Transferability in the automatic off-line design of robot swarms: from sim-to-real to embodiment and design-method transfer across different platforms

Miquel Kegeleirs[1*], *Member, IEEE*, David Garzón Ramos[1*], *Member, IEEE*, Ken Hasselmann[2], Lorenzo Garattoni[3], Gianpiero Francesca[3], and Mauro Birattari[1], *Senior Member, IEEE*

***Abstract*—Automatic off-line design is an attractive approach to implementing robot swarms. In this approach, a designer specifies a mission to be accomplished by the swarm, and an optimization process generates suitable control software for the individual robots through computer-based simulations. Most relevant literature has focused on effectively transferring control software from simulation to physical robots. Here, we investigate i) whether the design methods that generate control software are transferable across robot platforms and ii) whether control software generated via such methods is itself transferable. We experiment with two ground mobile platforms with equivalent functional capabilities. Our measure of transferability is based on the performance drop observed when control software and/or design methods are ported from one platform to another. Results indicate that, while the control software generated via automatic design is possibly transferable, better performance can be achieved when a transferable method is directly applied to the new platform.**

***Index Terms*—Automatic design, swarm robotics, transferability, methods and tools for robot system design, evolutionary robotics.**

I. Introduction

In robotics, the notion of *transferability* has been associated with different problems: simulation to reality (sim-to-real) transfer, embodiment transfer, task/skill transfer, knowledge transfer—among others, see [1]–[3]. The most common of these problems, sim-to-real transfer, or how to generate control software that is robust to crossing the reality gap, has received particular attention in swarm robotics, and efficient approaches to tackle it are now available [4]. In this letter, we discuss how a similar notion of transferability applies to two other problems in the automatic design of robot swarms: embodiment transfer and design-method transfer.

A robot swarm [5], [6] is a highly redundant group of robots that operate autonomously without relying on centralized control or external infrastructure. Instead, the robots rely on local sensing and communication to self-organize [7]. By acting collectively, the robots can accomplish missions that they could not accomplish individually [8]. Designing the collective behavior of a swarm is challenging. No universally applicable methodology exists for developing the control software of the individual robots so that a desired collective behavior emerges [9]. Typically, designers manually refine control software until the desired collective behavior is obtained. Yet, this trial-and-error process is costly, time-consuming, and does not guarantee reproducible or transferable results. Automatic off-line design [10], [11] is an appealing alternative. In this approach, the problem of designing control software is reformulated as an optimization problem. Given mission specifications and a platform description, an optimization algorithm searches for suitable control software for the robots—that is, a suitable aggregation of available building blocks into a predefined control architecture (e.g., basic modular behaviors or neurons of an artificial neural network) and/or appropriate values for their parameters. The design process is conducted via computer-based simulations, and the resulting control software is then transferred to physical robots and assessed in the target environment.

The automatic design of robot swarms can be seen as a reinforcement learning problem: the goal is to find a policy—i.e., a behavioral rule mapping observations into actions—that maximizes a reward (expressed by a given objective function) [12]. As Kaelbling et al. pointed out [13], classical automatic design methods—e.g., those based on neuroevolution [14]—perform a direct search in the space of behaviors without relying on the notion of state (which typically cannot be estimated by the individual agent) and the related notion of value of a state.

Due to the sim-to-real transfer problem, control software produced in simulation via automatic off-line design suffers from the effects of the *reality gap* [15]: unavoidable differ-

Manuscript received: July, 23, 2023; Revised October, 15, 2023; Accepted January, 16, 2024.

This paper was recommended for publication by Editor Tetsuya Ogata upon evaluation of the Associate Editor and Reviewers' comments. This work was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (DEMIURGE Project, grant agreement No 681872), Belgium's Wallonia-Brussels Federation through the ARC Advanced project GbO-Guaranteed by Optimization, Belgian Fonds de la Recherche Scientifique – FNRS, and the Colombian Ministry of Science, Technology and Innovation – Minciencias.

[1]Miquel Kegeleirs, David Garzón Ramos, and Mauro Birattari are with IRIDIA, Université libre de Bruxelles, Belgium; [2]Ken Hasselmann is with Royal Military Academy, Belgium; [3]Lorenzo Garattoni and Gianpiero Francesca are with Toyota Motor Europe. Correspondence to mauro.birattari@ulb.be.

[*]Miquel Kegeleirs and David Garzón Ramos contributed equally to this work and should be recognized as co-first authors. Original software was implemented by Miquel Kegeleirs. `Chocolate` and `EvoStick` were developed by Gianpiero Francesca. `NEAT`, `CMA-ES`, and `xNES` were provided and adapted by Ken Hasselmann. The experiments were performed by Miquel Kegeleirs with the assistance of David Garzón Ramos. The manuscript was drafted by David Garzón Ramos and revised by Miquel Kegeleirs and Mauro Birattari. All authors contributed to the development of the ideas, read the manuscript, and provided comments. The research was directed by Mauro Birattari.

Digital Object Identifier (DOI): see top of this page.

ences between simulation and reality can cause a performance drop [4], [16]–[18]. When comparing design methods, the smaller the performance drop, the greater a method's ability to cross the reality gap.

Building on this idea, we study the transferability of automatic design methods and control software across different platforms. Transferring a design method from one platform to another enables the effortless design of new tailored robot swarms—at the expense of the high computational power required in the design process. This also spares a human designer the complex task of developing from scratch an automatic method for a new platform. On the other hand, directly transferring the control software enables a faster and less expensive deployment of robot swarms, however not fully tailored for platform at hand.

We empirically show that transferring either the design methods or the control software gives rise to challenges akin to those presented by the reality gap. We do so by following experimental protocols previously introduced to study the sim-to-real problem in the automatic design of robot swarms. We conduct two experiments: 1) six methods evaluated in simulation with swarms of twenty robots and 2) two methods evaluated in simulation and in reality with swarms of three robots. In each experiment, we perform two studies: A) an automatic design method conceived for one platform is used to generate control software for another (i.e., design-method transfer), and B) control software designed for one platform is deployed to another (i.e., embodiment transfer). The two platforms considered are ground robots endowed with functionally equivalent sensing and actuation capabilities.

