Evaluation of alternative exploration schemes in the automatic modular design of robot swarms*

Gaëtan Spaey, Miquel Kegeleirs[0000−0002−9018−4995], David Garzón Ramos[0000−0001−7099−4213], and Mauro Birattari[0000−0003−3309−2194]

IRIDIA, Université Libre de Bruxelles, Belgium
mbiro@ulb.ac.be

Abstract. The swarm robotics literature has shown that complex tasks can be solved by large groups of simple robots interacting with each other and their environment. Most of these tasks require the robots to explore their environment, making exploration a building block of the behaviors of robot swarms. However, exploration schemes have rarely been thoroughly evaluated, especially in the context of automatic design. This is the case with AutoMoDe, an automatic modular design approach that designs control software by assembling predefined mission-agnostic modules that embed fixed and arbitrarily selected exploration schemes. In this paper, we study the influence of different exploration schemes on the automatic design of robot swarms. To do so, we introduce AutoMoDe-Coconut, a new variant of AutoMoDe with multiple configurable exploration schemes embedded within its modules. We test Coconut both in bounded and unbounded workspaces and we compare the results with those of AutoMoDe-Chocolate in order to understand the impact of the new exploration schemes. The results show that Coconut is prone to select exploration schemes that fulfill the requirements of the mission in hand. However, Coconut does not perform better than Chocolate, even in situations where the only exploration schemes available to Chocolate are at an apparent disadvantage. We conjecture that the overall exploration capabilities of the swarm are not the mere reflection of individual-level exploration schemes but result from a more complex interaction between the atomic behaviors of the individuals.

Keywords: Automatic design · exploration · random walk.

1 Introduction

A robot swarm is a large group of robots whose collective behavior results from local interactions of the robots between themselves and with their environment [9]. A robot swarm operates without relying on any external structure

* GS and MK contributed equally to this work and should be considered as co-first authors. The software used in the experiments was implemented by GS and MK. The experiments were designed by DGR and performed by GS and MK. The paper was drafted by GS and MK and edited by MK and MB; all authors read and commented the final version. The research was directed by MB.

or any form of centralized control [1, 32]. These characteristics make swarms of robots scalable, robust and flexible.

Unfortunately, the design of control software for robot swarms is a complex activity. Indeed, there is no reliable way to anticipate the global behavior of a swarm of robots based on the behavior of a single individual [12]. It is therefore common to resort to automatic design, for which multiple methods have been developed [26, 37, 10]. In particular, AutoMoDe [14] is an automatic modular design approach that produces control software by assembling preexisting software modules into an appropriate modular architecture—e.g., a finite-state machine or a behavior tree. The possible states come from a finite set of atomic behaviors, such as the attraction to light sources or the repulsion from other robots. A few methods belonging to the AutoMoDe family have been proposed so far. Most of them are variants of Vanilla, the first method proposed that meet the specifications of AutoMoDe [14]. In these variants, most of the atomic behaviors embed the same exploration scheme: ballistic motion.

We foresee that other exploration schemes, such as random walks [31, 27, 38, 11], could improve the exploration capabilities of robot swarms automatically generated via AutoMoDe.

To study the influence of different exploration schemes in automatic modular design of robot swarms, we introduce AutoMoDe-Coconut, a new variant of AutoMoDe able to select different exploration schemes. Following the tenets of the automatic offline design of robot swarms [4], we assess the capabilities of Coconut to design control software for missions that require the robot swarm to explore in different manners. To this aim, we conduct experiments on two classes of missions using realistic simulations and real robots experiments. We compare the performance of Coconut against the one of the state-of-the-art modular design method AutoMoDe-Chocolate [13]. We expect Coconut to outperform Chocolate in at least one class of mission thanks to its extended exploration capabilities. To the best of our knowledge, this is the first time different exploration schemes are compared in the context of automatic design.

The paper is structured as follows. In Section 2, we discuss related work in automatic design and exploration. In Section 3, we present Coconut, the automatic modular design method we investigate in the paper. In Section 4 we describe the experimental setup. In Section 5, we illustrate the results of the experiments. In Section 6, we conclude the paper and we sketch future research.

2 Related work

In single-robot systems, the control software is typically designed by hand by a human developer as the behavior of the robot is easy to derive from its specifications. In swarm robotics however, the link between the behavior of the individual robots that one should program and the global behavior of the swarm that one wishes to obtain is often particularly complex. Indeed, it is difficult to anticipate the behavior of a swarm solely based on the individual behavior of the robots [6]. The control software of the individual robots is therefore a trial and

error process, which is time consuming, prone to bias and errors, and difficult to replicate [5]. Automatic design appears to be a promising way to overcome the difficulties of generating control software for robot swarms [12].

