Toward an Empirical Practice in Offline Fully Automatic Design of Robot Swarms

Antoine Ligot[1], Andres Cotorruelo[1], Emanuele Garone[2], and Mauro Birattari[1],*

[1] IRIDIA, Université libre de Bruxelles (ULB), Brussels, Belgium.
[2] Service d'Automatique et d'Analyse des Systèmes (SAAS), Université libre de Bruxelles (ULB), Brussels, Belgium.

Abstract

Due to the lack of systematic empirical analyses and comparisons of ideas and methods, a clearly established state of the art is still missing in the optimization-based design of robot swarms. In this paper, we propose an experimental protocol for the comparison of fully-automatic design methods. This protocol is characterized by two notable elements: a way to define benchmarks for the evaluation and comparison of design methods, and a sampling strategy that minimizes the variance when estimating their expected performance. To define generally applicable benchmarks, we introduce the notion of *mission generator*: a tool to generate missions that mimic those a design method will eventually have to solve. To minimize the variance of the performance estimation, we show that, under some common assumptions, one should adopt the sampling strategy that maximizes the number of missions considered—a formal proof is provided as supplementary material. We illustrate the experimental protocol by comparing the performance of two off-line fully-automatic design methods that were presented in previous publications.

1 Introduction

A robot swarm is a decentralized system in which individual robots with local sensing and communication capabilities cooperate to accomplish a common mission [16, 11]. As a robot swarm cannot be programmed as a whole, obtaining a desired collective behavior can only be achieved by designing the appropriate behavior of the individual robots, which is made complex by the numerous robot-robot and robot-environment interactions that characterize the functioning of a

*The experiments were conceived by AL and MB and performed by AL. The theorem was proposed by MB; its proof was redacted by AC and revised by the four authors. MG 1 was implemented by AL. The paper was drafted by AL and revised by the four authors. The research was directed by MB.

swarm. Although methodologies exist to guide a designer in the development of control software for specific missions [26, 32, 2, 1, 10, 42, 37, 25], no generally applicable one has been proposed so far. Therefore, in most cases, designers proceed by trial and error and the design process is costly, time consuming, and not repeatable [10, 9].

Optimization-based design is a promising alternative. In this approach, an optimization algorithm searches for the instance of control software of the individual robots that maximizes a mission-dependent measure of the collective performance of the swarm—or, at least, for an instance that scores sufficiently well with respect to the given performance measure. The archetypal example of optimization-based design method is the application of evolutionary robotics to swarm robotics: robots are controlled by neural networks whose parameters and/or topologies are optimized by an evolutionary algorithm [46, 15].

Optimization-based design methods are typically categorized as either on-line [12] or off-line [5]. In on-line design, the design process takes place while the robots operate in the environment. In off-line design, it happens prior to the deployment of the robots, typically based on computer simulations. Due to the risk of damaging robots and environment with sub-optimal control software during the early phases of the design process, the relative small search space they can explore, and the limitation of their applications to scenarios in which the swarm can compute its collective performance, on-line methods appear to be more appropriate to adjust the parameters of an existing solution rather than to create one from the ground up [18]. Designing control software off-line on the basis of simulations does not suffer from the drawbacks illustrated above, and therefore appears to be a more effective and viable approach than on-line design. Unfortunately, off-line methods are faced with a major problem that is yet to be solved: the reality gap [13, 31, 44]. The reality gap is the intrinsic difference between simulation and reality. Due to the reality gap, control software developed using off-line methods often performs poorly in reality, despite giving good results in simulation [30]. As a consequence, executing automatically generated control software on physical robots is a mandatory step in the assessment of an off-line design method.

In addition to the on-line/off-line categorization of optimization-based design methods, another categorization that distinguishes between semi-automatic and fully-automatic design methods has recently been discussed [6]. In semi-automatic design, the design method is used as a tool by a human expert who is allowed to adapt the method to the mission at hand so that the produced solution satisfies the requirements. In fully-automatic design, the design method does not undergo any per-mission manual modifications. Due to the difference in operational functioning between semi- and fully-automatic design, they address different contexts of application. Semi-automatic is best suited for complex, one-of-a-kind missions for the solution of which it is reasonable to expect that sufficient time and resources can be allotted to allow human designers to iteratively adjust the functioning of a design method based on evaluations of control software produced by this method. The relative high cost of this human-in-the-loop process is justified by the exceptional nature of the missions to be solved.

Fully-automatic is best suited when one must solve multiple missions repeatedly, one after another, in such a way that the presence of a human expert in the design loop would be unfeasible due to monetary and time constraints. In this fully-automatic context, a design method is therefore expected to be able to design control software for any mission belonging to a given *class of mission* without any intervention of a human designer on a per-mission basis [6].

Many studies have been conducted in optimization-based design of robot swarms, and a number of promising methods have been proposed—see for Birattari *et al.* [6] for a recent overview and further references. Yet, a solid and appropriate empirical practice is still missing [5]. In fact, with very few exceptions, most studies introduce novel ideas or methods without comparing them with previously published ones. The lack of a well-established and consistently applied empirical benchmarking practice hinders the progress of the optimization-based design of robot swarms. It translates into the absence of a clear state of the art, which eventually hinders the practical exploitation of the best ideas proposed so far and their further development [8].

In this paper, we propose an experimental protocol for the evaluation and comparison of fully-automatic methods[1]. More precisely, we focus on two elements that are crucial in the definition of such protocol: publicly shared benchmarks and a sampling strategy that minimizes the variance of the performance estimation. Benchmarks are decisive tools in the identification of strengths and weaknesses of a method, and they promote the consistent application of meaningful, coherent, and well-defined evaluation criteria. Conceptually, a benchmark for the evaluation of fully-automatic design methods is a (possibly infinite) class of missions: a set of missions associated with a probability measure that determines their relative frequency of appearance. A class of missions might comprise both missions that are of different types (i.e., that differ by the nature of their goals), and missions of the same type that differ by minor variations. These minor variations can be at the level of the environment in which the swarm operates (e.g., different density of robots, presence of a reference point, number of points of interest), or at the level of the swarm itself (e.g., number of robots, initial configuration of the swarm). For example, two missions are of different type if the goal of one is for the robots to aggregate on a point of interest, and the one of the other is to gather objects initially scattered in the environment. For example, two missions of the same type differ in minor variations if, *ceteris paribus*, a task is to be accomplished in an environment in which a reference point such as a light source is present, and in an environment without a reference point. Although the difference between the two given missions might be qualified as minor, it can be sufficiently important that the two missions benefit from a tailored design. In the example, a design method is likely to exploit the reference point to produce control software that enables the robots to orient themselves in the environment and find the points of interest faster

[1]It should be noted that, although we present the experimental protocol in the context of swarm robotics—which is our specific domain of expertise—the reasoning also applies to other contexts where automatic design is performed (e.g., single robot systems, automatic decision algorithms).

than in the case where no reference point is present. An operational definition of a class of missions can be given in the form of a *mission generator*: a computer program that generates missions belonging to the class at hand. In other terms, running a mission generator is effectively a way to sample the corresponding class of missions—that is, selecting one of the missions of the set, according to the associated probability measure, and independently of the design method to which it applies. In this paper, we illustrate the concept of mission generator by presenting one we named MG 1, short for *mission generator 1*. We use MG 1 in an illustrative study in which we compare two previously proposed design methods.