II. Related work

The sim-to-real transfer problem is a crucial challenge in robotics and has gained growing attention in the recent years. Several ideas have been proposed to address this problem—for recent reviews focusing on reinforcement learning, see [1], [2]. Among the most prominent ones, domain randomization and domain adaption aim at improving simulation to generate control software that is directly transferable [19], [20]. Indeed, a key hypothesis is that the effects of the reality gap are the result of unfaithful simulation, in other words, simulators are not able to reproduce the complexity of the real world [1], [2]. However, in most studies the control software is typically assessed in simulation only and never transferred to the real robots [1], [2]. Rather than finding a solution to successfully cross the reality gap, other alternatives focus on how to avoid its occurrence. For example, transfer learning comprises two steps: a behavior is first generated in simulation and is then used as a target to train the robots directly in the real world [21], [22]. The main limitation of transfer learning is that it is time-consuming and involves using physical robots in the design process. Hybrid approaches are also emerging, where high-fidelity simulation and randomization are coupled with the two-step process of transfer learning [23].

Unfortunately, the ideas proposed are tailored to specific missions. It is, therefore, difficult to compare them and understand whether they could be reused or ported to another mission. A general, mission-independent approach is missing.

Further, possibly as a consequence of this lack of generality, no proper metric to evaluate transferability exists. Several studies associate the notion of robustness against perturbations with the one of sim-to-real transferability: the hypothesis is that if an instance of control software performs well in simulation on several variants of the target scenario, it would transfer well to the real world—which is seen as yet another variant. Typically, studies relying on this hypothesis do not eventually validate the results through real-robot experiments.

Finally, the literature on sim-to-real transfer has primarily focused on specific domains such as manipulation [24] and aerial navigation [25], while contributions from other domains, such as swarm robotics [17], are comparatively less prevalent.

Regarding other transferability problems, the literature is sparser. Often, the terminology used to describe these problems lacks consistency, making it difficult to categorize the different contributions. In particular, little work exists on embodiment transfer in the sense of transferring control software from one robot platform to another. Liu et al. [26] proposed an evolutionary model to transfer policies from robot to robot, but the assessment was done in simulation only. Bozcuoğlu et al. [27] and Kazhoyan et al. [28] performed extensive skill and knowledge transfer experiments on two different household robots to manipulate kitchen furniture and objects. However, most experiments were done with multiple units of one of the two platforms (the PR2 robot), with robot-to-robot transfer being limited to high-level plans rather than full control software. Design-method transfer is more specific to swarm robotics—as automatic design is prominent in this field.

III. Transferability in swarm robotics

In the automatic design of robot swarms, sim-to-real transfer has been extensively studied, with emphasis on the notion of reality gap introduced by Jakobi et al. [15] in neuroevolutionary robotics. Similarly to many reinforcement learning methods, neuroevolution does not typically cross the reality gap successfully [18].

Concerning sim-to-real transfer, the literature on neuroevolution shares similar hypotheses with the one on (deep) reinforcement learning, notably the one that the sim-to-real problem is a consequence of the fact that simulations are a too simplistic representation of the real word, which is more complex [29], [30]. As a consequence, the literature emphasizes the adoption of accurate simulators to address the problem. Ligot et al. [17] proposed a different hypothesis: the sim-to-real problem is the result of the differences between simulation and the real world, independently on whether one is more or less complex than the other. Hence, to cross the reality gap, methods that are intrinsically robust to the transition from one to the other might be more successful than methods relying on accurate simulation.

As an alternative to neuroevolution, Francesca et al. proposed AutoMoDe [4], an automatic modular design method that is intrinsically resilient to the reality gap. Birattari et al. [31] argued that the high representational power of the control software produced by neuroevolutionary methods, the neural networks, impairs its ability to cross the reality gap: the sim-to-real problem faced in the automatic design of robot swarms is reminiscent of the generalization problem faced in machine learning. AutoMoDe builds control software by assembling predefined modules, which reduces the representation power of the control software and increases its ability to cross the reality gap. Several studies in automatic modular design have shown good sim-to-real transfer [32]–[34].

Even though approaches to address the sim-to-real problem have been proposed, few studies focused on other transferability problems. As a first step in embodiment transfer, Kaiser et al. [35] developed a ROS2 package providing some predefined, manually designed, hardware-independent swarm behaviors that can be reused by different robots. However, generic control software automatically designed for a specific robot platform has never been reused directly on another platform.

Control software for robot swarms is often generated by an *(automatic) design method* [10] Hence, it is also possible to study how a design method, typically conceived for a specific platform, can be transferred to another one. The neuroevolution literature provides examples of ad-hoc adaptation and transformation of methods to apply them to various platforms and missions [36]. This indicates that, with some expert intervention, transferring neuroevolutionary methods is possible. Yet, although a method can occasionally resemble a previous one, no direct, unmodified transfer has been reported so far. The same applies to AutoMoDe: some variants of `Chocolate` [4] were applied to slightly different robots—e.g., robots endowed with communication capabilities [37] and robots that can perceive colors [33]. Some manually-applied adaptation of `Chocolate` was required, including modifying algorithms and redefining modules. Yet, methods of the AutoMoDe family were so far conceived for the e-puck robot, and differences between them are restricted to handling the addition of a new sensor/actuators. Neither in neuroevolution nor in AutoMoDe, the focus has ever been on creating methods that are intrinsically transferable between platforms, or that can produce control software that is transferable.

We contend that the methods and tools developed to address sim-to-real transfer can be applied to these other less studied transferability problems in the context of the automatic design of functionally equivalent robot swarms.

Our first working hypothesis is that transferring software from a platform X to a platform Y is akin to transferring software from an environment E_X (simulation) to an environment E_Y (reality). Indeed, in accordance with the statement of Ligot et al. [17], the reality gap occurs because E_X and E_Y are different, not because of their relative complexity.

