Wilcoxon signed rank test, with confidence of at least 95%. Moreover, to estimate the performance drop experienced in pseudo-reality, we present 95% confidence intervals, also computed via the paired Wilcoxon signed rank test. Finally, to study the statistical relationship between performance drop experienced in pseudo reality and (i) performance in simulation, or (ii) width of the artificial reality gap, we use the Pearson correlation test. The simulation models used, the control software generated, and the experimental data collected across the two experiments are available online as supplementary material (Ligot and Birattari 2019).

4 Experiment 1: design model and pseudo-reality are fixed

In this experiment, we address question Q1: we show that the complexity assumption leads to a contradiction and it is therefore unnecessary to assume that the effects of the reality gap are due to reality being more complex than simulation models—or equivalently, to simulation models being too simplistic. To do so, in a first stage that we shall call S_{AB}, we reproduce in simulation the experiments of Francesca et al. (2014): they used `EvoStick` and `Vanilla` to automatically generate control software on the basis of a simulation model that we refer to as model M_A; they then evaluated this control software on physical robots. They observed a performance drop that lead to a rank inversion between the two methods: `EvoStick` outperformed `Vanilla` in simulation, but `Vanilla` outperformed `EvoStick` in reality. Here, as Francesca et al. (2014) did, we generate control software on the basis of M_A but, differently from them, instead of evaluating it on physical robots, we use a pseudo-reality—that is, a second model that we shall call M_B. We choose M_B via trial and error so as to obtain a performance drop and a rank inversion that are qualitatively similar to those observed by Francesca et al. (2014). If we were to accept the complexity assumption, we would conclude that M_B is more complex than M_A. In a second stage that we shall call S_{BA}, we invert the role of the two models: control software is designed on M_B and tested on the pseudo-reality M_A. If, also in the second stage S_{BA} we observe performance drop and rank inversion, and if we were to accept the complexity assumption, we would conclude that M_A is more complex than M_B. Together with the result of stage S_{AB}, this would be a contradiction.

As a further contribution we make in this section, we analyze the correlation between performance on the model used for the design and performance drop experienced in pseudo-reality. This analysis is enabled by the device of the simulation-only reality gap: thanks to the pseudo-reality, we can perform a large amount of evaluations that we could not perform on physical robots.

Table 3 The two models of the e-puck robot. We report the parameters of the noise applied to the sensor readings and actuator values, and refer the reader to Section 3.1 for the details. The values for the proximity, light, and ground sensors determine the upper bounds of symmetric ranges for uniform white noises. The value for the range-and-bearing sensor is the probability of failing to estimate the relative position of a neighboring peer. The value for the wheels actuator is the standard deviation of a Gaussian white noise with mean 0.

sensor/actuator	M_A	M_B
proximity	0.05	0.05
light	0.05	0.90
ground	0.05	0.05
range-and-bearing	0.85	0.90
wheels	0.05	0.15

In the following, we characterize M_A and M_B by giving the values of their parameters, provide details on the experimental protocol, report the results, and discuss them.

Models. The parameters of models M_A and M_B are given in Table 3. Model M_A was used by Francesca et al. (2014) to simulate e-puck robots in the context of the automatic generation of their control software. The authors conceived M_A following the best practice in off-line automatic design: they injected noise in simulation according to distributions identified on the basis of empirical data (Miglino et al. 1995; Jakobi et al. 1995). We conceived model M_B via trial and error with the objective of obtaining rank inversion when using M_B as pseudo-reality to evaluate control software generated on the basis of M_A.

Protocol. In each stage S_{xy}—with x and y being A and B, or B and A—we used `EvoStick`, `Vanilla`, and `Chocolate` to automatically generate control software for AGGREGATION and FORAGING. In each stage, control software is automatically generated on the basis of model M_x, and its performance is assessed on model M_y. To measure the performance drop, we also record the performance on model M_x.

Each design method is executed 20 times with a design budget of 200 000 simulation runs. As an individual execution produces an instance of control software, each design method yields a total of 20 instances per mission and stage. All instances of control software are evaluated 20 times on model M_x and 20 times on model M_y.

Results. In both stages S_{AB} and S_{BA}, we observe a noticeable performance drop for `EvoStick`, which determines a rank inversion—see Figure 2 and 3. Indeed, when comparing the three design methods on the basis of their performance assessed on model M_x—that is, the design model—`EvoStick` performs significantly better than both `Vanilla` and `Chocolate`. However, when comparing the methods on the basis of their performance assessed on model M_y—that is, the pseudo-reality—both `Vanilla` and `Chocolate` perform significantly better than `EvoStick`.

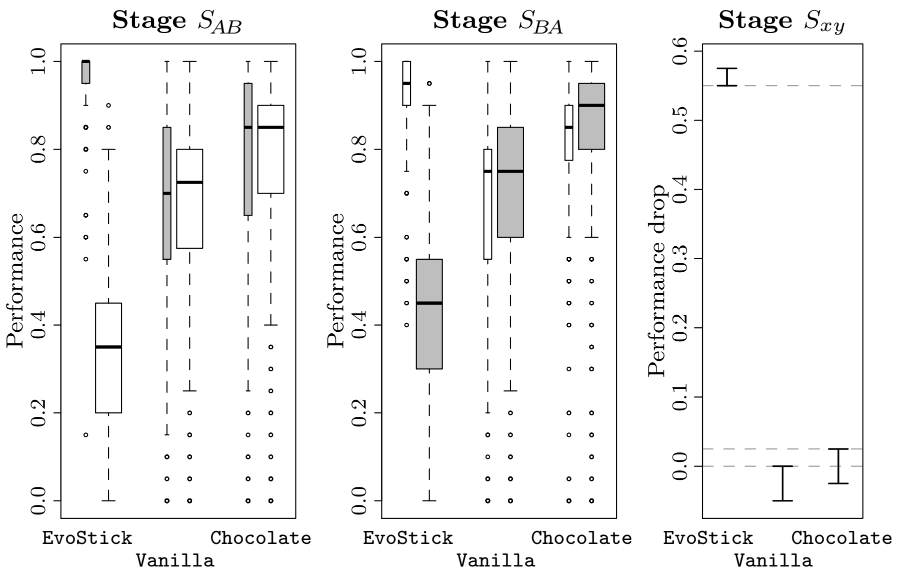

Fig. 2 AGGREGATION. *Left* and *center*: Performance in each stage—the higher, the better. Narrow boxes represent the performance assessed on the design model; wide boxes represent the performance assessed on the pseudo-reality. Gray boxes represent performance assessed on M_A; white boxes represent performance assessed on M_B. *Right*: Performance drop, aggregated across the two stages—the lower, the better. The segments represent the upper and lower bounds on the performance drop experienced in pseudo-reality—bounds are computed using Wilcoxon statistics, at 95% confidence.

