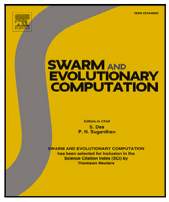
Swarm and Evolutionary Computation 83 (2023) 101395

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

journal homepage: www.elsevier.com/locate/swevo



Automatic modular design of robot swarms based on repertoires of behaviors generated via novelty search

Ken Hasselmann, Antoine Ligot, Mauro Birattari *

IRIDIA, Université Libre de Bruxelles, Av. Roosevelt 50, Brussels, 1050, Belgium

ARTICLE INFO

Keywords:
Swarm robotics
Automatic design
Novelty search
Modular design
Neural networks
Evolutionary robotics
AutoMoDe

ABSTRACT

Several methods have already been proposed to automatically design control software for robot swarms by assembling predefined modules. Yet, so far, the modules on which these methods operate have always been defined manually in a process that is time consuming, requires domain knowledge, and must be performed by an expert. Motivated by the goal of automatizing the definition of these modules, we propose an approach in which repertoires of modules, in the form of neural networks, are automatically generated via a quality-diversity evolutionary algorithm. To illustrate the proposal, we introduce `Nata`, a novel approach belonging to the AutoMoDe family. `Nata` automatically generates probabilistic finite-state machines in which states are selected from a repertoire of neural networks, and transition conditions are selected from a set of rules based on the sensory capabilities of the robotic platform considered. Both the repertoire of neural networks and the set of transition rules are automatically generated a priori, once and for all, in a mission-agnostic way. We study `Nata` on three missions, both in simulation and with real robots. `Nata` is the first modular automatic design method that assembles modules that were themselves generated automatically.

1. Introduction

Every robot of a swarm operates autonomously, only considers local information perceived by itself or shared by neighboring peers, and must cooperate with others to accomplish tasks that are beyond its individual capabilities [1]. The main challenge in swarm robotics is to conceive the behaviors of the individuals so that the many robot–robot and robot–environment interactions that characterize the operation of a swarm lead to a desired collective behavior [2,3]. A promising approach for creating robot swarms is automatic design [4,5].

In automatic design, the mission to be tackled is formally specified by a measure that indicates to which extent the swarm accomplishes it. An optimization algorithm maximizes this measure to find an appropriate individual behavior: in practice, it does so by fine-tuning the parameters of the control software that is then executed on the robots.

The most popular approach to automatically conceive control software for robot swarms is neuroevolution [4,6,7]. With this approach, an instance of control software is a neural network whose synaptic weights – and possibly the topology – are optimized by an evolutionary algorithm [8–11]. The main advantage of the neuroevolutionary approach is its ease of operation: to use it, one only needs to map the readings of the robot's sensors to the input nodes of the neural network, and the values of the output nodes to its actuators. No expertise in swarm robotics is necessary in the definition of the design method. The main drawback is the lack of robustness to the discrepancies between simulations and reality—what is called the reality gap [12,13]. In fact, a neuroevolutionary design method used in a fully automatic context – that is, in a one-shot process that does not allow human adaptations of the method to the problem at hand [14,15] – is likely to produce control software that performs well in simulation, but poorly in reality [16].

Another popular approach to the automatic design of robot swarms comprises modular methods, which generate instances of control software by combining low-level behaviors [17–19]. So far, the modular methods that have been introduced have the downside of requiring human expertise in either their operation or their conception. For example, Duarte et al. [17] proposed a method based on the hierarchical decomposition of complex behaviors into several simpler ones that are then automatically generated via neuroevolution. As the decomposition of complex behaviors is performed by hand and is mission dependent, this method cannot be used in a fully-automatic context.

Francesca et al. [19] proposed AutoMoDe, an approach that automatically combines hand-written modules into a predetermined control architecture. The modules are low-level behaviors executed by the robots, and conditions to transition from one low-level behavior to another. Although the modules are conceived by a human, they are created once and for all in a mission-agnostic fashion, which allows the

* Corresponding author.
E-mail address: mauro.birattari@ulb.be (M. Birattari).

https://doi.org/10.1016/j.swevo.2023.101395
Received 26 October 2022; Received in revised form 16 March 2023; Accepted 27 August 2023
Available online 30 August 2023

AutoMoDe approach to be used in a fully-automatic context [15]. The AutoMoDe approach currently comprises around ten design methods, which have all been shown to produce control software that cross the reality gap satisfactorily [19–28]. In fact, AutoMoDe has been created specifically to overcome the robustness issue faced by the neuroevolutionary approach [19] — see Section 2 for an explanation.

The definition and implementation of the modules that are to be combined automatically is critical to the success of a modular method. By success, we mean the ability of the method to exploit the capabilities of the robotic platform for which control software is conceived so that collective behaviors that are of interest to swarm robotics are produced [29].

A fully-automatic modular method is typically conceived to produce control software for a specific robotic platform, and cannot easily be ported to other ones. One can imagine that robotic platforms of different nature (e.g., air-based, ground-based, water-based), or robotic platforms of the same nature but with different capabilities (e.g., communication, vision, actuation) would require specific modules to appropriately utilize their capabilities. One can also imagine that, if these modules were to be conceived manually, this should be done by an expert in (swarm) robotics to obtain the best possible results.

In this paper, our goal is to conceive a new instance of AutoMoDe that requires less human expertise to be implemented or to be ported to different robotics platforms than the current instances.

In a previous research, we investigated the viability of replacing the hand-coded low-level behavior modules of `Chocolate`, the state-of-the-art AutoMoDe method [20], with automatically generated neural networks [27]. We defined a method, AutoMoDe-`Arlequin`, which produced control software that outperformed the one generated by a classical neuroevolutionary design method. Although these results were promising and opened the door to the exploration of novel design methods, we did not completely free ourselves from human expertise in the conception of `Arlequin`. Indeed, expert knowledge was involved in (i) the hand-coding of the condition modules and (ii) the definition of the performance measures that were used to generate the neural networks by artificial evolution—which is known to be particularly challenging [9,30].

To further dispense from human expertise in the definition of a modular method, we address here the two aforementioned points by (i) proposing a set of rules defined on the sensory capabilities of the robotic platform to automatically generate condition modules, and (ii) adopting an approach to generate repertoires of low-level behaviors in a mission-agnostic way, without the need to define an objective or a performance measure for each of the behaviors. Repertoires are large sets of low-level behaviors that are as diverse as possible. They are created using a quality-diversity algorithm that uses behavioral novelty of candidate behavior as the objective function of an optimization algorithm [31].

We present `Nata`,[1] a design method that automatically generates control software by selecting modules from a repertoire of mission-agnostic behaviors and assembling them into probabilistic finite-state machines. We test `Nata` with physical robots on well-studied swarm robotics missions. The repertoire is generated using novelty search with local competition [33]; the hyperparameters of the novelty search algorithm were selected to produce a repertoire similar to the one presented in [32].

`Nata` is, to the best of our knowledge, the first repertoire-based method to generate and assemble probabilistic finite-state machines for robot swarms, the first swarm robotics repertoire-based method to be tested on real robots, the first modular method that has been generated automatically—that is, the behaviors and transition rules on which the automatic design method operates are themselves generated automatically.

2. Background

As stated in Section 1, AutoMoDe is a modular approach to the automatic design of control software for robot swarms that was introduced to address the issue of the reality gap. The reality gap is the unavoidable difference between reality and the simulation models used during the automatic design process. The introduction of AutoMoDe was motivated by the observation that the reality-gap problem resembles the one of *overfitting* encountered in machine learning [19]. An important result in this domain is the *bias–variance trade-off*, which decomposes the generalization error of a learning algorithm into two terms: bias and variance [34,35]. Bias and variance are known to be correlated to the complexity of learning algorithms: high-complexity algorithms have high variance and low bias, whereas low-complexity ones have low variance and high bias. It is also known that high-complexity algorithms tend to overfit the training set more than less complex ones.

Francesca et al. [19] applied these notions to the automatic design of robot swarm, and conjectured that high-complexity automatic design methods tend to suffer more from the reality-gap problem than low-complexity ones because they are more likely to overfit the simulator used during the design phase. The AutoMoDe approach they proposed [19] has a lower complexity with respect to neuroevolution. Indeed, the control software it can produce is restricted to what can be obtained by selecting and combining a set of pre-defined modules into a given architecture, and by fine-tuning a small set of parameters—these restrictions are effectively a form of bias introduced to lower the variance.