Concerning the second notable element of the protocol we propose, that is, the sampling strategy, it should be noted that the performance of a fully-automatic design method is a stochastic variable that is affected by three sources of randomness: the mission to be solved, the realization of the design process, and the execution on the robots of the instance of control software produced by the design process itself. Taking for granted that multiple runs are needed to reduce the variance of the estimation of the expected performance, the questions that arise are: *How many missions should one consider? How many design processes should one run on each mission? How many times should one execute each of the instances of control software produced?* The obvious answer would be: the more the better! Indeed, by increasing indefinitely the number of missions, number of design processes performed on each mission, and the number of evaluations of each instance of control software generated, the variance of the estimation would converge to zero. However, in practice one has typically (if not always) to face practical constraints that limit the number of experiments that can be performed. Indeed, running experiments with robots is time consuming and could demand a large amount of resources. In this paper, we argue that, in order to estimate the expected performance of a fully-automatic design method under the assumption that a limited number of executions of the control software on the physical robots can be performed (and under a few other technical assumptions to be detailed in the following), the sampling strategy to be adopted to minimize the variance of the estimate is the one that maximizes the number of different missions considered. The sampling strategy therefore implies that one design process should be run on each mission considered and that the resulting instance of control software is executed once on the physical robots. An intuitive explanation of this claim is given in the body of the paper, and a formal proof is provided as supplementary material.

It is our contention that the protocol we propose here is crucial for the development of the optimization-based design of robot swarms into a mature scientific domain: it will contribute to make clear and objective comparisons between different methods that will allow to establish an objective state of the art. Eventually, this will promote the best ideas proposed so far and the elaboration of new ones. This protocol is to be adopted when one evaluates design methods in the fully-automatic context and thus wants to estimate their performance over a whole class of missions rather than over specific ones [5, 6]. The two components of the protocol are fundamentally complementary: the

sampling strategy recommends to evaluate a design method on the maximal number of missions possible, and the notion of mission generator allows one to sample as many missions as desired.

The remainder of the paper is organized as follows: in Section 2, we describe the implementation of the mission generator MG 1; in Section 3, we elaborate on the sampling strategy that minimizes the variance of the estimated performance of a fully-automatic design method; in Section 4, we present an illustrative experiment; and in Section 5, we share our concluding thoughts. In the illustrative experiment presented in Section 4, we demonstrate the concepts introduced by evaluating and comparing two previously proposed fully-automatic design methods on 30 missions. We evaluate the two methods using the sampling strategy described in Section 3 on the 30 missions generated by the mission generator MG 1 described in Section 2.

2 A mission generator

In this section, we present MG 1, a generator of missions for robot swarms. A mission generator samples missions from a class of missions according to the associated probability measure. The class of missions is defined on the basis of the capabilities of a robotic platform—which are described by a *reference model* that formally characterizes the sensors and actuators of the platform [20]. Indeed, it would not be reasonable to consider a class of missions containing ground missions if aquatic robots are expected to perform samples of this class. The probability measure can be tuned so as to mimic a realistic frequency of appearance of deployment conditions. For example, based on previous rescue missions at sea, it could be determined that 80% were performed when the sea swell was high, 15% when it was moderate, and 5% when it was low. These probabilities can be used to define a generator so that sampled missions reflect the conditions that the robot swarm will face when deployed, and therefore enable a realistic estimation of the expected performance of a design method.

MG 1 samples missions defined on the basis of the capabilities of an enhanced version of the e-puck robot [40]. This version of the e-puck robot is capable of perceiving a light source, the presence of nearby obstacles, and the gray-scale color of the ground directly below its body. It can also detect the presence and estimate the relative position of neighboring peers. These capabilities are formally described by the reference model RM 1.1 [29] reproduced in Table 1 for the reader's convenience. The missions take place in an enclosed area surrounded by walls—what we call an *arena*. The ground of the arena is gray, with some areas being black or white. Obstacles can be present within the arena, and a unique source of light is positioned right outside of the arena's walls. The light source is either on or off for the whole duration of the mission. Figure 1 illustrates the possible arenas that can be generated by MG 1 and provides details about the possible positions and sizes of the colored areas (i.e., black or white) and the obstacles.

MG 1 can instantiate missions of three types: FORAGING, HOMING, and

Table 1: Reference model RM 1.1 [29] of the version of the e-puck robot [40] enhanced with additional hardware modules [23]: the Overo Gumstix, the ground sensor, and the range-and-bearing module [24]. The original version of the e-puck can detect obstacles and measure the intensity of the ambient light. With the additional modules, the e-puck can detect the gray-level color of the floor situated under its body, and detect the number n and estimate the relative position of neighboring peers situated in an approximate range of 0.70 m. The range-and-bearing vector $V_d = \sum_{m=1}^{n}(\frac{1}{1+r_m}, \angle b_m)$, where r_m and $\angle b_m$ are range and bearing of neighbor m, respectively, points to the aggregate position of the neighboring peers. If $m = 0$, then $V_d = (1, \angle 0)$. The variables are updated every control cycle—that is, every 100 ms.

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	$v_l, v_r \in [-0.12, 0.12]\text{ms}^{-1}$

AGGREGATION-XOR. To ensure the soundness of the instances created, we implemented within MG 1 several mission-specific conditions that guide the configuration of the arenas. Independently of the type of mission to be accomplished, MG 1 selects the number of robots in the swarm, their initial distribution, and the duration of the mission. We considered three families of initial random distribution for the robots: (i) uniform, the robots are deployed anywhere in the arena; (ii) one-side, the robots are deployed on one half of the arena, either close or far from the light and independently of whether the latter is on or not; and (iii) not-on-colored-areas, the robots are deployed anywhere in the arena, but not on the black or white areas. Table 2 summarizes the main parameters of MG 1; the types of missions and their respective conditions are described in the following subsections.