For AGGREGATION, a same instance of control software generated by `EvoStick` behaves in a qualitatively different way in M_x and in M_y. In M_x, the robots tend first to navigate along the walls of the arena, and then to converge towards peers that are already located on one of the black zones. Once robots are on a black zone, they remain there, spinning in place. In M_y, the robots navigate in small circles without ever converging towards their peers. Eventually, they fail to find the black zones and to aggregate therein. Quantitatively, the performance drop that affects the control software generated by `EvoStick` is of at least 0.55—see Figure 2 (*right*). On the other hand, the control software produced by `Vanilla` and `Chocolate` behaves in a qualitatively similar way in M_x and M_y: the robots converge towards their peers to form clusters, and tend to remain on a black zone once they reach one. Quantitatively, the performance drop is much smaller than the one experienced by `EvoStick`: at most 0.00 and 0.02 for `Vanilla` and `Chocolate`, respectively—see Figure 2 (*right*).

The results for FORAGING are similar to those of AGGREGATION. Indeed, the behavior of a same instance of control software generated by `EvoStick` is qualitatively different in M_x and M_y. In M_x, the robots navigate in circles of radius approximately equal to half the one of the whole arena: they follow the walls around the nest and cross the arena so that they navigate on at

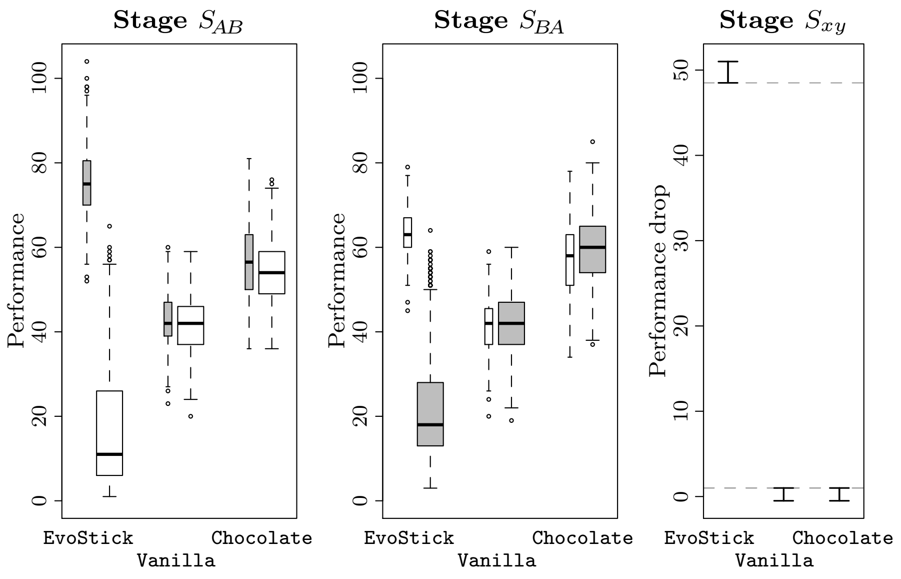

Fig. 3 FORAGING. *Left* and *center*: Performance in the two stages—the higher, the better. *Right*: Performance drop, aggregated across the two stages—the lower, the better. See caption of Figure 2 for a detailed description of the figures.

least one of the two food sources. In M_y, the robots navigate in much smaller circles that often do not cross any of the food sources, which results in a drop of performance of at least 48 items—see Figure 3 (*right*). Contrarily to `EvoStick`, `Vanilla` and `Chocolate` produce control software that behaves similarly in M_x and M_y: the robots randomly explore the arena and, as soon as they encounter one of the food sources, they navigate towards the light to reach the nest. `Chocolate` performs significantly better than `Vanilla` because the instances of control software it produces only explore the gray area of the arena: while searching for a food source, robots do not enter the nest. This increases the rate at which food sources are found. The performance drop experienced by `Vanilla` and `Chocolate` is at most 1 item—see Figure 3 (*right*). Videos illustrating the typical behaviors displayed by the control software generated by the three methods are available as supplementary material (Ligot and Birattari 2019).

The results also show a positive correlation between the performance drop experienced in pseudo-reality and the performance assessed on the design model—see Figure 4. This holds true for the three methods. Nonetheless, some difference should be noticed. For `EvoStick` and in the two missions, observations are concentrated in the top left quarter, which indicates a high performance on M_x, but a large performance drop. On the other hands, for `Vanilla` and `Chocolate`, the observations are close to the center of the figure, which indicates a relatively lower performance on M_x with respect to the one of `EvoStick`, but a performance drop that is centered around zero. For both missions, the cloud of observations for `Chocolate` is shifted to the right