`Arlequin` [27] was introduced to test the conjecture that the robustness to the reality gap of AutoMoDe results from the restriction of the space of the control software that it can generate rather than by the fact that the modules are skilfully hand-crafted. To this end, `Arlequin` assembles modules obtained via neuroevolution—also in this case, the modules are generated a priori and once and for all, in a mission-agnostic way. The modules of `Arlequin` were generated using performance measures defined to obtain behaviors that are qualitatively similar to the hand-crafted modules of `Chocolate`.

Conceiving the appropriate performance measure so that the neuroevolutionary process produces the desired swarm behavior is challenging and is typically done via trial-and-error [9,30]. Repertoire-based methods using quality-diversity algorithms appear to be an interesting step forward in order to free ourselves from the burden of creating such performance measures.

Quality-diversity algorithms, such as MAP-elites [36] or novelty search with local competition [33], are algorithms that explore a given search space to find high-quality solutions to a problem while maximizing their spatial diversity. These algorithms have already been used to create repertoires of behaviors for legged robots [33,36–39], wheeled robots [32,40–42], robotic arms [36,43,44], and UAV's [45]. Most of the works considered *open-loop* controllers [36–40,43–45], that is, the control software modules in the repertoire do not use sensory information and only describe locomotor behaviors. The use of *closed-loop* controllers – that is, control software modules that make use of sensory information – was presented in subsequent works [32,41,42].

In our research, we search the space of possible behaviors – in the form of neural networks –, to build a repertoire of high-quality behavioral modules to be used as building blocks of the robots control software. Gomes and Christensen [32] were the first to use a quality-diversity algorithm to generate a repertoire of swarm behaviors. The authors evaluated all behaviors of the repertoire on eight missions

[1] The methods belonging to the AutoMoDe family all have food-related names: Vanilla, Chocolate, Gianduja, TuttiFrutti... The method presented in this paper is based on the work of Gomes et al. of the University of Lisbon [32]. To acknowledge the source of inspiration and to celebrate the original and inspiring work of our colleagues, we named our method `Nata`, as in "pastéis de nata", the popular custard tarts from Belém, Lisbon.

to assess their quality. Results showed that the repertoire contained suitable solutions for all missions. In that work, behaviors of the repertoire were not assembled, the model of the robot was rather simple, and no real-robot experiments were conducted. Subsequently, Gomes and Christensen [41] presented EvoRBC-II, a method to evolve repertoires and assemble the modules using a supervisor decision tree as an arbitrator that selects the low-level behavior to be executed. The method was assessed on nine single-robot missions but no real-robot experiment was conducted.

3. Modular repertoire-based automatic design

`Nata` belongs to the AutoMoDe family of modular methods. It combines two types of modules – behaviors and conditions – into probabilistic finite-state machines using irace [46]. Irace is the software package implementing the Iterated F-race optimization algorithm; it is used with its default parameters. Iterated F-race is based on F-race [47], a racing procedure [48] originally proposed for the automatic configuration of stochastic optimization algorithms and metaheuristics.

A behavior is an action executed by a robot. A condition is a provision for switching from one behavior to another. The control software of each individual robot is a probabilistic finite-state machine in which each state is a behavior and each edge is associated with a condition that enables it depending on whether it is satisfied. Behaviors and conditions were generated automatically through procedures that we will detail in the following.

In this section, we describe the procedure we followed to create the repertoire of neural networks, we propose a methodology to generate condition modules, and we explain how `Nata` uses its repertoire and the generated conditions to produce control software for robot swarms.

3.1. Generation of a repertoire of behaviors using novelty search with local competition

In this section, we present the idea behind the use of novelty search with local competition and the creation of the repertoire of behaviors of `Nata`.

We created the repertoire of `Nata` following the idea introduced in [32] for building a repertoire of behaviors for robot swarms. Each behavior in the repertoire is a neural network that can be used as control software on a robot.

We generated this repertoire of behaviors using an evolutionary process driven by novelty search with local competition [33,49]. This process follows the framework introduced by Cully and Demiris [43] and used by Gomes and Christensen [32], in which the selection of novel solutions and the construction of the repertoire are two independent steps; the algorithm is summarized in Algorithm 1 and in Fig. 1. We first create an empty repertoire that will hold the set of best candidate neural networks and will eventually become the final repertoire. After generating the initial population, we evaluate all neural networks of the population in a set of randomly generated environments and compute the median behavior characterization and the mean quality score of each neural network. The randomly generated environments all share the same size, shape, and ground color but can have a floor patch of a random color (white, gray, or black) at a random position chosen between three possible ones, up to one obstacle box at the center of the arena, and up to one light source. We evaluate each neural network of the population on 10 random environments for 100 s.

The behavior characterization is a vector that represents the behavior of the robots in the swarm when evaluated in different environments [50]. To characterize the behavior of the swarm, we compute for each individual robot the 7 following features: the linear and angular speed; the distance to walls/obstacles, to other robots, and to the closest robot; the ambient light and the ground color perceived. The behavior characterization vector is composed of 14 real values: the mean and the standard deviation for each of the features described above. These values are computed based on the evaluations of the swarm on the 10 random environments.

The quality score represents the number of collisions that occurred during an evaluation of a neural network and is computed using the following function: $Q = 1 - C/(T \cdot N)$, where C is the number of collisions, T is the duration of the evaluation and N is the size of the swarm.

After these steps, we update the repertoire with the newly evaluated neural networks based on their novelty score and local competition score. The novelty score is the mean distance of the current neural network to its k nearest neighbors. The local competition score is the number of neural networks in the k nearest neighbors that are outperformed by the current neural network. In our case, we considered $k = 25$. The given neural network is then added to the repertoire if its nearest neighbor is sufficiently different from it (the distance between is greater than parameter l). The given neural network may also replace its nearest neighbors in the repertoire, if it is not sufficiently different (the distance between is less than l) but its quality score is strictly greater and the second nearest neighbor is sufficiently different from it. The following generation of neural networks is created by crossover and mutation on the genomes of the current generation. The evolutionary process is a modified version of the NEAT [51] algorithm to allow Pareto-based bi-objective optimization, where the two objectives are the novelty score and the local competition score.

The repertoire is thus built in a task-agnostic way and has to be built only once to be subsequently used in a great variety of swarm robotics missions.

Algorithm 1 Repertoire evolution with novelty search with local competition. In our case, $l = 1.5$ is the repertoire distance threshold, $s = 50$ is the repertoire growth per generation, $S = 2000$ is the archive capacity, and $k = 25$ is the nearest neighbors count. The archive A is filled during the execution of the algorithm with randomly selected neural networks from the population to encourage uniform behavior exploration [43]. It is used to compute the novelty and local competition scores.

$A \leftarrow \emptyset$, $R \leftarrow \emptyset$
$P \leftarrow$ `RandomInitialPopulation()`
for $g \in$ generation
  for $i \in$ P
    for $e \in$ E
      $q_e, b_e \leftarrow$ `Evaluate`(i, e)
    $B(i) \leftarrow$ `GeometricMedian`$(\{b_e : e \in E\})$
    $Q(i) \leftarrow \sum_{e \in E} \frac{1}{|E|} q_e$
  for $i \in$ P
    $\chi \leftarrow$ `NearestNeighbors`$(i, A \cup P, k)$
    $N(i) \leftarrow \sum_{x \in \chi} \frac{1}{|\chi|}$ `dist`$(B(i), B(x))$
    $LC(i) \leftarrow \sum_{x \in \chi} [Q(i) > Q(x)]$
    $\chi_1, \chi_2 \leftarrow$ `NearestNeighbors`$(i, R, 2)$
    if $|R| = 0$ **or** $|R| > 0$ **and** `dist`$(B(i), B(\chi_1)) > l$
      $R \leftarrow R \cup \{i\}$
    else if $|R| > 1$ **and** $Q(i) > Q(\chi_1)$ **and** `dist`$(B(i), B(\chi_2)) > l \cdot 0.1$
      $R \leftarrow (R \setminus \{\chi_1\}) \cup \{i\}$
  if $|A| > S - s$
    $A \leftarrow A \setminus$ `SelectRandom`$(A, |A| + s - S)$
  $A \leftarrow A \cup$ `SelectRandom`(P, s)
  $P \leftarrow$ `Breed`(P) based on $N(i)$ and $LC(i)$
return R