We created MG 1 as an open-source library[2] for the ARGoS simulator [41]. It should be noted that the library we share does not only implement MG 1, but a whole family of mission generators. Indeed, by modifying the parameters of MG 1, such as the frequency of appearance of the missions, one can create a different mission generator that would sample a different class of missions.

2.1 FORAGING

The robots must retrieve virtual items from food sources to nest areas. A robot is deemed to pick up an item when it enters an area representing a food source,

[2]Available as a GitHub repository: `https://github.com/demiurge-project/MissionGeneratorMG1`

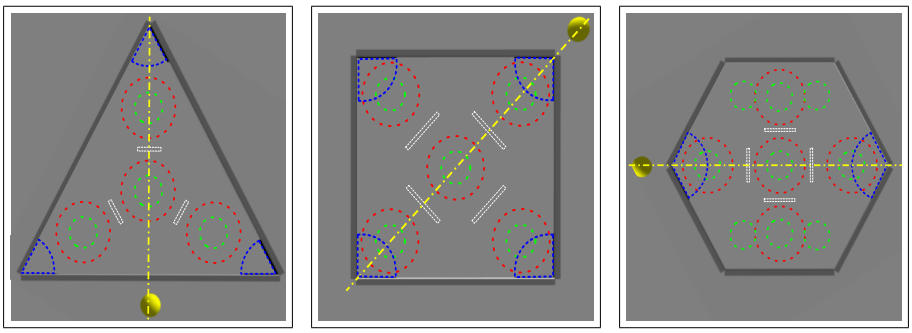

Figure 1: Schemes of the possible arenas that can be generated by MG 1, with placeholders for the possible environmental elements. The triangle arena covers an area of 2.5 m^2, the square one an area of 3.76 m^2, and the hexagonal one an area of 4.3 m^2. The dotted circles and portions of circles represent placeholders for possible black or white areas. The red circles are placeholders for circular areas of 0.3 m radius, the green ones for circular areas of 0.15 m radius, and the blue ones are placeholders for portions of circular areas of 0.40 m radius. The white dashed rectangles represent placeholders for possible obstacles whose height and width are fixed to 0.07 m and 0.026 m, respectively. The length of these obstacles depend on the shape of the arena: 0.25 m, 0.45 m, and 0.35 m for the triangle, square, and hexagonal one, respectively. The yellow sphere represents the light source. For missions of type FORAGING, the possible nest areas are restricted to the circles positioned on the axis of the light source pictured by the yellow dashed-and-dotted line.

and drop the item when it then enters an area representing a nest. There can be up to two nests and up to three food sources. The nests areas can only be placed on the axis perpendicular to the light source depicted as the yellow dotted-and-dashed line in Figure 1. The food sources can be placed all around the arena. The size and positions of the areas are selected randomly with equal probability. MG 1 ensures that the areas representing the nests and the ones representing the food sources are of different color (i.e., if the nest is white, the food sources are black, and vice versa). The objective function to be maximized is $F_{\text{FORAGING}} = I$, where I is the number of items dropped in the nest areas after the duration of the mission.

2.2 HOMING

The robots must aggregate on a black or white area designated as their home. MG 1 places between one and three areas in the arena: one for the home, and possibly two others to serve as distractions. MG 1 ensures that the area designated as the home is large enough so as to accommodate all the robots, and that the color of the distraction areas differs from that of the home area. There

Table 2: The parameters, their possible values, and the corresponding probability distributions in MG 1. The swarm can comprise 15 or 20 robots. There is a total of 5 different objective functions that can be selected by MG 1: one for FORAGING, and two for both HOMING and AGGREGATION-XOR. For the 5 objective functions to appear with equal probability, MG 1 selects HOMING and AGGREGATION-XOR with double the probability of FORAGING. See Sections 2.1 to 2.3 for descriptions of the types of missions.

Parameter	Possible values	Probability distribution
mission type	{FORAGING, HOMING, AGGREGATION-XOR}	$\{0.2, 0.4, 0.4\}$
duration	$\{60, 120, 180\}$	uniform
# robots	$\{15, 20\}$	uniform
shape arena	$\{triangle, square, hexagon\}$	uniform
initial distribution	$\{uniform, one\text{-}side, not\text{-}on\text{-}colored\text{-}areas\}$	uniform
Mission type: FORAGING		
light	$\{on, off\}$	$\{0.85, 0.15\}$
# nests	$\{1, 2\}$	$\{0.95, 0.05\}$
color nest	$\{black, white\}$	uniform
# food sources	$\{1, 2, 3\}$	$\{0.5, 0.4, 0.1\}$
# obstacles	$\{0, 1, 2, 3\}$	$\{0.2, 0.4, 0.3, 0.1\}$
Mission type: HOMING		
objective function	$\{anytime, endtime\}$	uniform
light	$\{on, off\}$	$\{0.30, 0.70\}$
# colored areas	$\{1, 2, 3\}$	$\{0.55, 0.30, 0.15\}$
color home	$\{black, white\}$	uniform
# obstacles	$\{0, 1, 2, 3\}$	$\{0.20, 0.40, 0.30, 0.10\}$
Mission type: AGGREGATION-XOR		
objective function	$\{anytime, endtime\}$	uniform
light	$\{on, off\}$	$\{0.30, 0.70\}$
# aggregation areas	$\{2, 3\}$	$\{0.80, 0.20\}$
color aggregation areas	$\{black, white\}$	uniform
# distraction areas	$\{0, 1\}$	$\{0.60, 0.40\}$
# obstacles	$\{0, 1, 2, 3\}$	$\{0.2, 0.4, 0.3, 0.1\}$

are two possible objective functions for this type of mission: *anytime* and *endtime*. The two objective functions are to be maximized and depend on N_{home}, the number of robots located in the aggregation area. With *anytime* the performance is measured by the objective function $F'_{\text{HOME}} = \sum_{t=1}^{T/100\,\text{ms}} N_{home}(t)$, where T is the duration of the mission (in seconds) and 100 ms is the period at which N_{home} is evaluated. With *endtime* the performance is measured once, at the end of the mission, and the objective function is $F''_{\text{HOME}} = N_{home}$.