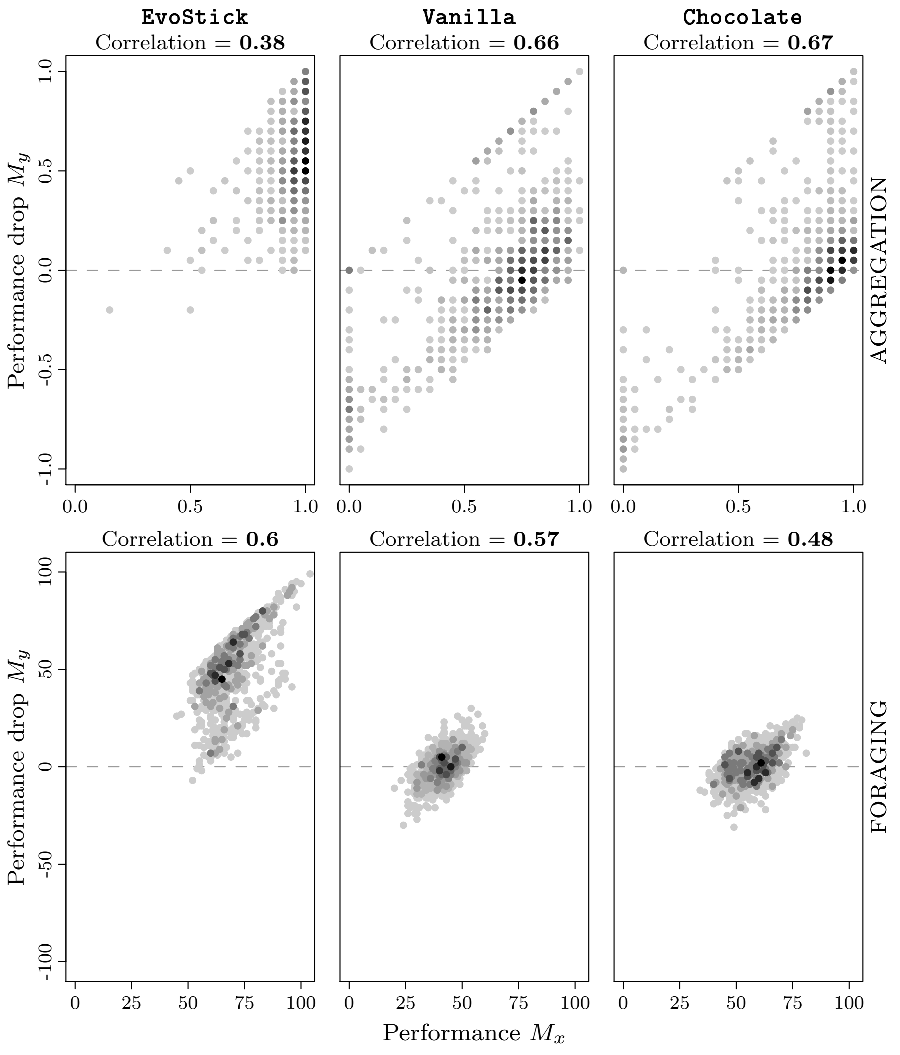

Fig. 4 Correlation between performance drop experienced in pseudo-reality and performance assessed in M_x. The gray level of a point indicates its frequency of observation: the darker, the higher the frequency. All correlations are significantly different from 0 with confidence of at least 95%. For AGGREGATION, the organization of the points in columns is due to the quantum of F_{A} equal to $0.05 = 1/20$, which corresponds to a difference of one robot, out of the twenty comprised in the swarm, on the most populated black zone—see Section 3.3.1.

with respect to the one of `Vanilla` as `Chocolate` obtains a relatively higher performance.

5 Experiment 2: design model is fixed, pseudo-reality is sampled

In this experiment, we address question Q2: we evaluate automatically generated control software on a range of pseudo-realities. In Section 4, we were able to find, by trial and error, a pseudo-reality that produced a performance drop and rank inversion that are similar to those previously observed when evaluating automatically generated control software on physical robots (Francesca et al. 2014). Here, we investigate whether the above only holds for that specific pseudo-reality, or whether it is a more general fact holding for multiple pseudo-realities. To do so, we use several pseudo-realities to assess the performance of control software generated on the basis of M_A, with each pseudo-reality being a model uniformly sampled from a predefined set of models. In other words, we create multiple artificial reality gaps between pairs of models: M_A and a randomly sampled one. Because some of the sampled models might be too similar to M_A to yield a noticeable performance drop, we do not expect that every single model sampled can be used by itself as a pseudo-reality on which we can observe results analogous to those observed in the first experiment. Nonetheless, we expect that, by evaluating control software on a sufficiently large number of such models, and by aggregating the results across all of them, we could obtain a correct overall picture of which methods are more likely to suffer from a performance drop and of whether a rank inversion should be expected.

Furthermore, we propose multiple measures to quantify the *width* of the artificial reality gap between a pair of models. By width of a reality gap, we mean some measure of the difference between the model on which control software is designed and the (pseudo-)reality in which it is evaluated. Eventually, our long-term goal would be to have a measure of this width that has predictive capability: this measure should allow one to predict the performance drop that a given design method would experience when crossing a reality gap of that width. Because all models considered in this paper differ only by the values of five parameters, each model is fully identified by a vector in a five dimensional space. Under this condition, we conjecture that the width of an artificial reality gap can be quantified by an appropriate distance measure between the vectors that identify the models involved in the artificial reality gap itself. We consider a number of distance measures between two vectors and the difference of a number of vector norms. We study their Pearson correlation with the performance drop experienced when designing control software on the model identified by one of the two vectors and evaluating it on the model identified by the other one.

In the following, we describe the set of models from which the pseudo-realities are sampled, detail the protocol followed, define the measures of difference between models, and report and discuss the results.

Models. Model M_A was used by Francesca et al. (2014) and in the previous experiment. Model M_B is sampled from a predefined set of models that we conceived such that (i) it comprises both model M_A and the model M_B used

Table 4 Ranges of possible values for the model M_B. The values of model M_A are reported here for reference. See caption of Table 3 for a description of the parameters.

sensor/actuator	ranges for M_B	M_A
proximity	$[0.00, 0.10]$	0.05
light	$[0.00, 1.50]$	0.05
ground	$[0.00, 0.10]$	0.05
range-and-bearing	$[0.70, 1.00]$	0.85
wheels	$[0.00, 0.20]$	0.05

in the first experiment, and (ii) it contains models that are more noisy than M_A and models that are less noisy. The set of models from which M_B is sampled is given in Table 4, together with the parameters of M_A, which are repeated here for the convenience of the reader.