Our second working hypothesis is that functionally equivalent robots can execute equivalent control software, which can be designed by the same design method. A key notion in our research is indeed that of *functionally equivalent robots*: two robots X and Y are functionally equivalent if 1) for each sensor of robot X, there is a sensor in robot Y that provides similar data, 2) for each actuator of robot X, there is an actuator in robot Y that provides a similar capability, and 3) there is a consistent ratio between the dimensions of the two robots and the ranges of sensing/action of their sensors/actuators. The reference model system developed by Hasselmann at al. [38] allows one to directly transfer a method from one platform to another by modifying the platform's model, without altering the method itself. We relied on these ideas to conceive a new platform that preserves the functional capabilities of the e-puck, while being larger and operating with different hardware [39].

TABLE I
EXPERIMENTAL SETUP FOR EXPERIMENTS 1 AND 2

Experiments		
Platforms	Mercator (M), e-puck (E)	
Missions	AGGREGATION, FORAGING, GRID EXPLORATION	
	Experiment 1	**Experiment 2**
Setup		
Number of robots	20M, 20E	3M, 3E
Arena size (m^2)	M: 73.95, E: 4.88	M: 5.49, E: 0.36
Design methods		
AutoMoDe	`Chocolate`, `Maple`	`Chocolate`
Neuroevolution	`EvoStick`, `NEAT`, `CMA-ES`, `xNES`	`EvoStick`
Protocol		
Design phase	Simulation	Simulation
Deployment phase	Pseudo-reality	Reality

Transferability analysis			
Study	**Label**	**Designed for**	**Tested on**
A	`EE`	e-puck	e-puck
A	`MM`	Mercator	Mercator
	`EM`	e-puck	Mercator
	`ME`	Mercator	e-puck
B	`EE+MM`	e-puck | Mercator	e-puck | Mercator
B	`EM+ME`	e-puck | Mercator	Mercator | e-puck

To prove our hypotheses, we conceived experiments to evaluate A) design-method and B) embodiment transfer. Our goal is to promote reusability of software in swarm robotics by transferring existing work to new robotics platforms, rather than starting from scratch. It is also our contention that, by extending the concept of reality gap to a broader *transfer gap*, future work on less studied transfer problems will benefit from the long experience of sim-to-real transfer.

IV. EXPERIMENT 1

Table I summarizes the experimental setup. We consider two swarms: one comprising twenty e-puck [40] robots, and the other, twenty Mercator [39] robots. The swarm size is the same we used in previous studies [16], [18], [32]. The sensing and actuation capabilities of the two platforms can be formally described by the same reference model RM 1.2 [38]—see Table II. While the two platforms are equipped with different hardware, their capabilities are functionally equivalent. This is key to enabling the transferability between the two platforms and allows a comparison. The e-puck and Mercator (Fig. 1) differ in size, with the e-puck being roughly one-third the size of the Mercator. They also differ in linear speed and sensor range. However, their speed/size and sensor-range/size ratios are approximately the same.

We design control software for e-pucks and Mercators using automatic methods originally conceived for the e-puck: two modular design methods of the AutoMoDe [31] family,

TABLE II
REFERENCE MODEL RM 1.2

Input	e-puck	Mercator	Description
$prox$	$([0,1]\,;[-1,1]\,\pi)$	$([0,1]\,;[-1,1]\,\pi)$	proximity vector
$light$	$([0,1]\,;[-1,1]\,\pi)$	$([0,1]\,;[-1,1]\,\pi)$	light vector
gnd	$\{b,g,w\}$	$\{b,g,w\}$	ground reading
n	$[0,2]$	$[0,2]$	no. of neighbors
V	$([0,1]\,;[-1,1]\,\pi)$	$([0,1]\,;[-1,1]\,\pi)$	neighbors vector
Output	**e-puck**	**Mercator**	**Description**
$v_{k\in\{l,r\}}$	$[-10,10]$ cm/s	$[-30,30]$ cm/s	target velocity

Period of the control cycle: 0.1 s

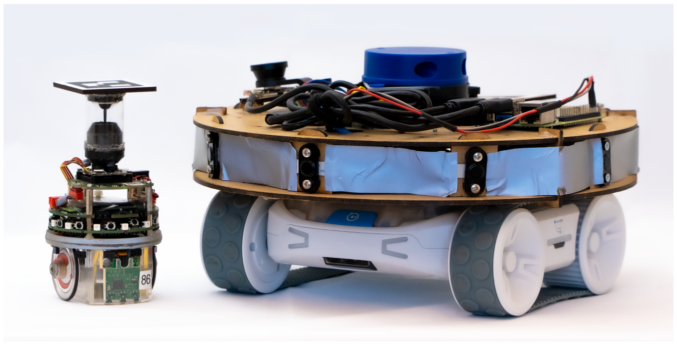
Fig. 1. From left to right, e-puck and Mercator robots.

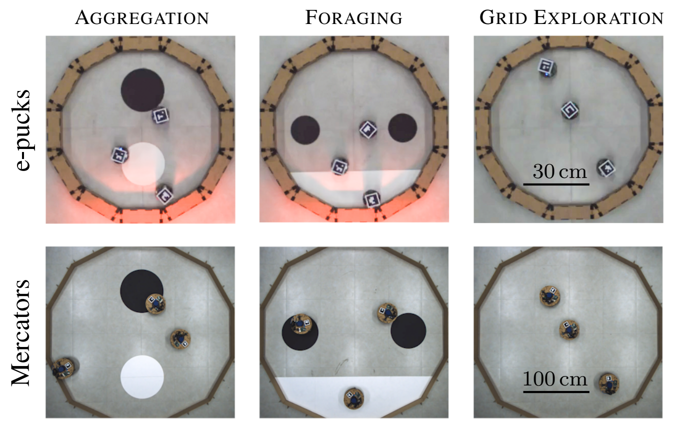

Fig. 2. Experimental scenarios in AGGREGATION, FORAGING, and GRID EXPLORATION (illustrated with the environments used in Experiment 2). The workspace of the e-pucks is about one-third the size of Mercators' workspace.