3.2. Generation of rules for condition modules

Condition modules are generated automatically via a set of rules that operate on the reference model of the robot. More precisely,

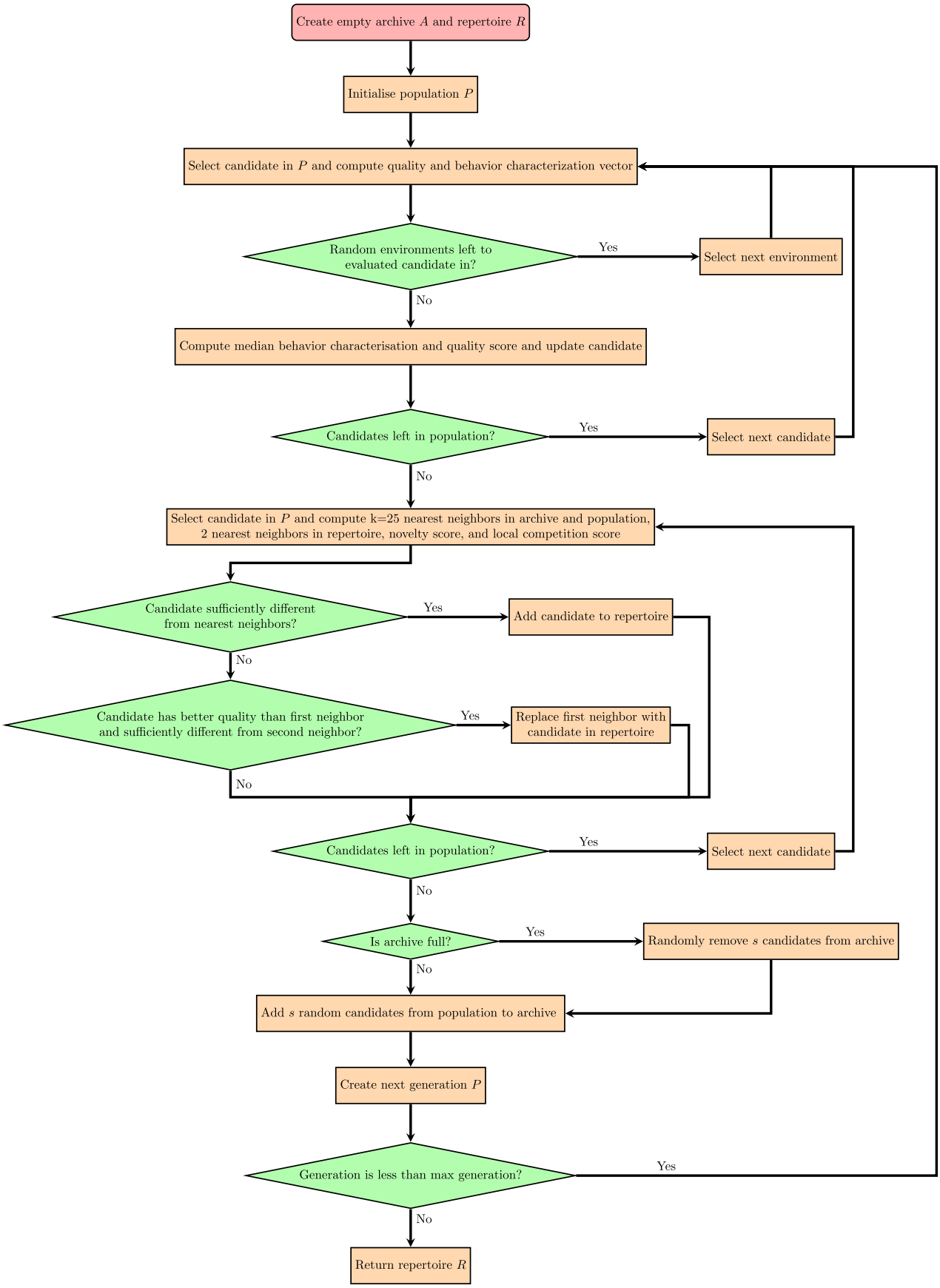


Fig. 1. Flowchart for the repertoire evolution. Creation of the repertoire using novelty search with local competition. Flowchart visualization of Algorithm 1.

Table 1
Condition modules: Five condition modules were generated based on the five inputs of reference model RM1.1 presented in Table 2: proximity, light, ground, number of neighboring robots perceived, and attraction vector.

Input	Input class	parameters
proximity sensor	continuous	$p \in [0,1]$: probability $\theta \in [0,8]$: threshold $d \in \{0,1\}$: trigger above or below θ
light sensor	continuous	$p \in [0,1]$: probability $\theta \in [0,8]$: threshold $b \in \{0,1\}$: trigger above or below θ
ground sensor	categorical	$p \in [0,1]$: probability $C \in \{\text{black, gray, white}\}$
number of neighboring robots perceived	continuous	$p \in [0,1]$: probability $\theta \in [0,20]$: threshold $d \in \{0,1\}$: trigger above or below θ
attraction vector	vector	$p \in [0,1]$: probability $Q \in \{[-\pi, \frac{-\pi}{2}], [\frac{-\pi}{2}, 0], [0, \frac{\pi}{2}], [\frac{\pi}{2}, \pi]\}$: quadrant

the input variables of the reference model, which represent sensory information, are used to define triggers for the condition modules. We separate input variables into three classes: categorical, continuous, or vector inputs. For each input variable, depending on the class to which it belongs, a specific ruleset is automatically applied to generate one transition module.

A condition module based on a categorical input triggers a transition with probability p when the input is detected to be C with $C \in \{c_1, c_2, \ldots, c_n\}$. If multiple sensors form the input, all the values must be detected to be C.

A condition module based on a continuous input triggers a transition with probability p when the input is detected to be above ($d = 1$) or below ($d = 0$) the threshold θ with $\theta \in \mathbb{R}$. If multiple sensors form the input, the sum of the values need to be above or below the threshold θ.

A condition module based on a vector input triggers a transition with probability p when the input is detected with angle in quadrant Q with $Q \in \left\{[-\pi, \frac{-\pi}{2}], [\frac{-\pi}{2}, 0], [0, \frac{\pi}{2}], [\frac{\pi}{2}, \pi]\right\}$. In fact, for vector inputs, the magnitude of the vector is disregarded, and the angle is used as a categorical input with four distinct values representing four equal quadrants of $\pi/2$ rad.

The parameters p, C, d, θ, and Q are fine-tuned by the optimization algorithm, when they apply. The details specific to the robotic platform in use, including the classes of the inputs considered here are given in Section 4.2.2 and Table 2.

The choices made to define the different rules that were applied to generate the transitions do have implications on the whole design of the system. These rules have their limitations, but it is our contention that, simpler, easier rules to apply, allow for more straightforward implementation. More fine-grained rules could lead to better design in transitions but we do not expect they would make any significant difference to the overall conclusions of this study.

The specific condition modules generated for the e-puck robot for `Nata` using reference model RM1.1 is presented in Table 1.

3.3. Assembly of condition and behavior modules

As customary in most methods belonging to the AutoMoDe family, `Nata` generates control software by assembling behavior and condition modules into probabilistic finite-state machines. `Nata` uses the optimization algorithm irace [46] to generate probabilistic finite-state machines that are limited in size: they can comprise up to 4 states and up to 4 outgoing transitions per state; this constraint was set in other AutoMoDe methods such as `Vanilla`, and `Chocolate`. For each state of the probabilistic finite-state machine, irace selects one of the neural networks of the repertoire. For each transition, irace configures one condition module.

4. Experiments and results

4.1. Experimental setup

The methods comprised in the study include: `Nata`, the novel method introduced in this article; `Arlequin` [27], the first method of the AutoMoDe family that uses neural networks as modules of a finite-state machine; `EvoStick` [19], a standard neuroevolutionary method; and `Chocolate` [19], the most studied method of the AutoMoDe family that uses hand-crafted modules—more information about methods is available in Section 4.2.1. We tested the ability of these methods to produce control software in a fully-automatic way, to solve three missions. The missions considered – which are described in Section 4.3 – are rather typical swarm robotics missions. Their complexity aligns with previous work in the fully automatic design of robot swarms [16]. The following sections present details about the material and methods used in this study.