Protocol. We consider two stages: S_{AB} and S_{BA}. In stage S_{AB}, an instance of control software is automatically generated on the basis of model M_A, and it is evaluated on M_A itself and on a pseudo-reality: a model M_B that is uniformly sampled from the range defined in Table 4. This process is repeated 20 times, which therefore results in the generation of 20 instances of control software on the basis of M_A and in the sampling of 20 models M_B. In stage S_{BA}, the same 20 models M_B sampled in stage S_{AB} are used to automatically generate control software which is then evaluated on M_B itself and on a pseudo-reality, whose role here is played by M_A.

We consider the missions AGGREGATION and FORAGING and two design methods: `EvoStick` and `Chocolate`. In this second experiment, we drop `Vanilla` because the first experiment showed that its results are very similar to those of `Chocolate`, which performs slightly better. Each design process is granted a design budget of 200 000 simulation runs. The 20 uniformly sampled models M_B are the same in each stage, for each mission, and for each automatic design method.

Measuring the width of an artificial reality gap. As we consider models that are fully and uniquely identified by five noise parameters, they can be represented as vectors in the five dimensional space. In this space, for a generic vector $\mathbf{v}$, we consider its norms ℓ^1, ℓ^2, and ℓ^∞, where

$$\ell^n = \|\mathbf{v}\|_n = \sqrt[n]{\sum_{i=1}^{5} |v_i|^n}.$$

In the case of the ℓ^∞ norm, by taking the limit of the above, we have $\ell^\infty = \max_i |v_i|$. Because each component of a vector defines the amount of simulation noise concerning a specific sensor or actuator, the higher the norm, the higher the overall amount of simulation noise.

As possible measures of the width between two models M_A and M_B, we consider the differences $\|\mathbf{b}\|_n - \|\mathbf{a}\|_n$ between the ℓ^n norm of their corresponding vectors $\mathbf{a}$ and $\mathbf{b}$, with $n \in \{1, 2, \infty\}$. Moreover, we consider the ℓ^n norm of their difference $\|\mathbf{b} - \mathbf{a}\|_n$, also with $n \in \{1, 2, \infty\}$. Differences of norms can be negative, with a negative value indicating that the evaluation model is less noisy than the design one. Moreover, differences of norms are anticommutative: the width of the gap from M_A to M_B and the one from M_B to M_A have the same absolute value but opposite sign. It should be noted that a null value of a difference of norms does not ensure that the two models are identical. This is a weakness of differences of norms as a measure of the width of the reality gap because one would expect that a zero measure indicates that the gap is null and therefore design and evaluation models are the same. On the other hand, norms of difference are non-negative and commutative: the larger the norm, the wider the gap; and the width of the gap from M_A to M_B and the one from M_B to M_A are equal. As an alternative to measuring the width of an artificial reality gap, one could measure how similar the design and evaluation models are. For example, this could be done using the *cosine similarity* of models $\mathbf{a}$ and $\mathbf{b}$:

$$\frac{\sum_i a_i b_i}{\sqrt{\sum_i a_i^2}\,\sqrt{\sum_i b_i^2}}.$$

The cosine similarity is commutative and, in a positive space such as the one considered here, is bounded between 0 and 1: zero indicates that the two vectors are orthogonal, a one indicates that they have the same orientation. Hence, the lower the cosine similarity, the wider the (pseudo-)reality gap.

For all the aforementioned measures, we consider both the unnormalized and normalized versions. We normalize each term v_i of a vector $\mathbf{v}$ with respect to the lower and upper bounds of its range—L_i and U_i, respectively. Ranges are given in Table 4. In particular, when a vector $\mathbf{m} = \{m_1, ..., m_5\}$ represents a model, it is normalized into a vector $\overline{\mathbf{m}} = \{\overline{m}_1, ..., \overline{m}_5\}$ where:

$$\overline{m}_i = \frac{m_i - L_i}{U_i - L_i} \qquad \text{for } i \in \{1, ..., 5\}. \tag{1}$$

Each component of $\overline{\mathbf{m}}$ ranges therefore between 0 and 1. When a vector $\mathbf{d}$ represents a difference $\mathbf{b} - \mathbf{a}$, it is normalized into $\overline{\mathbf{d}}$ where:

$$\overline{d}_i = \begin{cases} \dfrac{b_i - a_i}{U_i - a_i}, & \text{if } b_i >= a_i; \\ \dfrac{b_i - a_i}{a_i - L_i}, & \text{if } b_i < a_i; \end{cases} \qquad \text{for } i \in \{1, ..., 5\}. \tag{2}$$

Each component of $\overline{\mathbf{d}}$ ranges therefore between -1 and 1.

Results. In both stages and for both missions considered, a rank inversion occurred between `EvoStick` and `Chocolate`—see Figure 5 and 6. Indeed, the control software generated by `EvoStick` performs significantly better than the