2.3 AGGREGATION-XOR

The robots must select and aggregate on a single area among multiple ones present in the arena. MG 1 configures the arena to have two or three aggregation areas of the same color, large enough so that all robots can stand on

each of them. If two aggregation areas are placed, MG 1 can place yet another one of different color, small or large, that serves as a distraction. There are two possible objective functions for this type of mission: *anytime* and *endtime*. The two objective functions are to be maximized and depend on N, the total number of robots in the swarm; and N_i, the number of robots located in the aggregation area i, with $i \in \{a, b\}$ or $\{a, b, c\}$, depending on the number of aggregation areas. If MG 1 selects *anytime*, the objective function is $F'_{\text{XOR}} = \sum_{t=1}^{T/100\,\text{ms}} \max_i(N_i(t))/N$, where T is the duration of the mission (in seconds) and 100 ms is the period at which $\max_i(N_i)/N$ is evaluated. If MG 1 selects *endtime*, the objective function is $F''_{\text{XOR}} = \max_i(N_i)/N$, and it is computed at the end of the mission.

3 Sampling strategy for performance estimation

The performance of a fully-automatic design method is a stochastic variable and therefore estimating its expectation is a reasonable goal. The expectation should be computed with respect to all the sources of randomness involved in the process. The sources of randomness are the following:

The mission: the mission is randomly sampled from a class of missions according to the associated probability measure. If the class of missions is sampled multiple times, the missions to be solved by will (likely) differ one from the other.

The design process: the design process is stochastic in nature. If it is performed multiple times, it will (likely) produce different instances of control software.

The execution: the execution of an instance of control software on physical robots is a stochastic event—the resulting performance is therefore a stochastic quantity. If the same instance of control software is executed multiple times, the performance observed will (likely) vary.

Let us assume that an upper bound N on the number of executions is given. This assumption is realistic, as running experiments with robots is time consuming and could demand a large amount of resources. It is also realistic to assume that the number of executions is the real bottleneck in terms of demanded resources. Indeed, running experiments with robots is a labor-intensive activity and the expensive and time-consuming part in the research on the automatic design of control software for robot swarms. On the other hand, the design process is fully automatic and multiple instances can run in parallel on a high-performance computing cluster. We can assume that the cost (in abstract terms: time and resources) of running a design process is negligible compared to the one of running robot experiments. We can also assume that sampling a mission from a class of instances is inexpensive. Finally, we assume that, before running a design process on a given mission, we do not have any prior information on how well the control software we can generate automatically will perform, on

what will be the variance of the performance, and on what will be the variances related to the three sources of randomness.

A sampling strategy for estimating the expected performance of a design method on a class of missions, given that a maximum number N of executions can be performed, can be formally described by a triple $\langle n_m, n_d, n_x \rangle$, with $n_m \cdot n_d \cdot n_x \leq N$. The expected performance is estimated on the basis of n_m missions, n_d design processes per mission (to generate n_d instances of control software per mission), and n_x executions of each of them. It has to be noticed that any triple $\langle n_m, n_d, n_x \rangle$ yields an unbiased estimate of the expected performance. Yet, different triples might differ for what concerns the variance of the estimate they yield, and it is thus of interest to understand which triple minimizes such variance.

In statistics, similar problems were studied since the early 1940s [22] under the name of *nested sampling*,[3] Classical results are based on the assumption that the sampled variable can be written as a sum of three mutually independent random variables—e.g., see [28, 27]. Nested sampling has been used in numerous and very diverse fields—e.g., food quality control [39], agriculture [33], blood pH of female mice [45], premium of insurance contracts [14], estimation of income [17].

The theoretical foundation of this paper is summarized in Theorem 1, which shows that, given a maximum number of executions, the best strategy is to maximize the number of missions to be considered. Note that, unlike the results available in the literature [22, 28, 27], this theorem does not require that the sampled random variable is expressed as the sum of three mutually independent random variables.

Theorem 1. *Under the assumptions made above, given that a maximum number N of executions can be performed, the sampling strategy described by the triple $\mathcal{E} = \langle n_m, n_d, n_x \rangle$, with $n_m = N$, $n_d = 1$, and $n_x = 1$, is the one that minimizes the variance of the estimate.*

Proof. The variance of the estimator $\hat{\mu}$ associated with the sampling strategy $\mathcal{E}$ is:

$$\mathbb{E}\left[(\hat{\mu}_{\mathcal{E}} - \mu)^2\right] = \frac{\sigma^2_{AM}}{n_m} + \frac{\bar{\sigma}^2_{AD}}{n_m\, n_d} + \frac{\bar{\sigma}^2_{WM}}{n_m\, n_d\, n_x}, \tag{1}$$

where σ^2_{AM} is the *across-mission variance* and indicates how missions differ from one another, $\bar{\sigma}^2_{AD}$ is the *expected across-design variance* and indicates how designs differ from one another within a same mission (averaged across all possible missions), and $\bar{\sigma}^2_{WM}$ is the *expected within-mission variance* and indicates how scores differ from one another within a same mission (averaged across all possible missions). Formal definitions of these three variances and a formal proof of Equation 1 are given as supplementary material. Clearly, to minimize the variance of the estimator, the denominators need to be chosen so as to be as large as possible. It is straightforward to conclude that this happens when $n_m = N$, $n_d = n_x = 1$ under the constraint $n_m \cdot n_d \cdot n_x \leq N$. □

[3] Currently, the term *nested sampling* is in use in Bayesian statistics and refers to a completely different and unrelated technique.

The same conclusion (i.e., that the triple $\langle N, 1, 1\rangle$ is the one that minimizes the variance) is relevant also in the case one wishes to compare the expected performance of two design methods—the reasoning can be generalized to more than two design methods, as well. When two methods are considered, the reasoning presented above applies to the estimation of the expected value of the difference between the performance of the control software produced by the two methods under analysis.

It should be noticed that, in the setting described above, the naive approach that is often adopted and that consists in running multiple executions of the same instance of control software hides some catches that could lead to misleading results. In particular, it could lead to wrong conclusions when two (or more) design methods are compared. By taking $n_x \gg 1$, one runs the risk of undersampling the space of the missions and oversampling the space of the realizations of the design processes and/or the one of the executions. Let us consider the comparison of two design methods, A and B. Let us make the hypothesis that the expected performance of A over the given class of mission is better than that of B. Let us also make the hypothesis that A typically outperforms B on most of the missions of the class, whereas it is outperformed on a small subset of missions. If the sampling strategy adopted undersamples the space of the missions to allow multiple executions of the same instances of control software, there exists the risk that the missions on which B outperforms A are over-represented in the sample. If this happens, as $n_x \gg 1$, the risk exists that the observed difference of performance, which will be wrongly in favor of B, happens to be statistically significant. By undersampling the space of the missions and oversampling the one of the realizations of the design processes and/or of the executions, the confidence level imposed does not apply anymore to the overall estimation of the differences over the entire class of instances but rather to the subset of missions that have been sampled. The above reasoning could possibly appear clearer if we push things to the extreme. Let us sample a single mission ($n_m = 1$), run a single design process ($n_d = 1$), and use all the N evaluations available to test the single instance of control software obtained. The confidence level will refer to the performance difference on the specific mission that has been sampled—rather than to the whole class, as we intend. If we happen to sample one of the few missions on which B performs better than A, we will conclude that B is better than A and (if N is sufficiently large) that the difference will be statistically significant. Clearly, this does not extend to the whole class, and the results obtained will be wrong even if statistical significance was attained. A similar wrong conclusion could be reached also if $n_m = 1$, $n_d = N$, and $n_x = 1$. On the other hand, if $n_m = N$ (and consequently $n_d = n_x = 1$), the issue does not arise and the confidence level applies indeed to the significance of the difference across the whole class of missions, as it should.