4.2. Materials and methods

4.2.1. Automatic design methods

`EvoStick` is a neuroevolutionary method that generates fully-connected neural networks with no hidden layers: 25 input nodes (8 dedicated to the readings of the proximity sensors, 8 to those of the light sensors, 5 to those of the range-and-bearing, and 1 as bias) are directly connected to the two output nodes that dictate the speed of the wheels. The evolutionary algorithm that optimizes the synaptic weights in the range $[5, -5]$ has a population size of 100 individuals, which are evaluated 10 times per generation. New generations are populated by the 20 best individuals unchanged, and 80 are produced via mutations of these 20 best individuals. We refer the reader to the original paper for more details [52].

`Chocolate` has at its disposal a set of 6 low-level behaviors (i.e., exploration, stop, attraction-to-light, attraction-from-light, attraction-to-neighbors, repulsion-from-neighbors) and 6 conditions (i.e., black-floor, white-floor, gray-floor, neighbor-count, inverted-neighbor-count, fixed-probability). All the modules were created once and for all by hand; some of the low-level behaviors (i.e., exploration, attraction-to-neighbors, repulsion-from-neighbors) have parameters that slightly adjust their functioning; all conditions have parameters that adjust their triggering frequencies. `Chocolate` uses the optimization algorithm irace [46] to select, fine-tune, and combine these modules into probabilistic finite-state machines of up to 4 states (i.e., low-level behaviors) and up to 4 outgoing edges (i.e., conditions) per state. We refer the reader to the original paper for more details [20].

`Arlequin` is in many aspects similar to `Chocolate`: it uses the same optimization algorithm to produce probabilistic finite-state machines with the same properties. The only difference lies in the nature of the 6 low-level behaviors that are at its disposal: the hand-crafted modules of `Chocolate` are replaced by 6 neural networks generated by `EvoStick`. These 6 neural networks are generated once and for all, and are subsequently used without any further modification. We refer the reader to the original paper for more details [27]. The software used in the study, and in particular all implementations of design methods, is available online as supplementary material [53].

In Fig. 2, we present functional diagrams depicting the different automatic design methods compared in this study.

4.2.2. Protocol

Each design method is run 10 times, producing 10 instances of control software for each mission. These instances of control software are then tested once in simulation and once on a swarm of 20 e-puck robots.

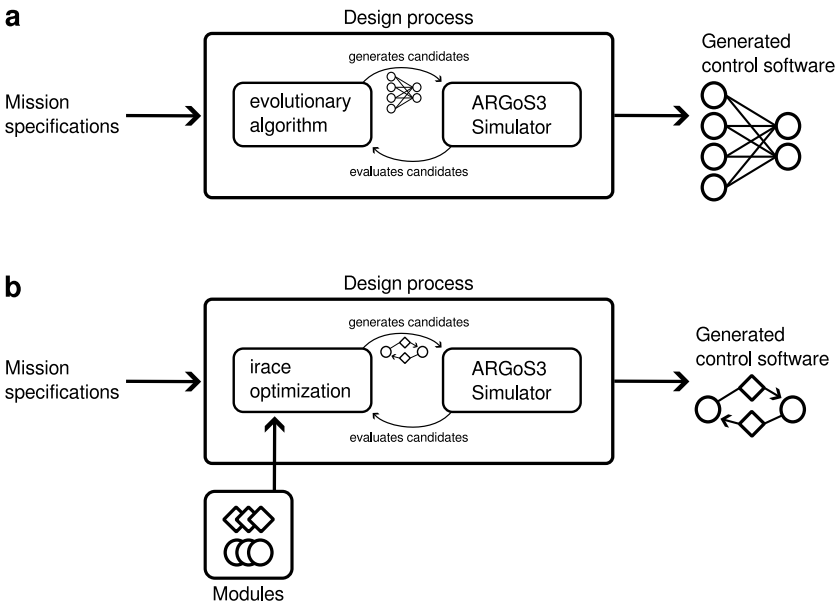


Fig. 2. Functional diagrams. a, Functional diagram of Evostick. Evostick generates control software using neuroevolution: an evolutionary algorithm optimizes the weights of a fully-connected neural network. The population comprises 100 individuals that are evolved via elitism and mutation. At the end of the process, the best performing neural network is selected. **b**, Functional diagram of Chocolate, Arlequin, and Nata. The three methods belong to the AutoMoDe family: they assemble a priori defined modules into probabilistic finite-state using the irace optimization algorithm. For Chocolate, the modules are a set of 6 hand-crafted behaviors, and 6 hand-crafted conditions. For Arlequin, the modules are a set of 6 automatically generated behaviors in the form of neural networks, and 6 hand-crafted conditions. For Nata, the modules are a repertoire of 670 automatically generated (via novelty search) behaviors in the form of neural networks, and 5 automatically generated conditions.

The e-puck robot. The e-puck is a small differential-drive robot equipped with various sensors and actuators. It measures 55 mm in height and 70 mm in diameter. The robot is equipped with light sensors capable of measuring ambient light, proximity sensors capable of detecting nearby obstacles, and ground sensors capable of detecting the gray-scale color of the floor under the robot. The robot is also equipped with two extension boards: the Overo Gumstix, a Linux single-board computer that increases the computing power of the robot, and the range-and-bearing board, allowing the robot to detect its neighboring peers by sensing their relative angle and distance.

Reference model. The specific configuration of the e-puck robot used in this study is formally described by reference model RM1.1 given in Table 2 — see [54] for

details. All methods in this study use the variables defined in this reference model to generate control software.

The argos simulator. All simulations in this study were performed using ARGoS3 [55] (version 48), a simulator specifically conceived for running large-scale swarm robotics experiments. We used the ARGoS-Epuck library [56] (version 48) to simulate the version of the e-puck robot equipped with the two extension boards (mentioned in Section 4.2.2).

The generated control software was directly cross-compiled so that the software running in simulation could be directly ported to the robot without undergoing any modifications.

Table 2
Reference model RM1.1: capabilities of the e-puck robotic platform used in this study. For inputs, the class – that defines the derived condition modules – is indicated in the last column. The attraction vector V points to the aggregate position of the n neighboring peers that are within the detection range of approximately 0.70 m. It is computed as $V = \sum_{m=1}^{n}(\frac{1}{1+r_m}, \angle b_m)$, where r_m and $\angle b_m$ are the range and the bearing of neighbor m, respectively. If $n = 0$, then $V = (1, \angle 0)$.

Input	Value	Description	Input class
$prox_{i\in\{1,\dots,8\}}$	[0,1]	reading of proximity sensor i	continuous
$light_{i\in\{1,\dots,8\}}$	[0,1]	reading of light sensor i	continuous
$gnd_{j\in\{1,2,3\}}$	{black, gray, white}	reading of ground sensor j	categorical
n	[0,20]	number of neighboring robots perceived	continuous
V	$([0.5, 20], [0, 2\pi])$	attraction vector	vector

Output	Value	Description
$v_{k\in\{l,r\}}$	$[-0.12, 0.12]\,\mathrm{m\,s^{-1}}$	target linear wheel velocity

Period of the control cycle: 100 ms.

4.3. Missions under analysis

The three missions (Fig. 3) considered are classical swarm robotics missions; here, we present the characteristics and objective functions of each of them.

4.3.1. XOR-aggregation

The robots must aggregate on one of the two black areas in the arena. The performance of the swarm is computed based on the following objective function:

$$F_a = \sum_{t=1}^{T}\sum_{i=1}^{N} I_i(t);$$

$$I_i(t) = \begin{cases} 1, & \text{if robot } i \text{ is in the area with a majority of robots;} \\ 0, & \text{otherwise.} \end{cases}$$

The duration of the experimental run is $T = 180$ s, and $N = 20$ is the size of the swarm.

4.3.2. Foraging

The robots must find one of the two black areas, which represent food sources and return to the white area, which represents the nest. The performance of the swarm is computed based on the following objective function:

$$F_f = K;$$

where K is the total number of round trips performed. The duration of an experimental run is $T = 180$ s, and the swarm size is $N = 20$.