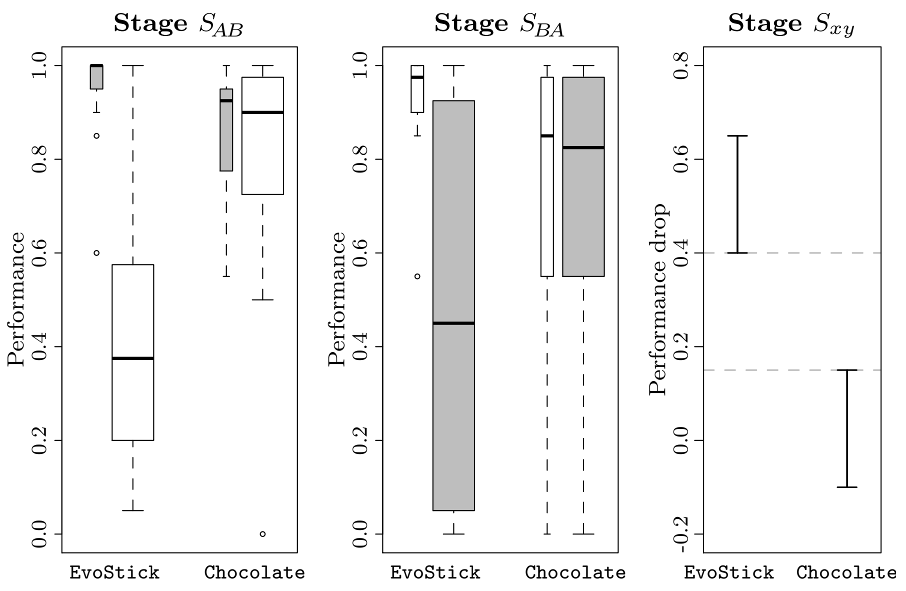

Fig. 5 Results for AGGREGATION. Narrow boxes represent the performance assessed on the model used as design context; wide boxes represent the performance assessed in pseudo-reality. Gray boxes represent performance assessed on model M_A; white boxes represent performance assessed on model M_B.

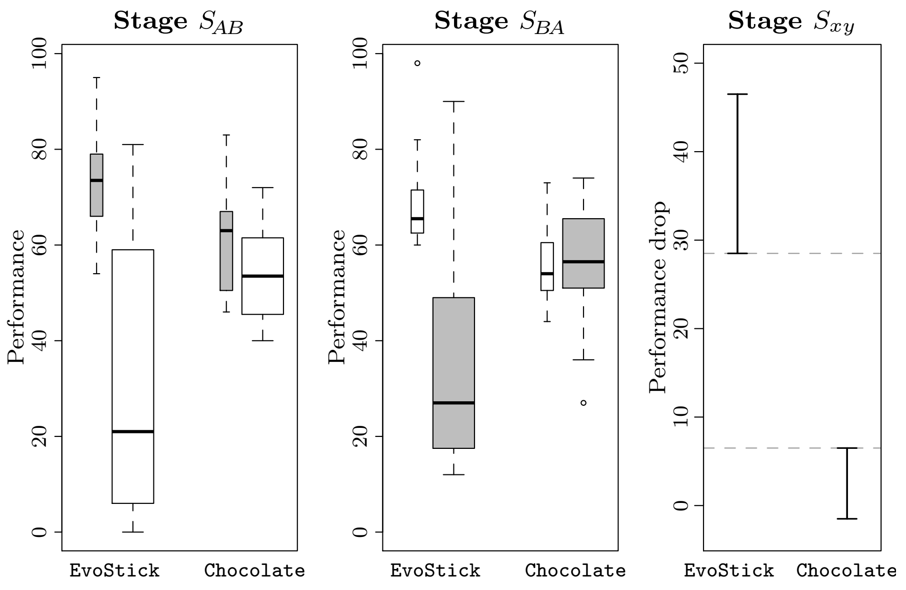

Fig. 6 Results for FORAGING. See caption of Figure 5 for a detailed description.

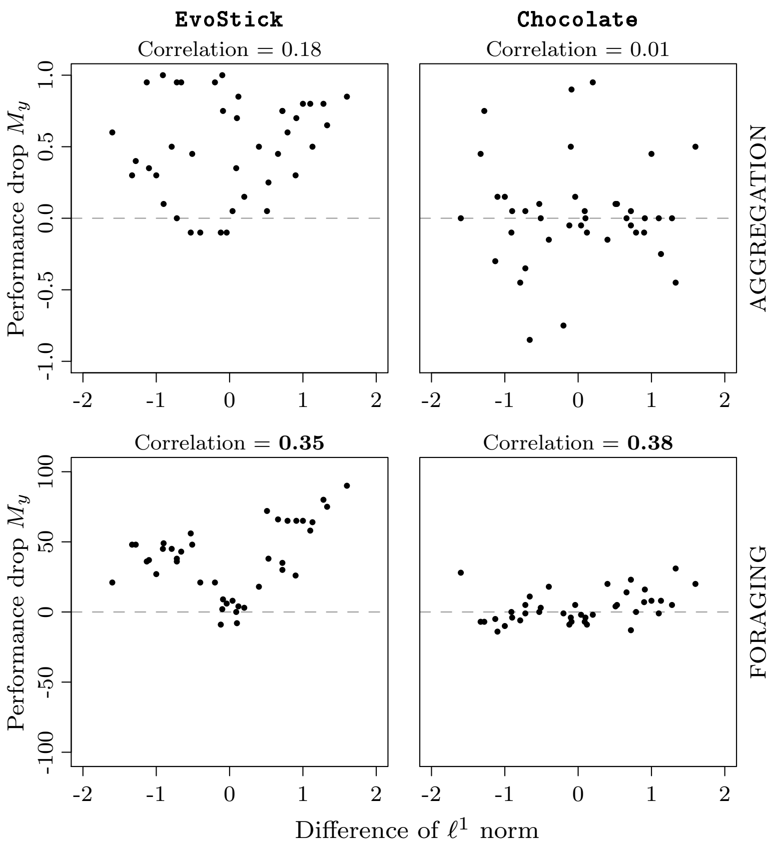

Fig. 7 Pearson correlation between performance drop and difference of ℓ^1 norms of the models used for design and evaluation. Performance drop is aggregated across stages S_{AB} and S_{BA}. Boldface values indicate that the correlation is significantly different from 0 with confidence of at least 95%.

one of `Chocolate` when the evaluation is performed on the design model, but the one of `Chocolate` performs significantly better than that of `EvoStick` when the evaluation is performed in pseudo-reality.