`Chocolate` and `Maple`, and four neuroevolutionary design methods [18], `EvoStick`, `NEAT`, `CMA-ES` and `xNES`.[1] `Chocolate` [16] is the state-of-the-art AutoMoDe method and produces control software in the form of finite-state machines. `Maple` [32] is a variant of `Chocolate` that produces control software in the form of behavior trees, and is identical to `Chocolate` in every other aspect. `EvoStick` [4] is a straightforward implementation of the neuroevolutionary approach. `NEAT` [41] (neuroevolution of augmenting topologies) is an evolutionary algorithm that automatically shapes the network topology and tune its parameters. The hyper-parameters are tuned as originally recommended by Stanley and Miikkulainen [41]. `CMA-ES` [42] (covariance matrix adaptation evolutionary strategy) is considered as one of the most effective evolutionary algorithms available. The values of the hyper-parameters are those used by Hasselmann et al. [18]. `xNES` [43] (exponential natural evolutionary strategies) is a popular variant of `CMA-ES`. The values of the hyper-parameters are the same as in `CMA-ES`. Previous results confirm that all the methods considered here can generate control software for various missions, including aggregation, foraging, and coverage [4], [16]–[18].

We consider three missions: AGGREGATION, FORAGING, and GRID EXPLORATION—see Fig. 2.

In AGGREGATION, the robots must aggregate on a black area. The environment also comprises a white area and a light source placed outside the arena, on the side of the black area. The robots can use the light and the white area to orientate themselves. The objective function to maximize is $F_{AAC} = \sum_{t=1}^{T} N(t)$, where T is the duration of the experiment and $N(t)$ is the number of robots on the black area at time t.

In FORAGING, the robots must retrieve as many objects as possible from two sources (black circular areas) and deposit them in the nest (white area). Because the platforms we consider have no grasping capabilities, we abstract the actions of retrieving and depositing objects: a robot retrieves an object when entering a source and deposits it when entering the nest. The objective function to maximize is $F_{FTS} = N_o$, where N_o is the total number of objects retrieved and deposited.

In GRID EXPLORATION, robots must explore the environment virtually divided in a 10x10 grid and continuously visit every cell of the grid. For each cell, we record the time t elapsed since the last time it was visited by a robot. Each time the cell is visited by a robot, t is reset to 0. The objective function to maximize is $F_{GE} = \sum_{i=1}^{T}(\frac{1}{N_{cells}}\sum_{j=1}^{N_{cells}} -t_{ij})$, where T is the duration of the experiment, N_{cells} is the total number of cells, and t_{ij} is the elapsed time at simulation time i since the cell j was visited by a robot.

We adjust the size of each environment according to the relative size of the robots: the e-pucks operate in an environment that is one-third the size of the Mercators' one. Each experiment lasts for $T = 120$ s. We produce a total of 360 instances of control software using `Chocolate`, `Maple`, `EvoStick`, `NEAT`, `CMA-ES`, and `xNES`—ten for each platform, mission, and method.[2] The design phase is conducted in simulation and the deployment phase is conducted in pseudo-reality. The notion of pseudo-reality was introduced by Ligot et al. [17] as a simulation model, different from the one used in the design process, that is to be used to evaluate control software. Ligot et al. showed that there is a correlation between the performance drop experienced by control software when transferred from simulation (the model used for the design) to pseudo-reality, and the one experienced when transferred from simulation to real robots [17], [34].

A design method produces every instance of control software with a budget of 100.000 simulations. We assess each

[1]The four neuroevolutionary methods are initialized with a fully connected feed-forward neural network and no hidden layer.

[2]The control software produced by all the methods is available at https://github.com/demiurge-project/Results-Transferability.git