Neuro-evolutionary robotics is the classical automatic design approach adopted in swarm robotics [26, 8]. In this approach, robots are controlled by a neural network whose parameters (and possibly the structure) are optimized using an evolutionary algorithm in an off-line process based on computer simulations [35, 29, 36]. The inputs of the neural networks are the readings of the sensors and outputs are the commands to be fed to the actuators. Unfortunately, the neuro-evolutionary approach is known to produce control software that crosses the reality gap poorly [19, 33]. Indeed, a noticeable drop in performance can be often observed when neural networks optimized in simulation are tested on real robots. This is the result of a sort of *over tting* of the control software to the simulator, which prevents it to then generalize to the real world [22].

An alternative approach to automatic design is the automatic modular design method proposed by Francesca et al.: AutoMoDe [14]. The original idea behind AutoMoDe is to inject a bias in the automatic design process by increasing the granularity of the control software architecture. This reduces the risk of overfitting the simulator and eventually increases the chance that the control software produced crosses the reality gap successfully, generalizing properly to reality. Multiple variants of AutoMoDe have been developed so far [13, 21, 18].

However, the exploration scheme used by AutoMoDe, ballistic motion, was selected arbitrarily from Vanilla and has been kept in all the following studies without a further discussion. Recent works have shown the relevance of the exploration scheme in robot swarms. Common random walks (Brownian motion [11], correlated random walk [31], Lévy walk [38] and Lévy taxis [27]) have been evaluated by Dimidov et al. [7] with a swarm of Kilobots. An optimal parametrization for these random walks was found with this configuration. Kegeleirs et al. [20] evaluated the same random walks, along with ballistic motion, for mapping with ten e-pucks and found that the parametrization does not generalize to other robotic platforms. Similarly, Ramachandran et al. [30] performed distributed mapping with three robots using a custom variant of Lévy walk. To the best of our knowledge, no study has been published that evaluates the performance of the aforementioned exploration schemes in the context of the automatic design of collective behaviors for robot swarms.

3 AutoMoDe-Coconut

Coconut builds on Chocolate [13]. As Chocolate, it belongs to the AutoMoDe class of methods originally defined by Francesca et al. [14]. These methods automatically generate control software by assembling predefined, mission-agnostic software modules. Like Chocolate, Coconut produces control software for the e-puck platform [25]. The control software produced by Coconut, like the one produced by Chocolate, has the form of a probabilistic finite-state machine. The modules are either low-level behaviors to be used as states of the state machine or

conditions to be associated with its edges. Conditions determine whether a transition should happen or not. Modules may have tunable parameters that modify their functioning. The topology of the probabilistic finite-state machine, the behaviors and the conditions to be included, and the value of their parameters are determined by an optimization algorithm that maximizes a mission-specific performance measure. The optimization algorithm adopted in Coconut, as well as in Chocolate, is Iterated F-race (irace) [23]. The only difference between Coconut and Chocolate is that, in Coconut, the modules have a parameter controlling the type of exploration scheme to use, whereas in Chocolate the scheme is fixed for each module. Indeed, Coconut embeds three different exploration schemes within its modules: ballistic motion with random rotations, ballistic motion with vector field and, random walk. As Coconut is identical to Chocolate in all other aspects, the discussion on performance differences can focus on the sole influence of these exploration schemes.

3.1 Robot platform

Coconut produces control software for the e-puck platform, extended with three hardware modules: the Overo Gumstix, the ground sensor, and the range and bearing. The e-puck is a circular two-wheeled robot, whose diameter is approximately 70 mm. It has 8 IR transceivers, positioned all around its body, that work both as light and proximity sensors. The Overo Gumstix module is a single-board computer that allows the e-puck to run Linux. The ground sensor module allows the e-puck to perceive the color of the floor. The range-and-bearing module [16] is an infrared communication device for local sensing and messaging. It operates by broadcasting a ping signal that can be received by robots within a range of about 0.7 m from the sender. A robot that receives a ping is able to estimate the relative position of the sender in polar coordinates. The capabilities of the e-puck platform are formally defined by the reference model RM 1.1 [17], see Table 1.