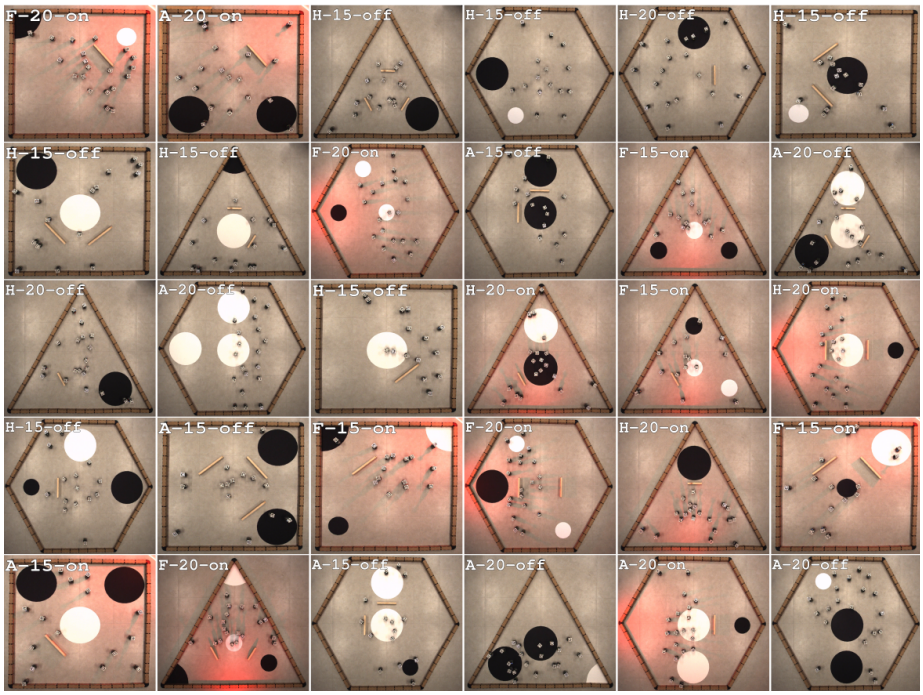

Figure 2: Pictures of the initial configurations of the missions generated by MG 1. Each image is labeled with the initial letter of the mission to be accomplished (that is, F for FORAGING, H for HOMING, and A for AGGREGATION-XOR), the number of robots in the swarm (that is, 15 or 20), and the status of the light (that is, on or off). Control software, mission descriptions, source codes, and videos of the experimental runs are available as supplementary material.

4 Illustrative Experiment

In this section, we assess and compare the performance of two previously proposed fully-automatic design methods following the sampling strategy described in Section 3: we consider 30 missions generated with MG 1, run each design method once on each mission, and execute each instance of control software produced once on the physical robots. We allocated a design budget of 100 000 simulation executions for each method on each mission: that is, each design process cannot exceed 100 000 simulation runs. To evaluate the intrinsic robustness of the methods, we also assess each instance of control software produced in simulation under the same initial conditions of the execution on the physical robots. Figure 2 shows pictures of the 30 arenas generated by MG 1.

We used `EvoStick` [21, 20] and `Chocolate` [19] to design control software for the 30 missions in a fully-automatic off-line way. `EvoStick` is a neuroevolutionary robotics method that produces control software in the form of fully-connected neural networks without hidden layers. The 25 input nodes of these

neural networks are fed with the readings of the sensors, as formally described by the reference model RM 1.1 described in Table 1. The 2 output nodes determine the velocity of the wheels. The 50 synaptic weights that connect the input nodes to the output nodes are real numbers in $[-5, 5]$, and are optimized by a genetic algorithm with a population size of 100 individuals and 10 evaluations per generation. The design budget of 100 000 simulation executions allocated to the genetic algorithm corresponds to 100 generations. `Chocolate` is a modular method that belongs to the AutoMoDe framework. It produces control software in the form of probabilistic finite state machines by selecting, fine-tuning, and combining pre-defined modules. The modules are programmed by hand *a priori* in a mission-agnostic way. They include 6 low-level behaviors to be used as states of the probabilistic finite state machines, and 6 conditions to be used as transitions between states. The probabilistic finite state machines produced can contain up to 4 states and 4 outgoing transitions per state, and are optimized by the optimization algorithm *irace* [7, 38]. It is important to notice that the modules of `Chocolate`, although programmed by hand, are defined once and for all and are not manually modified during the design process, which rightfully qualify `Chocolate` as a fully-automatic design method. We refer the reader to the original papers for further details on the methods [20, 19].

We chose `EvoStick` and `Chocolate` because the two methods have already been compared in several studies [19, 36, 35], and we wish to keep the focus of this paper on the experimental protocol used rather than on the outcome of a novel comparison of design methods. In these previous studies, results were always similar: `EvoStick` outperforms `Chocolate` in simulation, but `Chocolate` outperforms `EvoStick` is reality—in both cases, differences are significant with a confidence level of at least 95%. We expect to obtain similar results in our illustrative experiments. The novelty of the comparison we present here lies in the experimental protocol adopted. In the previously presented experiments, the experimental protocol considered 2 [36, 35] and 5 [19] missions selected by the experimenters, on which each method was executed 10 [36, 35] or 20 [19] times. These experimental protocols are not wrong, and the results obtained should not be disregarded. In fact, the adopted sampling strategy minimizes the variance of the expected performance for the specific missions considered [4, 3]. However, as discussed in Section 3, the results of these previous experiments strongly depend on the missions chosen, which, due to specificities that might be unknown to the experimenters, could favor a given design method over another. The protocol proposed in this paper aims at evaluating and comparing the performance of design methods over a whole class of missions rather than over specific ones, and should therefore be adopted when one evaluates design methods in the fully-automatic context [5, 6].