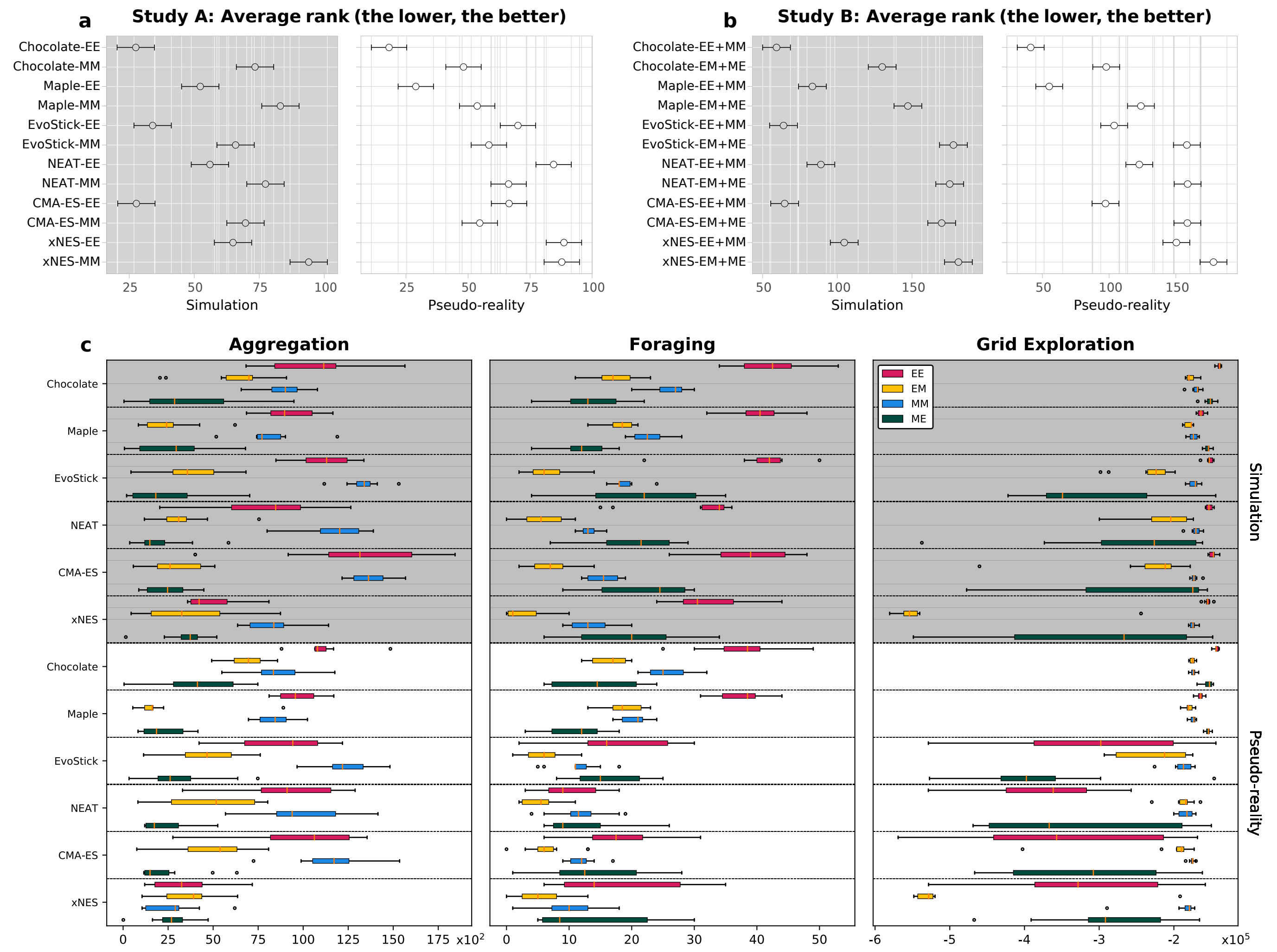

Fig. 3. Experiment 1: a) Study A, comparison of design methods when assessed on the platform for which they were conceived—e-pucks (EE)—and when assessed on the other—Mercators (MM); b) Study B, comparison of control software when assessed on the platform for which it has been produced (EE+MM) and when assessed on the other platform (EM+ME); c) raw results on the three missions. In a) and b), results are aggregated across the three missions using a Friedman test. For each method and platform, we present average ranks and 95% confidence intervals. Results displayed on a gray background are obtained in simulation; those on a white background, in pseudo-reality.

instance once in simulation and once in pseudo-reality, both on the platform on which it was designed and on the other. Concerning the pseudo-reality models, for the e-pucks we use the one defined by Ligot et al. [17]; for the Mercators we derived an equivalent model from the one of the e-puck. The evaluation of the performance in each of the three missions is represented by boxplots. Additionally, two Friedman tests [44] present aggregated results to support discussion on design-method and embodiment transfer—studies A and B, respectively. Any statement like “X performs significantly better/worse than Y” means that the confidence intervals of the Friedman test for X and Y do not overlap. This protocol has been used in [4], [16], [17], [32], [37] and is further discussed in [45]. We conduct simulations in ARGoS3 [46], a simulator widely used in swarm robotics research.

Study A – Design-method transfer: We aggregate the performance across the three missions using a Friedman test—see Fig. 3-a. In simulation, the control software produced by all methods demonstrates meaningful behavior and effectively performs the missions for both e-pucks and Mercators. In general, the control software produced by neuroevolution (`EvoStick`, `NEAT`, `CMA-ES`, and `xNES`) performs similarly or better than the one produced by AutoMoDe (`Chocolate` and `Maple`)—with the exception of FORAGING, for which the control software produced by `NEAT` and `xNES` perform significantly worse than the others, on both platforms. In pseudo-reality, the control software produced by AutoMoDe maintains satisfactory behavior on both platforms. In contrast, the control software produced by neuroevolution methods does not typically reproduce simulation results on either platform and suffers a significant drop in performance. In general, the control software produced by neuroevolution performs similarly or worse than the one produced by AutoMoDe—with the exception of `NEAT` for AGGREGATION on the Mercators, but it still suffers a significant drop in performance in comparison with the simulation.

Study B – Embodiment transfer: Also in this case, we aggregate the performance across the three missions using a Friedman test—see Fig. 3-b. In simulation, before the transfer, the control software produced by neuroevolution performs better than the one produced by AutoMoDe. After transferring the control software, we observe that the one produced by AutoMoDe methods performs better than the one produced by neuroevolutionary methods—see Fig. 3-c. There are two notable exceptions: i) in AGGREGATION, control software produced by Maple for the e-puck does not transfer well to the Mercator; ii) in FORAGING, control software produced by neuroevolutionary methods for the Mercator transfers better to the e-puck than the one produced by AutoMoDe methods. This is due to functional differences between the e-puck and the Mercator.[3] Yet, overall, we observe a larger performance drop when transferring the control software for neuroevolutionary methods than for AutoMoDe method. In pseudo-reality, both before an after the transfer, AutoMoDe methods yield significantly better results than neuroevolutionary ones—see Fig. 3-b. The performance drop is similar for all methods but, after the transfer, neuroevolutionary methods essentially yields poor behaviors while AutoMoDe ones keep satisfactorily behaviors.

V. EXPERIMENT 2

Experiment 2 is similar to Experiment 1, but is performed with real robots in the deployment phase. The following are the main differences in the experimental setup: i) we consider swarms of three robots (the size is an experimental limitation due to the fact that only three Mercators are available at the moment); ii) we evaluate two methods `Chocolate` and `EvoStick`. We selected these methods because they are the ones that have been most consistently examined in studies on the automatic design of collective behaviors for e-pucks [31].

The missions are the same as in Experiment 1. The size of each environment is adapted to the smaller number of robots—see Table I. The protocol is the same of Experiment 1, with the only difference that here we produce a total of 120 instances of control software using `Chocolate` and `EvoStick`—ten for each platform, mission, and method. The design phase is conducted in simulation and the deployment phase is conducted in reality. Likewise Experiment 1, a design method produces every instance of control software with a budget of 100.000 simulations. We assess each instance once in simulation and once on physical robots, both on the platform on which it was designed and on the other. Multimedia Materials provide videos of all evaluations on the real robots.