3.2 Set of modules

Coconut's modules are built upon those of Chocolate. They comprise 6 behaviors and 6 transitions:

Behaviors:

- rambling[1]: the robot explores randomly its environment;
- stop: the robot stands still;
- phototaxis: the robot goes towards the light source, if perceived;
- anti-phototaxis: the robot goes away from the light, if perceived;
- attraction: the robot goes towards its neighboring peers, if perceived;
- repulsion: the robot goes away from its neighboring peers, if perceived.

[1] Originally, this module was called exploration [14]. In this paper, we changed its denomination to avoid confusion with the notion of exploration scheme.

Table 1: Reference model RM1 of the e-puck robot [17]. RM1 abstracts sensors and actuators by defining the input and the output variables that are made available to the control software at each control step. Sensors are defined as input variables: the control software can only read them. Actuators are defined as output variables: the control software can only write them. Input and output variables are updated with a period of 100 ms.

sensor / actuator	variables
proximity	$prox_i \in [0; 1]$, with $i \in \{1; 2; \ldots; 8\}$
light	$light_i \in [0; 1]$, with $i \in \{1; 2; \ldots; 8\}$
ground	$ground_i \in \{white; gray; black\}$, with $i \in \{1; 2; 3\}$
range-and-bearing	$n \in [0; 20]$ $r_m \in [0; 0.70]$, with $m \in \{1; 2; \ldots; 20\}$ $b_m \in [0; 2\ ]$ rad, with $m \in \{1; 2; \ldots; 20\}$
wheels	$v_l; v_r \in [-0.12; 0.12]$ m/s

Conditions:

- black-floor: change state if floor is black;
- white-floor: change if it is white;
- gray-floor: change if it is gray;
- neighbor-count: change if sufficiently many neighboring peers are perceived;
- inverted-neighbor-count: change if they are sufficiently few;
- fixed-probability: change state with a fixed probability.

The behaviors are identical to those of Chocolate except for the exploration scheme adopted. In Chocolate, fixed default exploration schemes are used in the rambling module (ballistic motion with random rotations) as well as in the phototaxis, anti-phototaxis, attraction and repulsion modules when no light/no neighboring peers are perceived (ballistic motion with vector field). In Coconut, these five modules do not use a default exploration scheme but have instead a new parameter that has 3 possible values: BMVF, BMRR, and RW.

If = BMVF, the exploration scheme is a ballistic motion with vector field. The robot follows the two-dimensional vector $w = w_b$ w_o, where w_b represents the ballistic vector and w_o the perceived obstacle vector. The ballistic vector is trivially defined as $w_b = (1; \angle 0)$ and represent a straight motion. The obstacle vector w_o is calculated with equation 1.

$$w_o = \sum_{i=1}^{8} (r_i; \angle b_i) \tag{1}$$

Where r_i represents the reading of i, one of the eight proximity sensors of the robot and b_i the angle between this sensor and the front of the robot. The vector w_o therefore represents the average position of the sensed obstacle(s) as the sum of the eight vectors corresponding to proximity readings.

If = BMRR, the exploration scheme is a ballistic motion with random rotations. The robot moves in a straight line. When it encounters an obstacle, it

Table 2: Different value ranges of the key parameters of the random walks, along with the corresponding distributions [27, 7].

	Move length		Turning angle	
Brownian motion	Asymptotically Gaussian-like	3	Uniform	0
Correlated random walk	Asymptotically Gaussian-like	3	Wrapped Cauchy	$\in [0; 1]$
Lévy walk	Power law	$\in]1; 3]$	Uniform	0
Lévy taxis	Power law	$\in]1; 3]$	Wrapped Cauchy	$\in [0; 1]$

turns on itself for a random number of control cycles uniformly chosen between $[0; \ ]$, where is a parameter of the module. The parameter is an integer in the range $[0; 100]$.

If $= \mathsf{RW}$, the exploration scheme is a random walk. The robot follows the two-dimensional vector $w = w_{Lt} \quad w_o$, where w_{Lt} represents the Lévy taxis vector and w_o is calculated with equation 1, as in the ballistic motion with vector field. The Lévy taxis vector is calculated as $w_{Lt} = (1; \angle T_a)$ where T_a is the turning angle defined by equation 2.

$$T_a = 2 \arctan \quad \frac{1}{1+} \tan \quad (r \quad 0{:}5) \quad + bias \tag{2}$$

The turning angle changes after a number of control cycles governed by the movement length M_l defined by equation 3.

$$M_l = L_{min} r^{\frac{1}{1-\mu}} \tag{3}$$

These equations depend on the parameters and , 2 parameters of the module. The parameter is real-valued and chosen in the range $]1; 3]$. The parameter is real-valued as well and chosen in the range $[0; 1]$. Table 2 presents the values of and for the major state-of-the-art random walks that can be modeled by Equations 2 and 3. These additional parameters have also an effect on the search space size: it is larger for Coconut than for Chocolate.