Concerning the correlation between performance drop and the width of the artificial reality gap, all measures we considered, both in the normalized and unnormalized versions, provided similar results. In the following, we only report the results concerning the unnormalized version of the ℓ^1 norm of difference, the difference of ℓ^1 norms, and the cosine similarity because trends are more apparent there. The remaining results are reported as supplementary material (Ligot and Birattari 2019). Figures 7 and 8 show the correlation between the performance drop—aggregated across stages S_{AB} and S_{BA}—and the difference of norms and the norm of differences. In all figures, a disparity can be noticed between the performance drop experienced by `EvoStick`

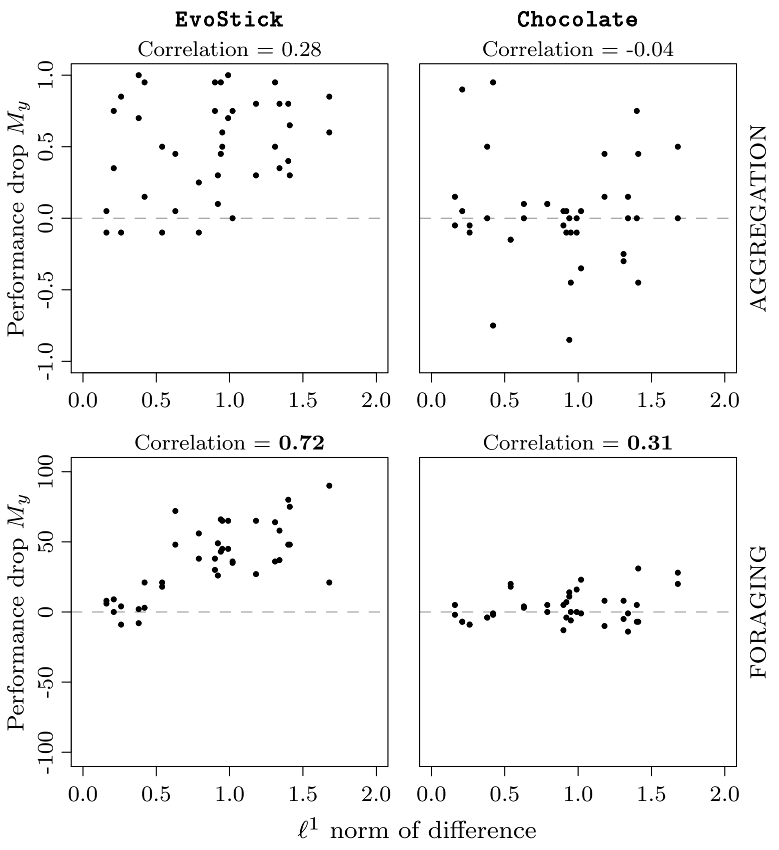

Fig. 8 Pearson correlation between performance drop and ℓ^1 norm of the difference between models used for design and evaluation. See caption of Figure 7 for more details.

and `Chocolate`: for `EvoStick`, most of observations are above the zero-drop line; for `Chocolate` they are more equally spread above and below. Moreover, the results for AGGREGATION fail to show a clear trend. For FORAGING, differences of norms display a V-shape pattern. This is particularly evident for `EvoStick`—see Figure 7 (*bottom left*). This is possibly due to the anticommutative nature of the measure. Because of the V-shape pattern, the Pearson correlation fails to be informative on the actual correlation between width of the reality gap and performance drop. Nonetheless, the difference of norms has some merits as it highlights an interesting fact: for `EvoStick`, the performance drop grows with the absolute value of the width. In particular, we can observe that for negative width—that is, when the evaluation model is less noisy than the design one—we register a noticeable performance drop. This further corroborates our conjecture that performance drop is to be explained as a sort of overfitting of the conditions experienced during the design, rather

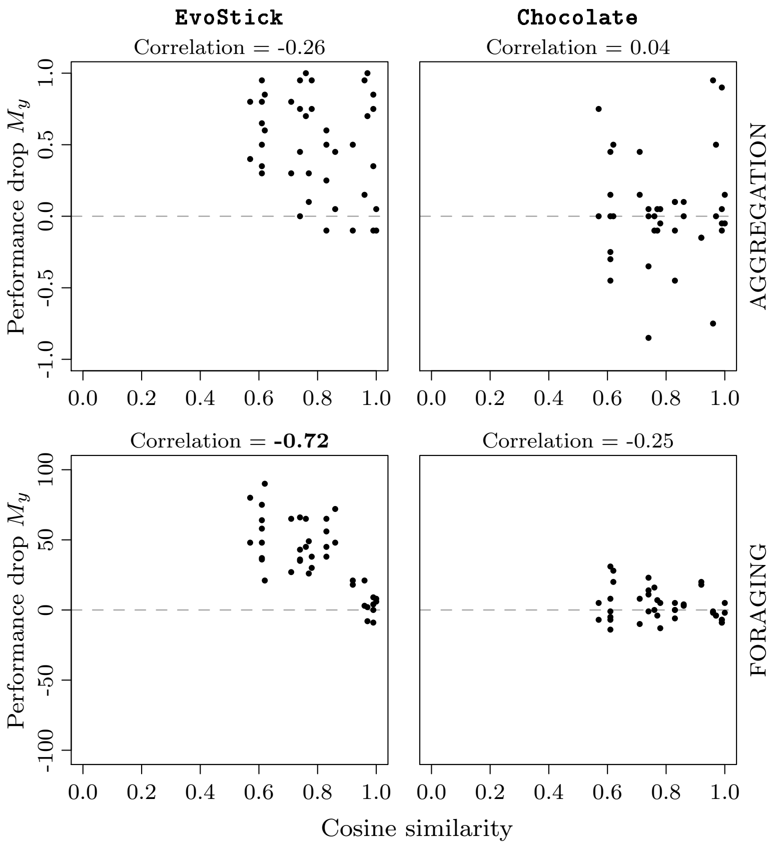

Fig. 9 Pearson correlation between performance drop and cosine similarity between models used for design and evaluation. See caption of Figure 7 for more details.

than as the result of evaluating in a complex (pseudo-)reality control software that has been designed on the basis of a simplistic simulation model.