Study A – Design-method transfer: We aggregate the performance across the three missions using a Friedman test—see Fig. 4-a. In simulation, the results are similar to those of Experiment 1. Also the results on real robots are similar to those of pseudo-reality, except that control software produced by `EvoStick` suffers a larger drop in performance: the behaviors of e-pucks are meaningless, and those of Mercators are meaningful but incomplete/suboptimal.

Study B – Embodiment transfer: Also in this case, we aggregate the performance across the missions using a Friedman test—see Fig. 4-b. Simulation results are similar to those of Experiment 1. On the physical robots, `Chocolate` demonstrates a performance drop after transferring the control software between e-pucks and Mercators, but it demonstrates satisfactory behavior in both cases. However, `EvoStick` demonstrates only poor behavior, with or without transfer.

The raw results obtained by the two methods under analysis on the three missions are given in Fig. 4-c.

VI. DISCUSSION

Performance difference between simulation and reality is a known effect of the reality gap: previous research has shown that AutoMoDe is more robust to the reality gap than neuroevolution [17], [18]. Yet, so far, the different degree of robustness to the reality gap between AutoMoDe and neuroevolution had been only reported for e-pucks. The results presented in this letter show that a similar different degree of robustness can be observed also on the Mercators.

Study A – Design-method transfer: In simulation, the swarm of e-pucks performs similarly or better than the one of Mercators, for all methods. This result was expected, as all design methods under analysis were conceived for the e-puck and were applied to Mercators without any adaptation. In (pseudo-)reality, the swarm of e-pucks perform similarly or better than the one of Mercators when they execute control software produced by AutoMoDe methods. On the other hand, the swarm of Mercators performs similarly or better than the e-pucks when they execute control software produced by neuroevolutionary methods. This result was unexpected and suggests that neuroevolutionary methods, although originally conceived for the e-pucks, are more robust to the reality gap when designing control software for Mercators. This indicates that the effects of the reality gap are not only method-dependent but also platform-dependent. Yet, the results obtained by neuroevolution in (pseudo-)reality are in any case outperformed by those obtained with AutoMoDe.

By comparing the relative performance of AutoMoDe methods and neuroevolutionary ones, we can conclude that, under the experimental conditions we considered, AutoMoDe transfers better from e-pucks to Mercators than neuroevolution.

Study B – Embodiment transfer: In simulation, the performance drop caused by the transfer is larger for neuroevolution than for AutoMoDe. In Experiment 2, we observe a rank inversion between `Chocolate` and `EvoStick`. Similar observations can be made in Experiment 1, for example between `Maple` and `EvoStick` or `CMA-ES`. It is known that `EvoStick` can achieve good performance in simulation by overfitting the design process to the simulated model of the e-puck [17]. We argue that this prevented a proper transfer to Mercator—as it happens when transferring control software between simulation and reality. In other words, the notion of overfitting used to explain differences in sim-to-real transfer

[3]In particular, as Mercators navigate at higher speed than the e-pucks, they are programmed to avoid obstacles from farther, making obstacle detection more sensitive. On the other hand, AutoMoDe modules, originally designed for e-pucks, use less sensitive obstacle avoidance. As a result, in the first case, the obstacle sensitivity of the Mercators conflicts with the modules of `Maple`. In the second case, the e-pucks benefit from higher obstacle sensitivity—hence less collisions—in the neuroevolutionary case, but not with AutoMoDe, which is restricted to use predefined modules.

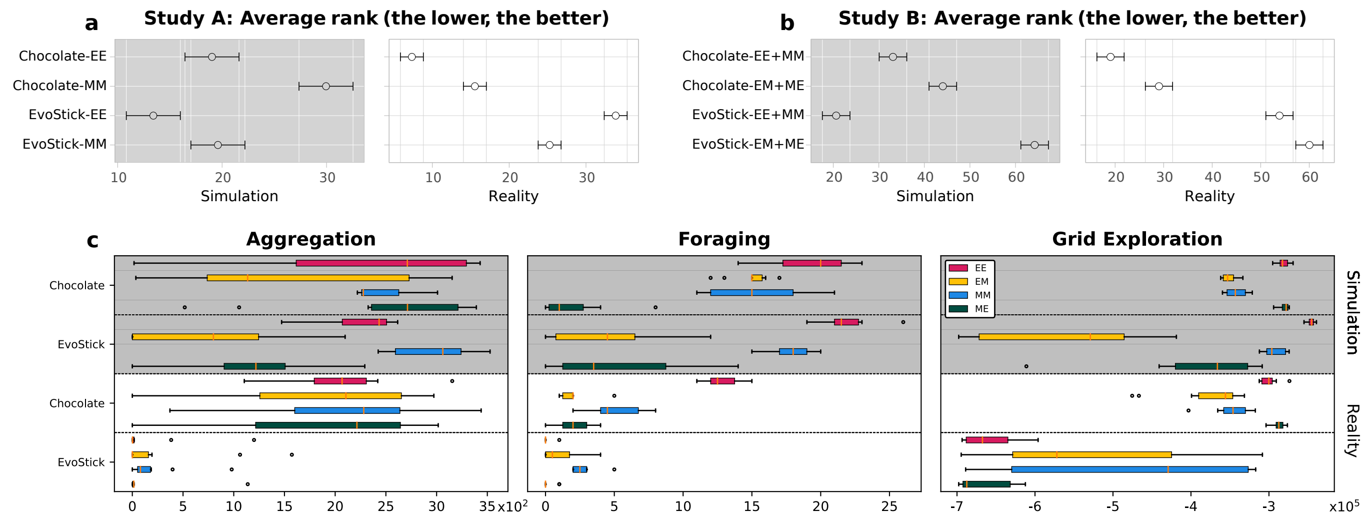

Fig. 4. Experiment 2: see caption of Fig. 3 for a description of the content. Here, results displayed on a gray background are obtained in simulation; those on a white background, with real robots.

seems to apply similarly to robot-to-robot transfer. When examining the results, it appears that this performance drop occurs independently of which robot is the source and which is the target. This is coherent with the interpretation of the reality gap according to which the performance drop observed when porting control software from simulation to reality is due to differences between the source and target environments—irrespective of their relative complexity [17].