3.3 Automatic design process

Coconut produces control software in the form of probabilistic finite-state machines. The topology of the probabilistic finite-state machine, the modules to be included and their parameters are defined by an optimization process. The space of the probabilistic finite-state machines that Coconut can possibly generate is constrained to those comprising at most 4 states having each at most 4 outgoing edges.

As an optimization algorithm, Coconut uses the implementation of Iterated F-race provided by the R package irace [23] with its default parameters. Iterated

F-race is based on F-race [2], a racing procedure where a set of candidate solutions are randomly sampled and then sequentially evaluated, over a set of test cases, to eventually select the most suitable one. Along the sequential evaluation of candidate solutions, a Friedman test is repeatedly performed to identify candidate solutions that perform significantly worse than at least another one. These solutions are discarded so that the evaluation can focus on the best ones. The algorithm terminates when only one candidate solution remains or when a predefined budget of evaluations is depleted. Iterated F-race consists of multiple iterations of F-race. After the first iteration, each subsequent one operates on a set of candidate solutions that are sampled around those that the previous iteration selected as the best ones. The algorithm terminates when a predefined budget of evaluations is depleted.

Within the optimization process, simulations are performed using the ARGoS3 simulator [28], version beta 48, together with the argos3-epuck library [15]. ARGoS3 is a modular multi-physics robot simulator specifically conceived to simulate robot swarms. Coconut uses the 2D dynamic physics engine of ARGoS3 to simulate the robots and the environment. The argos3-epuck library provides low-level implementations of the sensors and actuators of the e-puck robot with fine control on noise levels for all actuators and sensors. ARGoS3 and the argos3-epuck library inject a realistic level of sensor and actuator noise in all simulations as suggested by Miglino et al. [24] as a good practice for reducing the impact of the reality gap.

4 Experimental setup

In order to assess the performance impact of the new exploration schemes integrated in its modules, we compare Coconut to Chocolate on a set of missions, both in bounded and unbounded workspaces. Similarly to previous studies in automatic modular design, we also compare Chocolate and Coconut to Evostick, an automatic design method that implements a typical evolutionary robotics setup. Evostick was introduced in Francesca et al [14] to define a yardstick against which AutoMoDe variants can be compared. We expect Coconut to produce results similar to those of Chocolate in bounded workspace but to outperform Chocolate in unbounded workspace. The choice of the missions is motivated by the need to challenge both the general problem-solving capabilities of the two methods and their exploration capabilities. Therefore, the missions consist in missions already used to test other AutoMoDe variants as well as a new mission specifically targeting the exploration capabilities offered by the new exploration schemes. The chosen missions are aggregation, foraging, and grid exploration.

4.1 Missions

Each mission takes place in a dodecagonal workspace of $4.91\,\mathrm{m}^2$ surrounded by walls that are tall enough to prevent the robots from seeing anything beyond them. The floor is gray with the exception of black or white areas specific to

each mission. All missions are performed by a swarm of 20 e-puck robots for a duration of 120 s. In the following descriptions, the coordinates are in meters with the origin of the axes at the center of the workspace. The x axis points right and the y axis points up. The three missions are detailed below.

Aggregation: The robots must aggregate as fast as possible on a black spot at the center of the workspace. The floor is completely gray except for a black circular area of diameter 0.60 m at the center of the workspace. At the beginning of the experiment, the 20 robots are randomly placed in the whole workspace. Figure 1a shows the workspace of the mission. The performance of the swarm is measured by the sum of the time spent, in seconds, by each robot in the black area during the whole duration of the mission. Formally:

$$F_{aggregation} = \sum_{i=1}^{N} T_i \tag{4}$$

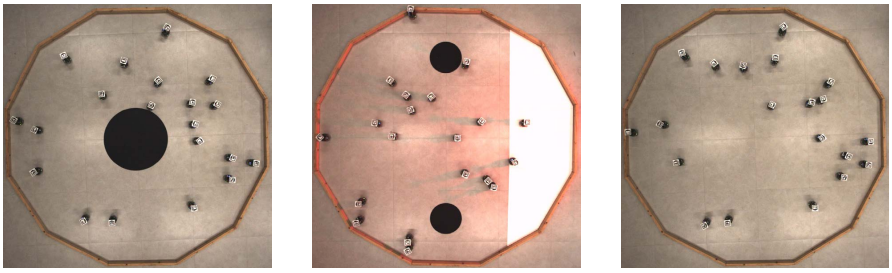
Where $N = 20$ is the number of robots and T_i is the aggregated time spent in the black area by robot i during the whole duration of the mission.