In simulation, `EvoStick` outperformed `Chocolate` in 18 out of the 30 considered missions, and the two methods obtained the same score in 3 missions. In reality, `Chocolate` outperformed `EvoStick` in 29 missions, and was outperformed only in one—see Fig. 3. Aggregating the performance observed on several different missions is not trivial because the range of performance for different missions might vary greatly. Therefore, naively averaging the performances

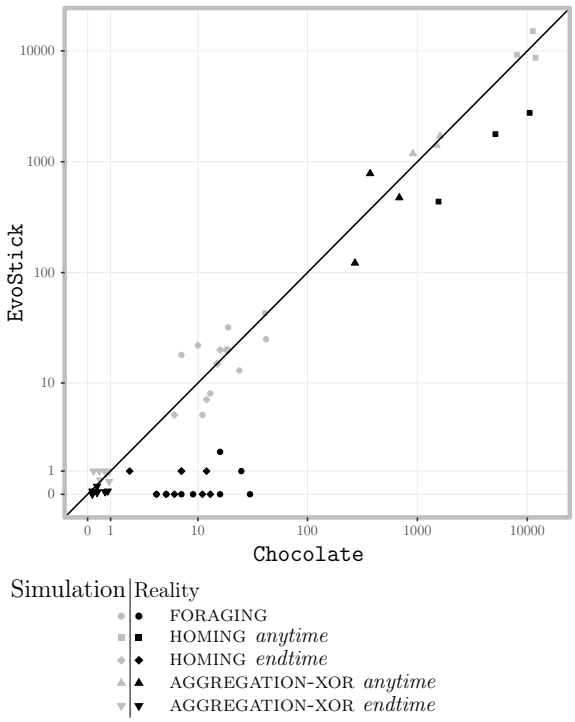

Figure 3: Scatter plot of the performance obtained by `Chocolate` and `EvoStick` for each mission in simulation (gray points) and in reality (black points). The performance are given in logarithmic scale. A point on the diagonal indicates that the two methods performed similarly on a given mission; a point below the diagonal indicates that `Chocolate` performed better than `EvoStick`, and inversely.

could give misleading results as missions for which the performance range is large would overshadow those for which it is small. Some form of normalization is needed to aggregate the performance observed on different missions. Here, for the purpose of this illustrative experiment, we address this issue by aggregating the performance observed on the different missions by reporting an estimation of the expected rank together with a 95% confidence interval—see Fig. 4. By computing the ranks on a per-mission basis, we put the results observed for the different missions on an equal footing, irrespectively of their possibly different ranges. We refer the reader to Francesca *et al.* [20] and Ligot & Birattari [34] for discussions on the conjecture behind the fact that the control software produced by `Chocolate` suffer less from the reality gap than the one produced by `EvoStick`.

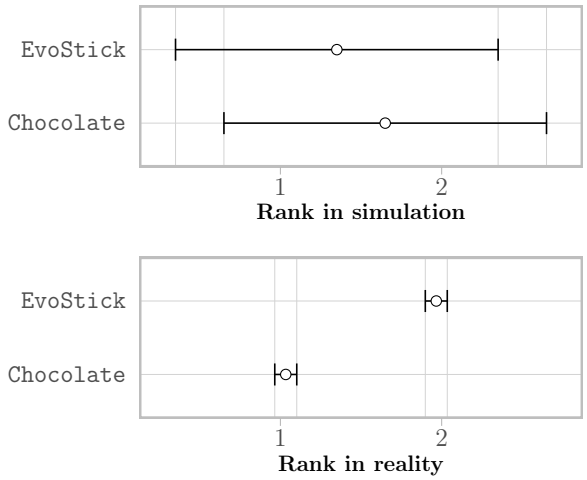

Figure 4: Aggregated performance obtained in simulation and in reality. The expected rank and 95% confidence interval of the expected rank are reported. The lower the expected rank, the better the performance. If two segments do not overlap, the expected ranks of the corresponding methods are significantly different with a confidence level of at least 95%.

5 Conclusion

We presented an experimental protocol for the fully-automatic design of control software for robot swarms. Elaborating adequate protocols for the assessment of semi-automatic methods is particularly challenging due to the role played by the human expert during the design process as well as the specific nature of the missions to be accomplished, and we leave this for future work. The protocol we proposed is characterized by two elements: the use of a random mission generator to produce test missions and a sampling strategy that minimizes the variance of the performance estimation. We have showed—both conceptually and formally—that the sampling strategy that minimizes the variance in the estimation of the expected performance is the one that maximizes the number of missions considered. The notion of mission generator goes therefore hand in hand with this sampling strategy as it allows to sample as many missions as desired.

To the best of our knowledge, the mission generator MG 1 we created is the first generator of missions for swarm robotics. Because its purpose is to illustrate the concepts introduced in this paper, MG 1 is relatively limited in the nature of the missions it generates: it generates missions belonging to only three types of missions to be solved by specific robots with specific capabilities. However, MG 1 can be extended and generalized in many different ways, and therefore can be the starting point for future generators. In fact, the core idea of MG 1 is to generate missions of different types that are characterized by specific

objective functions, and the definition of features of the environment together with relationships between these features. These elements can be easily reused to create mission generators dedicated to other robots, including robots of different nature (e.g., flying robots), under the condition that reasonable distributions can be devised for every variable of the missions. It should be also noted that in MG 1 we consider the number of robot as a parameter of the mission. As an alternative, the definition of the mission could impose a constraint on the maximum/minimum number of robots comprised in the swarm and the selection of the most appropriate number of robots could be left to the design process. The idea of defining the size of swarm automatically within the design process has been already explored by Salman *et al.* [43].

To illustrate the protocol, we conducted an experiment in which we evaluated and compared two previously proposed automatic design methods on 30 missions generated by MG 1. In this illustrative study, we allocated the same design budget to the two methods for each mission they had to solve. A thorough assessment of the capabilities of fully-automatic design methods would require the evaluation of these methods under different levels of the design budget to understand which methods performs best under which conditions. Further, in the illustrative study, we estimated the performance of the methods under analysis on each mission but we did not compute the aggregate performance across the missions. The main difficulty when estimating the performance of a design method on different missions is that the range of performance might vary greatly across the missions at hand. Some sort of normalization prior to the aggregation of the performance is mandatory. Here, we aggregated the performance observed by reporting the expected rank, which is an implicit form of normalization. The drawback of using ranks is that they do not provide an estimation of the overall performance of each of the design methods under analysis. As an alternative, one could normalize the performance on each mission based on the knowledge of the theoretical maximal and minimal performance, or based on a reasonable estimate of them (including, for example, the best and worse performance observed empirically [30]). However, in the illustrative experiment presented in this paper these alternatives did not appear to be appropriate as no prior knowledge was available and only two methods were involved in the study, providing therefore too little data to perform a meaningful normalization.