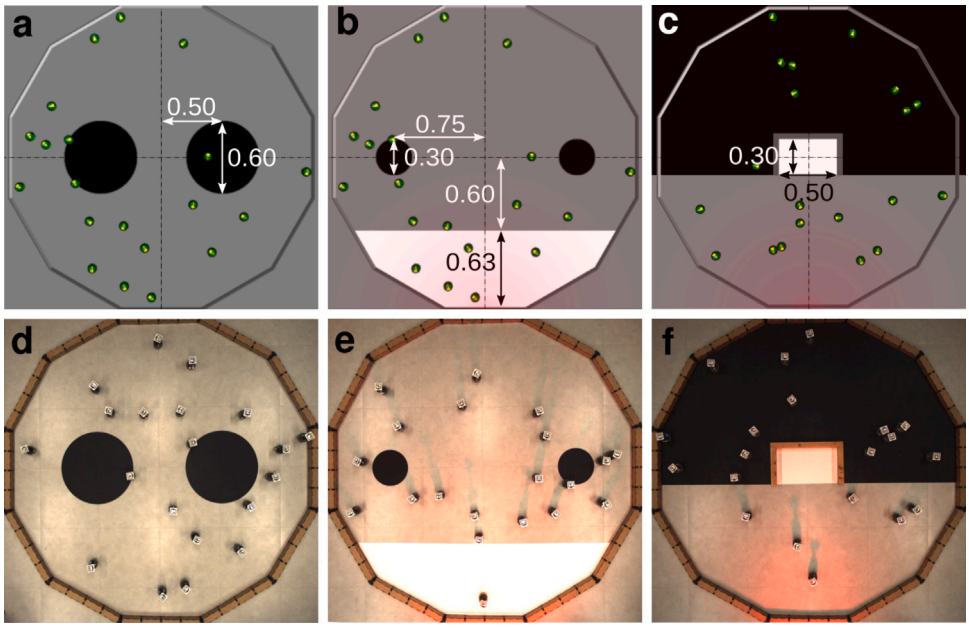


Fig. 3. Arenas for the three missions. a, d, XOR-Aggregation. **b, e**, Foraging. **c, f**, Shelter w/Cues. **a–c** simulation; **d–f**, real robots. The 20 robots operate in a dodecagonal arena of 4.91 m^2, the red glow in **b**, **c**, **e**, and **f** indicates the presence of a light source at the bottom side of the arena. Dimensions (in meters) of the elements present in the arenas are given in **a**, **b**, and **c**.

4.3.3. Shelter with cues from the environment

The robots must aggregate in a shelter: a white area open on one side and surrounded by walls on the three other ones. A light source is positioned outside of the arena and is directed towards the open side of the shelter. The floor of the arena on the side of the walls of the shelter is black. The performance of the swarm is computed based on the following objective function:

$$F_s = \sum_{t=1}^{T} \sum_{i=1}^{N} I_i(t);$$

$$I_i(t) = \begin{cases} 1, & \text{if robot } i \text{ is in the shelter;} \\ 0, & \text{otherwise.} \end{cases}$$

The duration of the experimental run is $T = 180\,s$, and $N = 20$ is the size of the swarm.

4.4. Statistical analysis

We present the performance of the different methods using notched box-and-whiskers plots. In these plots, the thick black line in the middle of the box represents the median of the performance distribution. The upper and lower edges of the boxes represent the upper and lower quartiles of the distribution. The upper and lower horizontal ticks represent the upper and lower observations in 1.5 times the interquartile range; dots represent outliers.

To aggregate the performance of the methods across missions we used a min–max normalization: for each mission, the performance is normalized based on the maximum and minimum performance observed in reality across all methods. The normalized performance in reality ranges thus between 0 and 1, while the one in simulation can exceed 1. Aggregated performance distributions are then represented as notched box-and-whiskers plots.

We also report the aggregated performance drop experienced by each method across the three missions, together with the 95% confidence interval. The performance drop is computed as the difference between the performance assessed in simulation and the one observed on the physical robots, which we normalize with the performance assessed in simulation. The performance drop is to be minimized: a value of 1 means that the performance in reality is null, and a value of 0 that the performance is equal in simulation and in reality. Note that negative performance drop are possible if control software performs better in reality than in simulation.

We eventually performed a Friedman test [57], which aggregates all results obtained in real robot experiments by ranking the performance of all methods across missions. We report the average rank of each method and its 95% confidence interval. We have a statistically significant difference (with at least 95% confidence) between two methods when their confidence intervals do not overlap.

4.5. Results

Fig. 4 shows the performance obtained by the four methods on the three missions both in simulation and on physical robots. In simulation, for the Foraging mission, `EvoStick` outperforms the other three modular methods, which perform similarly (the notches representing the 95% confidence interval on the boxplots overlap). For XOR-Aggregation, all methods perform similarly, whereas for Shelter w/Cues differences between methods are all significant: `EvoStick` performed best, followed by `Arlequin`, then `Chocolate` and finally `Nata`. Overall, when aggregating all results (see Fig. 5a), `EvoStick` produced control software that performed best in simulation. Among the modular methods, `Arlequin` and `Chocolate` produced control software that perform similarly, and the one produced by `Nata` is outperformed by the one of `Chocolate`.

In reality, things are different. In fact, we observed several rank inversions between the methods. For Foraging, `Nata` and `Chocolate` perform similarly, and outperform both `Arlequin` and `EvoStick`. For XOR-Aggregation, `Chocolate` is the best performing method. `Nata` and `Arlequin` perform similarly and slightly worse than `Chocolate`, but considerably better than `EvoStick`. For Shelter w/Cues, the median performance of `Chocolate` is twice better than those of the other three methods, which perform similarly. Overall, when executed on a swarm of e-puck robots, the control software produced by `Chocolate` is the best performing one, followed by the one produced by `Nata`, then `Arlequin` and `EvoStick` (see Fig. 5a). According to the Friedman rank sum test, the differences between these four methods are all significant with a confidence level of at least 95% (see Fig. 5c).

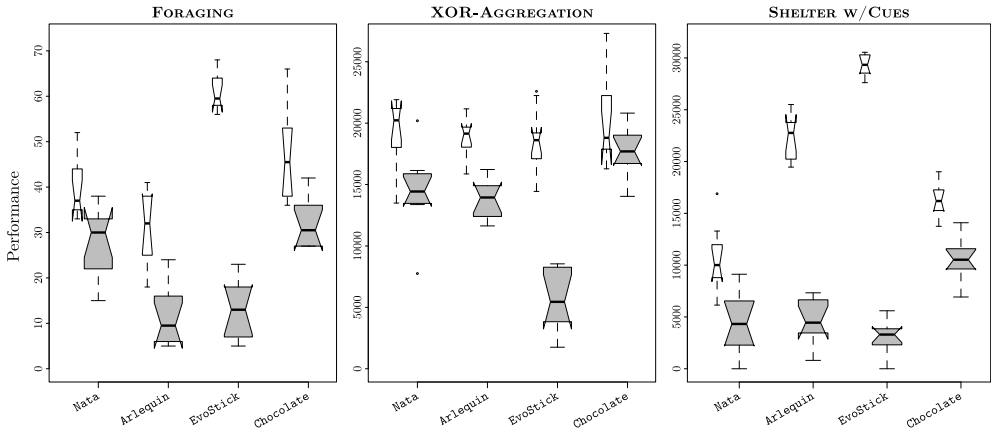


Fig. 4. Results per mission. Performance obtained in simulation (narrow white boxes) and in reality (thick gray boxes) in all three missions (the higher the better).