Norms of differences, which are commutative and effectively fold negative widths onto positive ones, highlight a correlation between width of the reality gap and performance drop—see Figure 8 (*bottom*). Pearson correlation is significant both for `EvoStick` and `Chocolate`, but is larger for `EvoStick`. Also the cosine similarity highlights a correlation between width of the reality gap and performance drop—more precisely, a negative correlation between similarity of the design and evaluation models and performance drop—see Figure 9 (*bottom*). In this case, the Pearson correlation is significant only for `EvoStick`.

All in all, norms of differences and the cosine similarity appear to be the most appropriate choices among those we explored to measure the width of a (pseudo-)reality gap.

6 Conclusions

In the paper, we shed light on one of the most challenging issues in off-line automatic design of control software for robot swarms: the reality gap. Because of the reality gap, evaluation on physical robots is the only conclusive way of assessing the performance of an instance of control software. Nonetheless, tests with real hardware are time consuming, expensive, and possibly dangerous. Therefore, it would be highly desirable to have a simulation-only procedure that could provide reliable estimations of real-world performance. Such a procedure would greatly benefit off-line fully-automatic, semi-automatic (human-in-the-loop), and manual design. In particular, in the case of fully-automatic design, such evaluation procedure could contribute to handle the so-called *overdesign*: it has been shown that, past an optimal number of steps of the design process, the performance observed in reality diverges from the one in simulation (Birattari et al. 2016). As a consequence, protracting a design process indefinitely could be counterproductive. A simulation-only evaluation procedure could be used to implement an *early stopping* mechanism that halts the design process when overdesign have occurred (Morgan and Bourlard 1990; Caruana et al. 2001). In the case of semi-automatic design (human-in-the-loop), designers could use the fast and relatively inexpensive estimations of a simulation-only procedure to inspect the control software produced and to gain insight into its strengths and weaknesses. The information gathered could be used to steer the design process towards better control software. For example, designers could fine-tune hyperparameters of the optimization algorithm, add terms to the objective function to reward or penalize elements of the swarm behavior, or adjust simulation models to increase the robustness to the reality gap. A simulation-only procedure could benefit also a manual design process as it has been shown that, under some conditions, also human designers are prone to produce control software that is negatively affected by the reality gap (Francesca et al. 2015).

The results presented in this paper suggest that a simulation-only evaluation procedure can possibly be developed leveraging the notion of pseudo-reality as intended in the experiment of Section 5: (i) a range of models could be defined around the one used for the design, and (ii) each instance of control software generated by the design method under analysis could be evaluated on several models that are uniformly sampled from the defined range. The performance across such several models would provide information on the intrinsic robustness of the control software produced by the method at hand, and could be beneficial to the design in the various ways highlighted above.

Although in the paper we considered only models that differ by sensor and actuator noise, other ways could be conceived to generate a range of models to be used as a pseudo-reality. For example, one could change the distribution of noise, add an offset to noise or other parameters of the model, or completely modify the structure of the model. Further research is needed to assess these ideas and possibly devise more appropriate ones.

Acknowledgements.

The project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 681872). Mauro Birattari acknowledges support from the Belgian *Fonds de la Recherche Scientifique* – FNRS.

References

Andrychowicz M, Baker B, Chociej M, Jozefowicz R, McGrew B, Pachocki J, Petron A, Plappert M, Powell G, Ray A, Schneider J, Sidor S, Tobin J, Welinder P, Weng L, Zaremba W (2018) Learning dexterous in-hand manipulation. In: eprint arXiv:1808.00177

Beni G (2004) From swarm intelligence to swarm robotics. In: Şahin E, Spears WM (eds) Swarm Robotics, SAB, Springer, Berlin Heidelberg, Germany, LNCS, vol 3342, pp 1–9

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

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

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

Birattari M, Delhaisse B, Francesca G, Kerdoncuff Y (2016) Observing the effects of overdesign in the automatic design of control software for robot swarms. In: Dorigo M, et al (eds) Swarm Intelligence, 10th International Conference, ANTS, Springer, Cham, Switzerland, LNCS, vol 9882, pp 45–57

Boeing A, Braunl T (2012) Leveraging multiple simulators for crossing the reality gap. In: International Conference on Control, Automation, Robotics and Vision, ICARCV, IEEE Press, Piscataway NJ, pp 1113–1119

Bongard J, Lipson H (2004) Once more unto the breach: co-evolving a robot and its simulator. In: Pollack J, et al (eds) Artificial Life IX: Proceedings of the Conference on the Simulation and Synthesis of Living Systems, pp 57–62

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 (2015) 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 JM, Liu W, Winfield AF (2012) Environment-driven distributed evolutionary adaptation in a population of autonomous robotic agents. Mathematical and Computer Modelling of Dynamical Systems 18(1):101–129

Brooks R (1992) Artificial life and real robots. In: Varela FJ, Bourgine P (eds) Towards a Practice of Autonomous Systems. Proceedings of the First European Conference on Artificial Life, MIT Press, Cambridge MA, pp 3–10

Caruana R, Lawrence S, Lee Giles C (2001) Overfitting in neural nets: backpropagation, conjugate gradient, and early stopping. In: Leen T, Dietterich T, Tresp V (eds) Advances in Neural Information Processing Systems 13, NIPS 2000, MIT Press, pp 402–408

Dorigo M, Birattari M (2007) Swarm intelligence. Scholarpedia 2(9):1462

Floreano D, Mondada F (1996) Evolution of plastic neurocontrollers for situated agents. In: Maes P, et al (eds) From animals to animats 4: Proceedings of the International Conference on Simulation of Adaptive Behavior, ETH Zurich, Switzerland

Floreano D, Urzelai J (2001) Evolution of plastic control networks. Autonomous Robots 11(3):311–317