Although a number of issues could be addressed to define more advanced protocols, the ideas proposed in the paper are a significant step towards addressing the current lack of objective comparisons in the optimization-based design of robot swarms and, consequently, the lack of a clearly identified state of the art.

Acknowledgment

The project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (DEMIURGE Project, grant agreement No 681872) and from Belgium's Wallonia-

Brussels Federation through the ARC Advanced Project GbO—Guaranteed by Optimization. M. Birattari acknowledges support from the Belgian *Fonds de la Recherche Scientifique*—FNRS, of which he is a Research Director.

References

[1] Jacob Beal, Stefan Dulman, Kyle Usbeck, Mirko Viroli, and Nikolaus Correll. "Organizing the aggregate: languages for spatial computing". In: *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments*. Ed. by Mernik Marjan. Hershey, PA, USA: IGI Global, 2012, pp. 436–501.

[2] Spring Berman, Vijay Kumar, and Radhika Nagpal. "Design of control policies for spatially inhomogeneous robot swarms with application to commercial pollination". In: *IEEE International Conference on Robotics and Automation, ICRA*. Piscataway, NJ, USA: IEEE, 2011, pp. 378–385.

[3] Mauro Birattari. *Notes on the estimation of the expected performance of automatic methods for the design of control software for robot swarms*. Tech. rep. TR/IRIDIA/2020-10. Belgium: IRIDIA, Université libre de Bruxelles, 2020.

[4] Mauro Birattari. *On the estimation of the expected performance of a metaheuristic on a class of instances. How many instances, how many runs?* Tech. rep. TR/IRIDIA/2004-01. Belgium: IRIDIA, Université libre de Bruxelles, 2004.

[5] Mauro Birattari, Antoine Ligot, Darko Bozhinoski, Manuele Brambilla, Gianpiero Francesca, Lorenzo Garattoni, David Garzón Ramos, Ken Hasselmann, Miquel Kegeleirs, Jonas Kuckling, Federico Pagnozzi, Andrea Roli, Muhammad Salman, and Thomas Stützle. "Automatic off-line design of robot swarms: a manifesto". In: *Frontiers in Robotics and AI* 6 (2019), p. 59.

[6] Mauro Birattari, Antoine Ligot, and Ken Hasselmann. "Disentangling automatic and semi-automatic approaches to the optimization-based design of control software for robot swarms". In: *Nature Machine Intelligence* 2.9 (2020), pp. 494–499.

[7] Mauro Birattari, Zhi Yuan, Prasanna Balaprakash, and Thomas Stützle. "F-Race and Iterated F-Race: an overview". In: *Experimental Methods for the Analysis of Optimization Algorithms*. Ed. by Thomas Bartz-Beielstein, Marco Chiarandini, Luis Paquete, and Mike Preuss. Berlin, Germany: Springer, 2010, pp. 311–336.

[8] Mauro Birattari, Mark Zlochin, and Marco Dorigo. "Towards a theory of practice in metaheuristics design: A machine learning perspective". In: *RAIRO—Theoretical Informatics and Applications* 40.2 (2006), pp. 353–369.

[9] Darko Bozhinoski and Mauro Birattari. "Designing control software for robot swarms: software engineering for the development of automatic design methods". In: *Proceedings of the 1st International Workshop on Robotics Software Engineering, RoSE*. New York, NY, USA: ACM, 2018, pp. 33–35.

[10] Manuele Brambilla, Arne Brutschy, Marco Dorigo, and Mauro Birattari. "Property-driven design for swarm robotics: a design method based on prescriptive modeling and model checking". In: *ACM Transactions on Autonomous Adaptive Systems* 9.4 (2014), 17:1–17:28.

[11] Manuele Brambilla, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo. "Swarm robotics: a review from the swarm engineering perspective". In: *Swarm Intelligence* 7.1 (2013), pp. 1–41.

[12] Nicolas Bredeche, Evert Haasdijk, and Abraham Prieto. "Embodied evolution in collective robotics: a review". In: *Frontiers in Robotics and AI* 5 (2018), p. 12.

[13] Rodney Allen Brooks. "Intelligence without representation". In: *Artificial Intelligence* 47 (1991), pp. 139–159.

[14] Hans Bühlmann. "Experience rating and credibility". In: *ASTIN Bulletin: The Journal of the IAA* 4.3 (1967), pp. 199–207.

[15] Stéphane Doncieux, Nicolas Bredeche, Jean-Baptiste Mouret, and A.E. Eiben. "Evolutionary robotics: what, why, and where to". In: *Frontiers in Robotics and AI* 2 (2015), p. 4.

[16] Marco Dorigo, Mauro Birattari, and Manuele Brambilla. "Swarm robotics". In: *Scholarpedia* 9.1 (2014), p. 1463.

[17] Robert E Fay III and Roger A Herriot. "Estimates of income for small places: an application of James-Stein procedures to census data". In: *Journal of the American Statistical Association* 74.366a (1979), pp. 269–277.

[18] Gianpiero Francesca and Mauro Birattari. "Automatic design of robot swarms: achievements and challenges". In: *Frontiers in Robotics and AI* 3.29 (2016), pp. 1–9.

[19] Gianpiero Francesca, Manuele Brambilla, Arne Brutschy, Lorenzo Garattoni, Roman Miletitch, Gaëtan Podevijn, Andreagiovanni Reina, Touraj Soleymani, Mattia Salvaro, Carlo Pinciroli, Franco Mascia, Vito Trianni, and Mauro Birattari. "AutoMoDe-Chocolate: automatic design of control software for robot swarms". In: *Swarm Intelligence* 9.2–3 (2015), pp. 125–152.

[20] Gianpiero Francesca, Manuele Brambilla, Arne Brutschy, Vito Trianni, and Mauro Birattari. "AutoMoDe: a novel approach to the automatic design of control software for robot swarms". In: *Swarm Intelligence* 8.2 (2014), pp. 89–112.

[21] Gianpiero Francesca, Manuele Brambilla, Vito Trianni, Marco Dorigo, and Mauro Birattari. "Analysing an evolved robotic behaviour using a biological model of collegial decision making". In: *From Animals to Animats 12. Proceedings of the twelveth International Conference on Simulation of Adaptive Behavior, SAB*. Ed. by Tom Ziemke, Christian Balkenius, and John Hallam. Vol. 7426. Lecture Notes in Computer Science. Berlin, Germany: Springer, 2012, pp. 381–390.