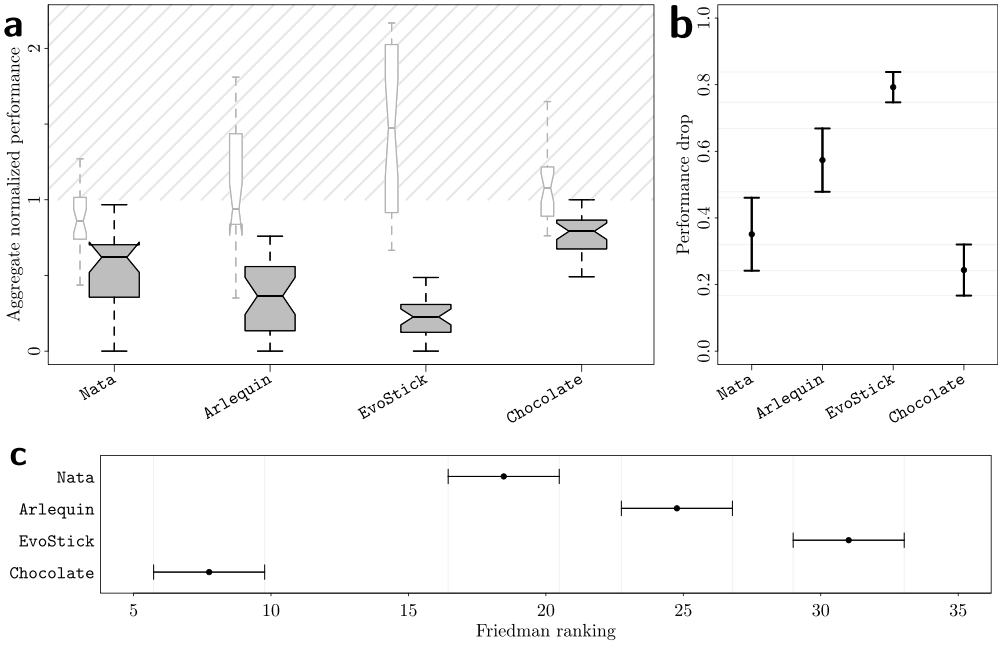


Fig. 5. Aggregated Results. a. Aggregated normalized results. Aggregated performance in simulation (narrow white boxes) and in reality (thick gray boxes) across the three missions (the higher the better). Prior to aggregation, results are normalized between the lowest and the highest performance observed in reality by any method in a given mission. As a result, aggregated reality results range from 0 to 1, whereas aggregated simulation results might exceed 1 (shadowed area). This is observed because, in many cases, the performance in simulation exceeded the one in reality. **b.** Normalized performance drop experienced by the methods, aggregated across all missions, and 95% confidence interval. For a given instance of control software, the performance drop is computed as the difference between the performance assessed in simulation and the one observed in reality, and is normalized with the one assessed in simulation. **c.** Friedman test results. Friedman test on the aggregate results in reality of the three missions, the plot shows the average rank and the 95% confidence interval (the lower the better).

Fig. 5b shows the performance drop from simulation to reality, which gives an estimation of how much the methods are affected by the reality gap. `EvoStick` suffers from the biggest overall performance drop and shows very poor performance in reality in all three missions, whereas `Chocolate` suffered from the lowest overall performance drop and is the method that is the best at crossing the reality gap. `Nata` suffered from a significantly lower performance drop than `EvoStick` and `Arlequin`, but bigger than the one of `Chocolate`, although not significantly.

Both in simulation and with physical robots, the results that we obtained using already existing methods, namely `Chocolate`, `EvoStick` and `Arlequin`, are consistent with previous studies (see [10,16,27,58]).

It is also worth mentioning that, as it can be observed in the supplementary videos [53] and as shown by the results in simulation, the different missions considered in the study can be accomplished by the design methods, and the allocated computing resources are sufficient to reach a satisfactory level of performance. All results obtained in simulation and real-robot experiments are available as supplementary data [53].

5. Discussion and conclusions

In this paper, we addressed the drawbacks of both the neuroevolutionary robotics approach and the modular one: neuroevolutionary methods suffer from important performance drops due to the reality gap, whereas modular methods require human designers to meticulously implement behavior and condition modules. We presented `Nata`, a modular method that produces control software by automatically selecting and combining task-agnostic behaviors that were themselves automatically generated, via novelty search. `Nata` produced control software that is more robust to the reality gap than the

one produced by the classical neuroevolutionary approach, whereas the process adopted to generate the modules to be combined requires less human expertise than the ones previously adopted by other modular methods. The results presented are to be considered as a proof of concept; in the future, we will work on further assessing the capabilities of Nata and extending them. In particular, we will focus on identifying the class of mission that Nata can tackle and on further improving its robustness to the reality gap.

The performance of Nata is satisfactory as it exceeds the one of EvoStick and Arlequin. Admittedly, Nata did not reach the performance level of Chocolate. Yet, it is worth noting that in contrast to Chocolate– which was conceived by a human expert who identified and implemented the relevant low-level behaviors and transition conditions – Nata was defined automatically: low-level behaviors and transition conditions were generated automatically. To the best of our knowledge, Nata is therefore the first modular automatic design method that has been generated automatically. In this sense, in Nata, the principles of automatic design have been applied at a meta level: the automatic design of collective behaviors for robot swarms is achieved by a method that was itself generated automatically.

One possible drawback of modular methods and specifically to automatically generating the behavioral modules to be combined into probabilistic finite-state machines – or any other control architecture – is the risk of introducing a source of overfitting within the method, and nullify the benefits of modularity. In modular approaches overfitting can occur at two levels: during the implementation of the modules, and during their combination and their (possible) fine-tuning. The overfitting that occurs during the implementation of the modules is mission independent, and is caused by a mismatch between how the robots behave in simulation and in reality when executing a given module. The overfitting that occurs during the combination and fine-tuning of the modules is mission specific, and could be caused by unforeseen interactions between the robots and/or between the robots and the environment. The two levels contribute to the effects of the reality gap experienced by an instance of control software generated by a modular design method. It is hard to identify which level contributes the most and, therefore, on which level one should work to reduce the overall performance drop.

The combination process (i.e., optimization algorithm, control architecture, and constraints) is identical in Nata, Arlequin, and Chocolate. It is therefore reasonable to assume that the overfitting that occurs at the level of the implementation of the modules is the most important factor that explains the discrepancies we observed between the three methods.

As Nata produced control software that is more robust to the reality gap than the one produced by Arlequin, it appears that generating behaviors in the form of neural networks using novelty search with the aim of minimizing collisions is more appropriate than using direct evolution. The modules of Chocolate were crafted by hand and evaluations on physical robots during their implementation ensured that they crossed the reality gap satisfactorily. Evaluating the modules of the repertoire of Nata on robots with the objective of potentially pruning the repertoire to only keep modules that cross the reality gap satisfactorily would be extremely costly and time consuming. Indeed, a repertoire typically contains several hundreds of behavioral modules and each module should be evaluated in multiple environments so as to correctly assess its behavior characterization vector. We foresee that the pruning of the repertoire could be done by leveraging the concept of pseudo-reality [59]. A pseudo-reality is a simulation model, different from the one used during the design, that is used to mimic the reality gap experienced when going from simulation to reality. Evaluations of behavioral modules in pseudo-reality, which are fast and inexpensive, could give an idea on their robustness to the reality gap and could subsequently be used to filter and select modules with the best potential to perform seamlessly in reality.

Ethical approval

Not applicable

Funding

This 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), from Belgium's Wallonia-Brussels Federation through the ARC Advanced Project GbO (Guaranteed by Optimization), and from the Belgian Fonds de la Recherche Scientifique–FNRS (SwarmSim and SwarmUp). M. Birattari acknowledges support from the Belgian Fonds de la Recherche Scientifique–FNRS, of which he is a Research Director.

CRediT authorship contribution statement

Ken Hasselmann: Conceptualization, Methodology, Software, Formal analysis, Investigation, Writing – original draft, Writing – review & editing, Visualization. **Antoine Ligot:** Conceptualization, Methodology, Formal analysis, Investigation, Writing – review & editing, Visualization. **Mauro Birattari:** Conceptualization, Methodology, Writing – review & editing, Supervision, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data and software is available on the supplementary website associated to the paper.