In (pseudo-)reality, the results show that, although all methods suffer from a performance drop, the control software produced by neuroevolution is particularly affected and barely demonstrates meaningful behaviors. This is particularly clear in Experiment 2, where `EvoStick` demonstrate poor behaviors, with or without transfer. This confirms that, overall, control software produced by AutoMoDe transfers better than the one produced by neuroevolution.

Finally, the comparison of the results of the two studies, in both Experiment 1 and Experiment 2, shows that better results are obtained by transferring the design method (Study A) than by transferring the control software (Study B), for both AutoMoDe and neuroevolutionary methods. Indeed, for each platform, results obtained when the control software is designed for that platform are always similar or better than those obtained when the control software is transferred from the other platform (EE better than ME, MM better than EM—see Fig. 3-c and Fig. 4-c).

VII. Conclusion

The results show that automatic design methods and the control software they produce can be transferred from one robot platform to another, provided that the two have equivalent sensing and actuation capabilities. The best results are obtained by transferring a method, that is, by applying an automatic design method originally conceived for a platform to another one, as opposed to transferring the control software it produces for the original platform to the other one.

Yet, control software designed by an automatic design method still exhibit meaningful behavior when transferred to another platform. We will further investigate if control software produced for a platform, and already available, can be used as a starting point in a design process to generate control software for another platform—so as to improve and/or speed up the design process itself. Still, this requires to first transfer the design method to the new platform, or to translate the obtained control software for the new method.

The results also show that neuroevolutionary methods suffer a larger performance drop than AutoMoDe ones when transferred to another platform. This holds also when transferring control software. That is, AutoMoDe is more robust to the design-method transfer and embodiment transfer. AutoMoDe has proven to be intrinsically more robust to the reality gap than neuroevolution. Its robustness to the sim-to-real transfer is a property given by its modular nature and the restricted representational power of the control software produced [18]. We conclude that methods that are resilient to the reality gap—the transfer from one model to another—are resilient also to other types of transfer. It is our contention that the notion of reality gap can be generalized into a broader *transfer gap*. To corroborate this contention, future work should investigate whether protocols to predict the robustness of design methods to the reality gap [34] can be used to predict the transferability of control software across platforms.

As of today, there are several platform-specific methods that can automatically generate collective behaviors for robot swarms. Our contribution is meant to motivate further research on the evaluation of these methods in more capable robot platforms, and on the realization of new methods that are intrinsically robust to the transfer process. Constraining the representational power of the generated control software is a key factor in achieving this robustness. In modular design, this comes naturally. In neuroevolution, regularization techniques could be used to improve resilience [18].

REFERENCES

[1] E. Salvato, G. Fenu, E. Medvet, and F. A. Pellegrino, "Crossing the reality gap: A survey on sim-to-real transferability of robot controllers in reinforcement learning," *IEEE Access*, vol. 9, pp. 153 171–153 187, 2021.

[2] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *SSCI 2020*. IEEE, 2020, pp. 737–744.

[3] C. Devin, A. Gupta, T. Darrell, P. Abbeel, and S. Levine, "Learning modular neural network policies for multi-task and multi-robot transfer," in *ICRA 2017*. IEEE, 2017, pp. 2169–2176.

[4] G. Francesca, M. Brambilla, A. Brutschy, V. Trianni, and M. Birattari, "AutoMoDe: a novel approach to the automatic design of control software for robot swarms," *Swarm Intell.*, vol. 8, no. 2, pp. 89–112, 2014.

[5] E. Şahin, "Swarm robotics: from sources of inspiration to domains of application," in *SAB 2004*, ser. LNCS, vol. 3342. Springer, 2005, pp. 10–20.

[6] G. Beni, "From swarm intelligence to swarm robotics," in *SAB 2004*, ser. LNCS, vol. 3342. Springer, 2005, pp. 1–9.

[7] M. Dorigo, M. Birattari, and M. Brambilla, "Swarm robotics," *Scholarpedia*, vol. 9, no. 1, p. 1463, 2014.

[8] M. Dorigo, G. Theraulaz, and V. Trianni, "Swarm robotics: past, present, and future [point of view]," *Proceedings of the IEEE*, vol. 109, no. 7, pp. 1152–1165, 2021.

[9] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo, "Swarm robotics: a review from the swarm engineering perspective," *Swarm Intell.*, vol. 7, no. 1, pp. 1–41, 2013.

[10] M. Birattari, A. Ligot, D. Bozhinoski, M. Brambilla, G. Francesca, L. Garattoni, D. Garzón Ramos, K. Hasselmann, M. Kegeleirs, J. Kuckling, F. Pagnozzi, A. Roli, M. Salman, and T. Stützle, "Automatic off-line design of robot swarms: a manifesto," *Front. Robot. AI*, vol. 6, p. 59, 2019.

[11] M. Birattari, A. Ligot, and K. Hasselmann, "Disentangling automatic and semi-automatic approaches to the optimization-based design of control software for robot swarms," *Nat. Mach. Intell.*, vol. 2, no. 9, pp. 494–499, 2020.

[12] S. Doncieux, N. Bredeche, J.-B. Mouret, and A. Eiben, "Evolutionary robotics: what, why, and where to," *Front. Robot. AI*, vol. 2, p. 4, 2015.

[13] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: a survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[14] S. Nolfi, *Behavioral and Cognitive Robotics: An Adaptive Perspective*. Institute of Cognitive Sciences and Technologies, National Research Council, 2021.

[15] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: the use of simulation in evolutionary robotics," in *ECAL '95*, ser. LNAI, vol. 929. Springer, 1995, pp. 704–720.

[16] G. Francesca, M. Brambilla, A. Brutschy, L. Garattoni, R. Miletitch, G. Podevijn, A. Reina, T. Soleymani, M. Salvaro, C. Pinciroli, F. Mascia, V. Trianni, and M. Birattari, "AutoMoDe-Chocolate: automatic design of control software for robot swarms," *Swarm Intell.*, vol. 9, no. 2–3, pp. 125–152, 2015.