Floreano D, Husbands P, Nolfi S (2008) Evolutionary robotics. Handbook of Robotics pp 1423–1451

Francesca G, Birattari M (2016) Automatic design of robot swarms: achievements and challenges. Frontiers in Robotics and AI 3(29):1–9

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

Francesca G, Brambilla M, Brutschy A, Garattoni L, Miletitch R, Podevijn G, Reina A, Soleymani T, Salvaro M, Pinciroli C, Birattari M (2015) AutoMoDe-Chocolate: automatic design of control software for robot swarms. Swarm Intelligence 9(2/3):125–152

Garattoni L, Francesca G, Brutschy A, Pinciroli C, Birattari M (2015) Software infrastructure for e-puck (and TAM). Tech. Rep. TR/IRIDIA/2015-004, IRIDIA, Université libre de Bruxelles, Belgium

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

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

Haasdijk E, Bredeche N, Eiben A (2014) Combining environment-driven adaptation and task-driven optimisation in evolutionary robotics. PloS ONE 9(6):e98466

Hamann H (2018) Swarm robotics: a formal approach. Springer, Berlin, Germany

Hamann H, Wörn H (2008) A framework of space–time continuous models for algorithm design in swarm robotics. Swarm Intelligence 2(2–4):209–239

Hasselmann K, Ligot A, Francesca G, Birattari M (2018a) Reference models for AutoMoDe. Tech. Rep. TR/IRIDIA/2018-002, IRIDIA, Université libre de Bruxelles, Belgium

Hasselmann K, Robert F, Birattari M (2018b) Automatic design of communication-based behaviors for robot swarms. In: Dorigo M, et al (eds) Swarm Intelligence, ANTS, LNCS, vol 11172, Springer, Cham, Switzerland, pp 16–29

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

Jakobi N (1998) Minimal simulations for evolutionary robotics. PhD thesis, University of Sussex, Falmer, UK

Jakobi N, Husbands P, Harvey I (1995) Noise and the reality gap: the use of simulation in evolutionary robotics. In: Morán F, et al (eds) Advances in Artificial Life, Springer, London, UK, LNCS, vol 929, pp 704–720

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 JB, Doncieux S (2013) The transferability approach: crossing the reality gap in evolutionary robotics. IEEE Transactions on Evolutionary Computation 17(1):122–145

Kuckling J, Ligot A, Bozhinoski D, Birattari M (2018) Behavior trees as a control architecture in the automatic modular design of robot swarms. In: Dorigo M, et al (eds) Swarm Intelligence, ANTS, LNCS, vol 11172, Springer, Cham, Switzerland, pp 30–43

Lee JB, Arkin RC (2003) Adaptive multi-robot behavior via learning momentum. In: George Lee CS (ed) IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, IEEE Press, Piscataway NJ, pp 2029–2036

Ligot A, Birattari M (2019) Simulation-only experiments to mimic the effects of the reality gap in the automatic design of robot swarms. Supplementary material, `http://iridia.ulb.ac.be/supp/IridiaSupp2019-002`

López-Ibáñez M, Dubois-Lacoste J, Pérez Cáceres L, Birattari M, Stützle T (2016) The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives 3:43–58

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

Mondada F, Franzi E, Ienne P (1994) Mobile robot miniaturisation: A tool for investigation in control algorithms. In: Yoshikawa T, Miyazaki F (eds) Experimental Robotics III, Springer, Berlin, Heidelberg, pp 501–513

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

Morgan N, Bourlard H (1990) Generalization and parameter estimation in feedforward nets: some experiments. In: Touretzky DS (ed) Advances in Neural Information Processing Systems 2, NIPS 1990, Morgan Kaufmann, San Francisco, pp 630–637

Nolfi S, Floreano D, Miglino G, Mondada F (1994) How to evolve autonomous robots: different approaches in evolutionary robotics. In: Brooks RA, Maes P (eds) Artificial Life IV: Proceedings of the Workshop on the Synthesis and Simulation of Living Systems, MIT Press, Cambridge MA, pp 190–197

Peng XB, Andrychowicz M, Zaremba W, Abbeel P (2018) Sim-to-real transfer of robotic control with dynamics randomization. In: 2018 IEEE International Conference on Robotics and Automation (ICRA), pp 1–8

Pinciroli C, Trianni V, O'Grady R, Pini G, Brutschy A, Brambilla M, Mathews N, Ferrante E, Di Caro G, Ducatelle F, Birattari M, Gambardella L, Dorigo M (2012) ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. Swarm Intelligence 6(4):271–295

Reina A, Valentini G, Fernández-Oto C, Dorigo M, Trianni V (2015) A design pattern for decentralised decision making. PloS ONE 10(10):e0140950

Şahin E (2004) Swarm robotics: From sources of inspiration to domains of application. In: Şahin E, Spears WM (eds) Swarm Robotics, SAB, Springer, Berlin Heidelberg, Germany, LNCS, vol 3342, pp 10–20

Silva F, Urbano P, Correia L, Christensen AL (2015) odNEAT: An algorithm for decentralised online evolution of robotic controllers. Evolutionary Computation 23(3):421–449

Silva F, Duarte M, Correia L, Oliveira S, Christensen A (2016) Open issues in evolutionary robotics. Evolutionary Computation 24(2):205–236

Urzelai J, Floreano D (2000) Evolutionary robotics: coping with environmental change. In: Whitney LD, et al (eds) Proceedings of Conference on the Genetic and Evolutionary Computation Conference, GECCO, Morgan Kaufmann, San Francisco CA, pp 941–948

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

Zagal JC, Ruiz-Del-Solar J (2007) Combining simulation and reality in evolutionary robotics. Journal of Intelligent & Robotic Systems 50(1):19–39

Zagal JC, Ruiz-Del-Solar J, Vallejos P (2004) Back to reality: Crossing the reality gap in evolutionary robotics. In: IFAC/EURON Symposium on Intelligent Autonomous Vehicles, IAV, Elsevier, vol 37, pp 834 – 839