References

[1] M. Dorigo, M. Birattari, Swarm intelligence, Scholarpedia 2 (9) (2007) 1462, http://dx.doi.org/10.4249/scholarpedia.1462.
[2] M. Brambilla, E. Ferrante, M. Birattari, M. Dorigo, Swarm robotics: a review from the swarm engineering perspective, Swarm Intell. 7 (1) (2013) 1–41, http://dx.doi.org/10.1007/s11721-012-0075-2.
[3] H. Hamann, Swarm Robotics: A Formal Approach, Springer, Cham, Switzerland, 2018, http://dx.doi.org/10.1007/978-3-319-74528-2.
[4] M. Dorigo, G. Theraulaz, V. Trianni, Reflections on the future of swarm robotics, Science Robotics 5 (2020) eabe4385, http://dx.doi.org/10.1126/scirobotics.abe4385.
[5] M. Dorigo, G. Theraulaz, V. Trianni, Swarm robotics: past, present, and future [point of view], Proc. IEEE 109 (7) (2021) 1152–1165, http://dx.doi.org/10.1109/JPROC.2021.3072740.
[6] V. Trianni, Evolutionary Swarm Robotics, Springer, Berlin, Germany, 2008, http://dx.doi.org/10.1007/978-3-540-77612-3.
[7] M.K. Heinrich, M. Wahby, M. Dorigo, H. Hamann, Swarm robotics, in: A. Cangelosi, M. Asada (Eds.), Cognitive Robotics, MIT Press, Cambridge, MA, USA, 2022, http://dx.doi.org/10.7551/mitpress/13780.003.0009.
[8] J.C. Bongard, Evolutionary robotics, Commun. ACM 56 (8) (2013) 74–83.
[9] S. Doncieux, J.-B. Mouret, Beyond black-box optimization: a review of selective pressures for evolutionary robotics, Evolut. Intell. 7 (2) (2014) 71–93, http://dx.doi.org/10.1007/s12065-014-0110-x.
[10] G. Francesca, M. Birattari, Automatic design of robot swarms: achievements and challenges, Front. Robot. AI 3 (29) (2016) 1–9, http://dx.doi.org/10.3389/frobt.2016.00029.
[11] F. Silva, M. Duarte, L. Correia, S.M. Oliveira, A.L. Christensen, Open issues in evolutionary robotics, Evolut. Comput. 24 (2) (2016) 205–236, http://dx.doi.org/10.1162/EVCO_a_00172.
[12] R.A. Brooks, Intelligence without representation, Artificial Intelligence 47 (1991) 139–159, http://dx.doi.org/10.1007/s00422-006-0080-x.
[13] N. Jakobi, P. Husbands, I. Harvey, Noise and the reality gap: the use of simulation in evolutionary robotics, in: F. Morán, A. Moreno, J.J. Merelo, P. Chacón (Eds.), Advances in Artificial Life: Third European Conference on Artificial Life, in: Lecture Notes in Artificial Intelligence, vol. 929, Springer, Berlin, Germany, 1995, pp. 704–720, http://dx.doi.org/10.1007/3-540-59496-5_337.

[14] 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, T. Stützle, Automatic off-line design of robot swarms: a manifesto, Front. Robot. AI 6 (2019) 59, http://dx.doi.org/10.3389/frobt.2019.00059.

[15] M. Birattari, A. Ligot, K. Hasselmann, Disentangling automatic and semi-automatic approaches to the optimization-based design of control software for robot swarms, Nat. Mach. Intell. 2 (9) (2020) 494–499, http://dx.doi.org/10.1038/s42256-020-0215-0.

[16] K. Hasselmann, A. Ligot, J. Ruddick, M. Birattari, Empirical assessment and comparison of neuro-evolutionary methods for the automatic off-line design of robot swarms, Nature Commun. 12 (2021) 4345, http://dx.doi.org/10.1038/s41467-021-24642-3.

[17] M. Duarte, S.M. Oliveira, A.L. Christensen, Evolution of hierarchical controllers for multirobot systems, in: H. Sayama, J. Rieffel, S. Risi, R. Doursat, H. Lipson (Eds.), ALIFE 14: The Fourteenth International Conference on the Synthesis and Simulation of Living Systems, MIT Press, Cambridge, MA, USA, 2014, pp. 657–664, http://dx.doi.org/10.7551/978-0-262-32621-6-ch105.

[18] M. Duarte, S.M. Oliveira, A.L. Christensen, Hybrid control for large swarms of aquatic drones, in: H. Sayama, J. Rieffel, S. Risi, R. Doursat, H. Lipson (Eds.), ALIFE 14: The Fourteenth International Conference on the Synthesis and Simulation of Living Systems, MIT Press, Cambridge, MA, USA, 2014, pp. 785–792, http://dx.doi.org/10.7551/978-0-262-32621-6-ch105.

[19] G. Francesca, M. Brambilla, A. Brutschy, V. Trianni, M. Birattari, AutoMoDe: a novel approach to the automatic design of control software for robot swarms, Swarm Intell. 8 (2) (2014) 89–112, http://dx.doi.org/10.1007/s11721-014-0092-4.

[20] G. Francesca, M. Brambilla, A. Brutschy, L. Garattoni, R. Miletitch, G. Podevijn, A. Reina, T. Soleymani, M. Salvaro, C. Pinciroli, F. Mascia, V. Trianni, M. Birattari, AutoMoDe-Chocolate: automatic design of control software for robot swarms, Swarm Intell. 9 (2–3) (2015) 125–152, http://dx.doi.org/10.1007/s11721-015-0107-9.

[21] K. Hasselmann, M. Birattari, Modular automatic design of collective behaviors for robots endowed with local communication capabilities, PeerJ Comput. Sci. 6 (2020) e291, http://dx.doi.org/10.7717/peerj-cs.291.

[22] J. Kuckling, A. Ligot, D. Bozhinoski, M. Birattari, Behavior trees as a control architecture in the automatic modular design of robot swarms, in: M. Dorigo, M. Birattari, C. Blum, A.L. Christensen, A. Reina, V. Trianni (Eds.), Swarm Intelligence: 11th International Conference, ANTS 2018, in: Lecture Notes in Computer Science, vol. 11172, Springer, Cham, Switzerland, 2018, pp. 30–43, http://dx.doi.org/10.1007/978-3-030-00533-7_3.

[23] J. Kuckling, K. Ubeda Arriaza, M. Birattari, Simulated annealing as an optimization algorithm in the automatic modular design of robot swarms, in: K. Beuls, B. Bogaerts, G. Bontempi, P. Geurts, N. Harley, B. Lebichot, T. Lenaerts, G. Louppe, P. Van Eecke (Eds.), Proceedings of the Reference AI & ML Conference for Belgium, Netherlands & Luxemburg, BNAIC/BENELEARN 2019, vol. 2491, CEUR Workshop Proceedings, Aachen, Germany, 2019.

[24] M. Salman, A. Ligot, M. Birattari, Concurrent design of control software and configuration of hardware for robot swarms under economic constraints, PeerJ Comput. Sci. 5 (2019) e221, http://dx.doi.org/10.7717/peerj-cs.221.

[25] G. Spaey, M. Kegeleirs, D. Garzón Ramos, M. Birattari, Comparison of different exploration schemes in the automatic modular design of robot swarms, in: K. Beuls, B. Bogaerts, G. Bontempi, P. Geurts, N. Harley, B. Lebichot, T. Lenaerts, G. Louppe, P. Van Eecke (Eds.), Proceedings of the Reference AI & ML Conference for Belgium, Netherlands & Luxemburg, BNAIC/BENELEARN 2019, vol. 2491, CEUR Workshop Proceedings, Aachen, Germany, 2019.

[26] D. Garzón Ramos, M. Birattari, Automatic design of collective behaviors for robots that can display and perceive colors, Appl. Sci. 10 (13) (2020) 4654, http://dx.doi.org/10.3390/app10134654.

[27] A. Ligot, K. Hasselmann, M. Birattari, AutoMoDe-Arlequin: neural networks as behavioral modules for the automatic design of probabilistic finite state machines, in: M. Dorigo, T. Stützle, M.a.J. Blesa, C. Blum, H. Hamann, M.K. Heinrich, V. Strobel (Eds.), Swarm Intelligence: 12th International Conference, ANTS 2020, in: Lecture Notes in Computer Science, vol. 12421, Springer, Cham, Switzerland, 2020, pp. 109–122, http://dx.doi.org/10.1007/978-3-030-60376-2_21.

[28] J. Kuckling, V. van Pelt, M. Birattari, Automatic modular design of behavior trees for robot swarms with communication capabilities, in: P.A. Castillo, J.L. Jiménez Laredo (Eds.), Applications of Evolutionary Computation: 24th International Conference, EvoApplications 2021, in: Lecture Notes in Computer Science, vol. 12694, Springer, Cham, Switzerland, 2021, pp. 130–145, http://dx.doi.org/10.1007/978-3-030-72699-7_9.

[29] M. Birattari, A. Ligot, G. Francesca, AutoMoDe: a modular approach to the automatic off-line design and fine-tuning of control software for robot swarms, in: N. Pillay, R. Qu (Eds.), Automated Design of Machine Learning and Search Algorithms, in: Natural Computing Series, Springer, Cham, Switzerland, 2021, pp. 73–90, http://dx.doi.org/10.1007/978-3-030-72069-8_5.

[30] D. Floreano, J. Urzelai, Evolutionary robots with on-line self-organization and behavioral fitness, Neural Netw. 13 (2000) 431–443, http://dx.doi.org/10.1016/S0893-6080(00)00032-0.

[31] J.K. Pugh, L.B. Soros, K.O. Stanley, Quality Diversity: a new frontier for evolutionary computation, Front. Robot. AI 3 (2016) 40, http://dx.doi.org/10.3389/frobt.2016.00040.