[17] A. Ligot and M. Birattari, "Simulation-only experiments to mimic the effects of the reality gap in the automatic design of robot swarms," *Swarm Intell.*, vol. 14, pp. 1–24, 2020.

[18] K. Hasselmann, A. Ligot, J. Ruddick, and M. Birattari, "Empirical assessment and comparison of neuro-evolutionary methods for the automatic off-line design of robot swarms," *Nat. Commun.*, vol. 12, p. 4345, 2021.

[19] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *ICRA 2018*, 2018, pp. 3803–3810.

[20] F. Muratore, F. Treede, M. Gienger, and J. Peters, "Domain randomization for simulation-based policy optimization with transferability assessment," in *CoRL 2018*, ser. PMLR, vol. 87. PMLR, 2018, pp. 700–713.

[21] S. Barrett, M. E. Taylor, and P. Stone, "Transfer learning for reinforcement learning on a physical robot," in *AAMAS-ALA 2010*, vol. 1, 2010.

[22] J. Hua, L. Zeng, G. Li, and Z. Ju, "Learning for a robot: Deep reinforcement learning, imitation learning, transfer learning," *Sensors*, vol. 21, no. 4, 2021.

[23] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," in *ICRA 2019*. IEEE, 2019, pp. 8973–8979.

[24] J. Matas, S. James, and A. J. Davison, "Sim-to-real reinforcement learning for deformable object manipulation," in *CoRL 2018*, ser. PMLR, vol. 87. PMLR, 2018, pp. 734–743.

[25] S. Daftry, J. A. Bagnell, and M. Hebert, "Learning transferable policies for monocular reactive MAV control," in *ISER 2016*. Springer, 2017, pp. 3–11.

[26] X. Liu, D. Pathak, and K. M. Kitani, "REvolveR: Continuous evolutionary models for robot-to-robot policy transfer," https://arxiv.org/abs/2202.05244, 2022.

[27] A. K. Bozcuoğlu, G. Kazhoyan, Y. Furuta, S. Stelter, M. Beetz, K. Okada, and M. Inaba, "The exchange of knowledge using cloud robotics," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1072–1079, 2018.

[28] G. Kazhoyan, S. Stelter, F. K. Kenfack, S. Koralewski, and M. Beetz, "The robot household marathon experiment," in *ICRA 2021*. IEEE, 2021, pp. 9382–9388.

[29] J. C. Zagal, J. Ruiz-del Solar, and P. Vallejos, "Back to reality: crossing the reality gap in evolutionary robotics," *IFAC Proceedings Volumes*, vol. 37, no. 8, pp. 834–839, 2004.

[30] S. Koos, J.-B. Mouret, and S. Doncieux, "The transferability approach: crossing the reality gap in evolutionary robotics," *IEEE Trans. Evol. Comput.*, vol. 17, no. 1, pp. 122–145, 2013.

[31] M. Birattari, A. Ligot, and G. Francesca, "AutoMoDe: a modular approach to the automatic off-line design and fine-tuning of control software for robot swarms," in *Automated Design of Machine Learning and Search Algorithms*, ser. NCS. Springer, 2021, pp. 73–90.

[32] A. Ligot, J. Kuckling, D. Bozhinoski, and M. Birattari, "Automatic modular design of robot swarms using behavior trees as a control architecture," *PeerJ Computer Science*, vol. 6, p. e314, 2020.

[33] D. Garzón Ramos and M. Birattari, "Automatic design of collective behaviors for robots that can display and perceive colors," *Applied Sciences*, vol. 10, no. 13, p. 4654, 2020.

[34] A. Ligot and M. Birattari, "On using simulation to predict the performance of robot swarms," *Sci. Data*, vol. 9, p. 788, 2022.

[35] T. K. Kaiser, M. J. Begemann, T. Plattenteich, L. Schilling, G. Schildbach, and H. Hamann, "ROS2SWARM - a ROS 2 package for swarm robot behaviors," in *ICRA 2022*, 2022, pp. 6875–6881.

[36] S. Nolfi, J. Bongard, P. Husbands, and D. Floreano, *Evolutionary Robotics*. Springer International Publishing, 2016.

[37] K. Hasselmann and M. Birattari, "Modular automatic design of collective behaviors for robots endowed with local communication capabilities," *PeerJ Computer Science*, vol. 6, p. e291, 2020.

[38] K. Hasselmann, A. Ligot, G. Francesca, D. Garzón Ramos, M. Salman, J. Kuckling, F. J. Mendiburu, and M. Birattari, "Reference models for AutoMoDe," IRIDIA, ULB, Tech. Rep. TR/IRIDIA/2018-002, 2018.

[39] M. Kegeleirs, R. Todesco, D. Garzón Ramos, G. Legarda Herranz, and M. Birattari, "Mercator: hardware and software architecture for experiments in swarm SLAM," IRIDIA, ULB, Tech. Rep. TR/IRIDIA/2022-012, 2022.

[40] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli, "The e-puck, a robot designed for education in engineering," in *ROBOTICA 2009*. Instituto Politécnico de Castelo Branco, 2009, pp. 59–65.

[41] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, 2002.

[42] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evol. Comput.*, vol. 9, no. 2, pp. 159–195, 2001.

[43] T. Glasmachers, T. Schaul, S. Yi, D. Wierstra, and J. Schmidhuber, "Exponential natural evolution strategies," in *GECCO 2010*. ACM, 2010, pp. 393–400.

[44] W. J. Conover, *Practical Nonparametric Statistics*, 3rd ed., ser. Wiley Series in Probability and Statistics. John Wiley & Sons, 1999.

[45] A. Ligot, A. Cotorruelo, E. Garone, and M. Birattari, "Towards an empirical practice in off-line fully-automatic design of robot swarms," *IEEE Trans. Evol. Comput.*, vol. 26, no. 6, pp. 1236–1245, 2022.

[46] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. A. Di Caro, F. Ducatelle, M. Birattari, L. M. Gambardella, and M. Dorigo, "ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems," *Swarm Intell.*, vol. 6, no. 4, pp. 271–295, 2012.