[22] Mohonlal Ganguli. "A note on nested sampling". In: *Sankhyā: The Indian Journal of Statistics* (1941), pp. 449–452.

[23] Lorenzo Garattoni, Gianpiero Francesca, Arne Brutschy, Carlo Pinciroli, and Mauro Birattari. *Software infrastructure for e-puck (and TAM)*. Tech. rep. TR/IRIDIA/2015-004. Belgium: IRIDIA, Université libre de Bruxelles, 2015.

[24] Álvaro Gutiérrez, Alexandre Campo, Marco Dorigo, Jesus Donate, Félix Monasterio-Huelin, and Luis Magdalena. "Open e-puck range & bearing miniaturized board for local communication in swarm robotics". In: *IEEE International Conference on Robotics and Automation, ICRA*. Ed. by Kinugawa Kosuge. Piscataway, NJ, USA: IEEE, 2009, pp. 3111–3116.

[25] Heiko Hamann. *Swarm robotics: a formal approach*. Cham, Switzerland: Springer, 2018.

[26] Heiko Hamann and Heinz Wörn. "A framework of space–time continuous models for algorithm design in swarm robotics". In: *Swarm Intelligence* 2.2–4 (2008), pp. 209–239.

[27] Sahai Hardeo and Mario Miguel Ojeda. "Three-Way Nested Classification". In: *Analysis of Variance for Random Models: Unbalanced Data*. Boston, MA, USA: Birkhäuser, 2005, pp. 329–369.

[28] Sahai Hardeo and Mario Miguel Ojeda. "Two-Way Nested Classification". In: *Analysis of Variance for Random Models*. Vol. 2. Boston, MA, USA: Birkhäuser, 2005, pp. 287–328.

[29] Ken Hasselmann, Antoine Ligot, Gianpiero Francesca, David Garzón Ramos, Muhammad Salman, Jonas Kuckling, Fernando J. Mendiburu, and Mauro Birattari. *Reference models for AutoMoDe*. Tech. rep. TR/IRIDIA/2018-002. Belgium: IRIDIA, Université libre de Bruxelles, 2018.

[30] Ken Hasselmann, Antoine Ligot, Ruddick Julian, and Mauro Birattari. "Empirical assessment and comparison of neuro-evolutionary methods for the automatic off-line design of robot swarms". In: *Nature Communications* 12 (2021), p. 4345.

[31] Nick Jakobi, Phil Husbands, and Inman Harvey. "Noise and the reality gap: the use of simulation in evolutionary robotics". In: *Advances in Artificial Life: Third european conference on artificial life*. Ed. by F. Morán, A. Moreno, Juan J. Merelo, and P. Chacón. Vol. 929. Lecture Notes in Artificial Intelligence. Berlin, Germany: Springer, 1995, pp. 704–720.

[32] Sanza Kazadi. "Model independence in swarm robotics". In: *International Journal of Intelligent Computing and Cybernetics* 2.4 (2009), pp. 672–694.

[33] R Kerry, MA Oliver, and ZL Frogbrook. "Sampling in precision agriculture". In: *Geostatistical applications for precision agriculture*. Springer, 2010, pp. 35–63.

[34] Antoine Ligot and Mauro Birattari. "Simulation-only experiments to mimic the effects of the reality gap in the automatic design of robot swarms". In: *Swarm Intelligence* (2019), pp. 1–24.

[35] Antoine Ligot, Ken Hasselmann, and Mauro Birattari. "AutoMoDe-Arlequin: neural networks as behavioral modules for the automatic design of probabilistic finite state machines". In: *Swarm Intelligence – ANTS*. Ed. by Marco Dorigo, Thomas Stützle, María J. Blesa, Christian Blum, and Volker Strobel. Vol. 12421. LNCS. Cham, Switzerland: Springer, 2020, pp. 109–122.

[36] Antoine Ligot, Jonas Kuckling, Darko Bozhinoski, and Mauro Birattari. "Automatic modular design of robot swarms using behavior trees as a control architecture". In: *PeerJ Computer Science* 6 (2020), e314.

[37] Yuri Kaszubowski Lopes, Stefan M. Trenkwalder, André B. Leal, Tony J. Dodd, and Roderich Groß. "Supervisory control theory applied to swarm robotics". In: *Swarm Intelligence* 10.1 (2016), pp. 65–97.

[38] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. "The irace package: iterated racing for automatic algorithm configuration". In: *Operations Research Perspectives* 3 (2016), pp. 43–58.

[39] Sophie Marcuse. "Optimum allocation and variance components in nested sampling with an application to chemical analysis". In: *Biometrics* 5.3 (1949), pp. 189–206.

[40] Francesco Mondada, Michael Bonani, Xavier Raemy, Jim Pugh, Christopher Cianci, Adam Klaptocz, Stéphane Magnenat, Jean-Christophe Zufferey, Dario Floreano, and Alcherio Martinoli. "The e-puck, a robot designed for education in engineering". In: *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*. Ed. by Paulo Gonçalves, Paulo Torres, and Carlos Alves. Castelo Branco, Portugal: Instituto Politécnico de Castelo Branco, 2009, pp. 59–65.

[41] Carlo Pinciroli, Vito Trianni, Rehan O'Grady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni A. Di Caro, Frederick Ducatelle, Mauro Birattari, Luca Maria Gambardella, and Marco Dorigo. "ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems". In: *Swarm Intelligence* 6.4 (2012), pp. 271–295.

[42] Andreagiovanni Reina, Gabriele Valentini, Cristian Fernández-Oto, Marco Dorigo, and Vito Trianni. "A design pattern for decentralised decision making". In: *PLOS ONE* 10.10 (2015), e0140950.

[43] Muhammad Salman, Antoine Ligot, and Mauro Birattari. "Concurrent design of control software and configuration of hardware for robot swarms under economic constraints". In: *PeerJ Computer Science* 5 (2019), e221.

[44] Fernando Silva, Miguel Duarte, Luís Correia, Sancho Moura Oliveira, and Anders Lyhne Christensen. "Open issues in evolutionary robotics". In: *Evolutionary Computation* 24.2 (2016), pp. 205–236.

[45] Robert R Sokal, F James Rohlf, et al. *Biometry: the principles and practice of statistics in biological research*. 1995.

[46] Vito Trianni. *Evolutionary Swarm Robotics*. Berlin, Germany: Springer, 2008.