[32] J. Gomes, A.L. Christensen, Task-agnostic evolution of diverse repertoires of swarm behaviours, in: M. Dorigo, M. Birattari, C. Blum, A.L. Christensen, A. Reina, V. Trianni (Eds.), Swarm Intelligence: 11th International Conference, ANTS 2018, in: Lecture Notes in Computer Science, vol. 11172, Springer, Cham, Switzerland, 2018, pp. 225–238, http://dx.doi.org/10.1007/978-3-030-00533-7_18.

[33] J. Lehman, K.O. Stanley, Evolving a diversity of virtual creatures through novelty search and local competition, in: N. Krasnogor (Ed.), GECCO'11: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, ACM, New York, NY, USA, 2011, pp. 211–218, http://dx.doi.org/10.1145/2001576.2001606.

[34] S. Geman, E. Bienenstock, R. Doursat, Neural networks and the bias/variance dilemma, Neural Comput. 4 (1) (1992) 1–58, http://dx.doi.org/10.1162/neco.1992.4.1.1.

[35] D. Wolpert, On bias plus variance, Neural Comput. 9 (1997) 1211–1243, http://dx.doi.org/10.1162/neco.1997.9.6.1211.

[36] J.-B. Mouret, J. Clune, Illuminating search spaces by mapping elites, 2015, http://arxiv.org/abs/1504.04909.

[37] M. Duarte, J. Gomes, S.M. Oliveira, A.L. Christensen, Evolution of repertoire-based control for robots with complex locomotor systems, IEEE Trans. Evol. Comput. 22 (2) (2018) 314–328, http://dx.doi.org/10.1109/TEVC.2017.2722101.

[38] A. Cully, J.-B. Mouret, Evolving a behavioral repertoire for a walking robot, Evolut. Comput. 24 (1) (2016) 59–88, http://dx.doi.org/10.1162/EVCO_a_00143.

[39] V. Vassiliades, K. Chatzilygeroudis, J.-B. Mouret, Using centroidal voronoi tessellations to scale up the multidimensional archive of phenotypic elites algorithm, IEEE Trans. Evol. Comput. 22 (4) (2018) 623–630, http://dx.doi.org/10.1109/TEVC.2017.2735550.

[40] M. Duarte, J. Gomes, S.M. Oliveira, A.L. Christensen, EvoRBC: evolutionary repertoire-based control for robots with arbitrary locomotion complexity, in: T. Friedrich, F. Neumann, A.M. Sutton (Eds.), GECCO'16: Proceedings of the Genetic and Evolutionary Computation Conference, ACM, New York, NY, USA, 2016, pp. 93–100, http://dx.doi.org/10.1145/2908812.2908855.

[41] J. Gomes, A.L. Christensen, Comparing approaches for evolving high-level robot control based on behaviour repertoires, in: 2018 IEEE Congress on Evolutionary Computation (CEC), IEEE, Piscataway, NJ, USA, 2018, pp. 1–6, http://dx.doi.org/10.1109/CEC.2018.8477699.

[42] D.M. Bossens, D. Tarapore, QED: Using quality-environment-diversity to evolve resilient robot swarms, IEEE Trans. Evol. Comput. 25 (2) (2021) 346–357, http://dx.doi.org/10.1109/TEVC.2020.3036578.

[43] A. Cully, Y. Demiris, Quality and diversity optimization: A unifying modular framework, IEEE Trans. Evol. Comput. 22 (2) (2018) 245–259, http://dx.doi.org/10.1109/TEVC.2017.2704781.

[44] S. Kim, S. Doncieux, Learning highly diverse robot throwing movements through quality diversity search, in: P.A.N. Bosman (Ed.), GECCO'17: Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann Publishers, San Francisco, CA, USA, 2017, pp. 1177–1178, http://dx.doi.org/10.1145/3067695.3082463.

[45] S.A. Engebråten, J. Moen, O.A. Yakimenko, K. Glette, Evolving a repertoire of controllers for a multi-function swarm, in: K. Sim, P. Kaufmann (Eds.), Applications of Evolutionary Computation: 21st International Conference, EvoApplications 2018, in: Lecture Notes in Computer Science, vol. 10784, Springer, Cham, Switzerland, 2021, pp. 734–749, http://dx.doi.org/10.1007/978-3-319-77538-8_49.

[46] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, M. Birattari, T. Stützle, The irace package: iterated racing for automatic algorithm configuration, Oper. Res. Perspect. 3 (2016) 43–58, http://dx.doi.org/10.1016/j.orp.2016.09.002.

[47] M. Birattari, T. Stützle, L. Paquete, K. Varrentrapp, A racing algorithm for configuring metaheuristics, in: W.B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M.A. Potter, A.C. Schultz, J.F. Miller, E.K. Burke, N. Jonoska (Eds.), GECCO'02: Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation, Morgan Kaufmann Publishers, San Francisco, CA, USA, 2002, pp. 11–18.

[48] O. Maron, A.W. Moore, The Racing Algorithm: model selection for lazy learners, Artif. Intell. Rev. 11 (1–5) (1997) 193–225, http://dx.doi.org/10.1023/A:1006556606079.

[49] J. Lehman, K.O. Stanley, Abandoning objectives: evolution through the search for novelty alone, Evolut. Comput. 19 (2) (2011) 189–223, http://dx.doi.org/10.1162/EVCO_a_00025.

[50] J. Gomes, P. Mariano, A.L. Christensen, Systematic derivation of behaviour characterisations in evolutionary robotics, in: H. Sayama, J. Rieffel, S. Risi, R. Doursat, H. Lipson (Eds.), ALIFE 14: The Fourteenth International Conference on the Synthesis and Simulation of Living Systems, MIT Press, Cambridge, MA, USA, 2014, pp. 212–219, http://dx.doi.org/10.1162/978-0-262-32621-6-ch036.

[51] K.O. Stanley, R. Miikkulainen, Evolving neural networks through augmenting topologies, Evolut. Comput. 10 (2) (2002) 99–127, http://dx.doi.org/10.1162/106365602320169811.

[52] G. Francesca, M. Brambilla, V. Trianni, M. Dorigo, M. Birattari, Analysing an evolved robotic behaviour using a biological model of collegial decision making, in: T. Ziemke, C. Balkenius, J. Hallam (Eds.), From Animals To Animats 12: 12th International Conference on Simulation of Adaptive Behavior, SAB 2012, in: Lecture Notes in Computer Science, vol. 7426, Springer, Berlin, Germany, 2012, pp. 381–390, http://dx.doi.org/10.1007/978-3-642-33093-3_38.

[53] K. Hasselmann, A. Ligot, M. Birattari, Automatic modular design of robot swarms based on repertoires of behaviors generated via novelty search: supplementary material, 2022, https://iridia.ulb.ac.be/supp/IridiaSupp2022-002.

[54] K. Hasselmann, A. Ligot, G. Francesca, D. Garzón Ramos, M. Salman, J. Kuckling, F.J. Mendiburu, M. Birattari, Reference models for AutoMoDe, TR/IRIDIA/2018-002, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2018.

[55] 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, M. Dorigo, ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems, Swarm Intell. 6 (4) (2012) 271–295, http://dx.doi.org/10.1007/s11721-012-0072-5.

[56] L. Garattoni, G. Francesca, A. Brutschy, C. Pinciroli, M. Birattari, Software infrastructure for e-puck (and TAM), TR/IRIDIA/2015-004, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2015.

[57] W.J. Conover, Practical Nonparametric Statistics, third ed., in: Wiley Series in Probability and Statistics, John Wiley & Sons, New York, NY, USA, 1999.

[58] A. Ligot, M. Birattari, Simulation-only experiments to mimic the effects of the reality gap in the automatic design of robot swarms, Swarm Intell. 14 (2020) 1–24, http://dx.doi.org/10.1007/s11721-019-00175-w.

[59] A. Ligot, M. Birattari, On mimicking the effects of the reality gap with simulation-only experiments, in: M. Dorigo, M. Birattari, S. Garnier, H. Hamann, M. Montes de Oca, C. Solnon, T. Stützle (Eds.), Swarm Intelligence: 11th International Conference, ANTS 2018, in: Lecture Notes in Computer Science, vol. 11172, Springer, Cham, Switzerland, 2018, pp. 109–122, http://dx.doi.org/10.1007/978-3-030-00533-7_9.