Foraging: The robots must retrieve as many objects as possible from two sources and drop them in a specific area, the nest. The sources and nest are represented respectively by two black spots and a white area. The two black spots are black circular areas of diameter 0.30 m located at the coordinates $(0, 0.75)$ and $(0, -0.75)$. The white area covers the whole region of the workspace with $x > 0.60$. Moreover, a light source is placed behind the nest at coordinates $(1.25, 0)$ at 0.75 m from the ground. Figure 1b shows the workspace of the mission. Since the e-puck robot doesn't have grasping capabilities, the transportation of objects is abstracted. Therefore, it is supposed that a robot grabs an object (if it isn't already holding one) when it enters a source and drops the object (if it has one) when it enters the nest. At the beginning of the experiment, the 20 robots are randomly placed in the workspace. The performance of the swarm is measured by the sum of the number of objects retrieved by each robot, during the whole duration of the mission. Formally:

$$F_{foraging} = \sum_{i=1}^{N} O_i \tag{5}$$

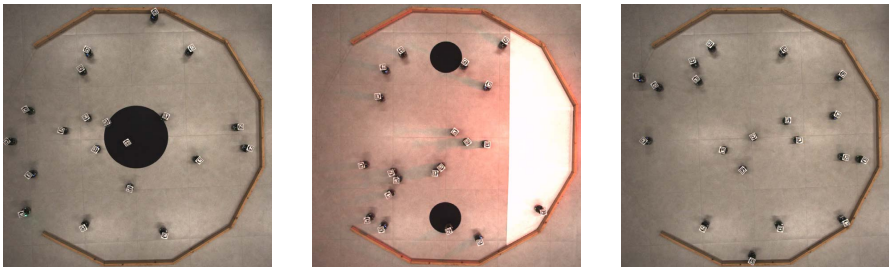
Where $N = 20$ is the number of robots and O_i is the number of objects retrieved by the robot i.

Grid exploration: The robots must explore and cover as much space as possible. The floor is completely gray. At the beginning of the experiment, the 20 robots are randomly placed in the workspace. Figure 1c shows the workspace of the mission. In order to measure the performance of the swarm, the arena is divided in a grid of 10 tiles by 10 tiles. For each tile, we retain the time t elapsed since the last time it was visited by a robot. Each time the tile is visited by a robot, this time is reset to 0. The performance of the swarm is measured by the sum over all control cycles of the opposite of the average time t over all the tiles.



(a) Aggregation (b) Foraging (c) Grid exploration

Fig. 1: Workspaces of the 3 bounded missions, including an example of initial positions for the robots



(a) Aggregation (b) Foraging (c) Grid exploration

Fig. 2: Workspaces of the 3 unbounded missions, including an example of initial positions for the robots

Formally:

$$F_{gridexploration} = \sum_{i=1}^{N_{cc}} \frac{1}{N_{tiles}} \sum_{j=1}^{N_{tiles}} t_{ij} \tag{6}$$

Where N_{cc} is the number of control cycles for the whole experiment, N_{tiles} is the number of tiles and t_{ij} is the time, at the control cycle i, since the tile j was crossed by a robot.

For each of these missions, we evaluated an alternative version in an unbounded workspace. The only difference between a mission and its unbounded counterpart is that 3 walls have been removed from the workspace of the unbounded workspace mission. The 3 walls are the same for all the missions, namely the leftmost wall and its 2 neighbors. Figure 2 displays the workspaces of those alternative unbounded missions. These 2 sets of missions constitute the bounded and the unbounded classes of missions studied in this paper.

4.2 Protocol

Coconut, Chocolate and Evostick are executed 10 times on each of the 3 missions of each class with a budget of 100:000 evaluations. This design process produces

10 instances of control software per mission and per method. Each of these instances is then evaluated once on its respective mission [2]. The results of these evaluations are then presented mission by mission.

Then, each instance is uploaded on real e-pucks and evaluated once in a real environment with the same geometry and features as in the simulation. The results of these evaluations are also presented for each mission.

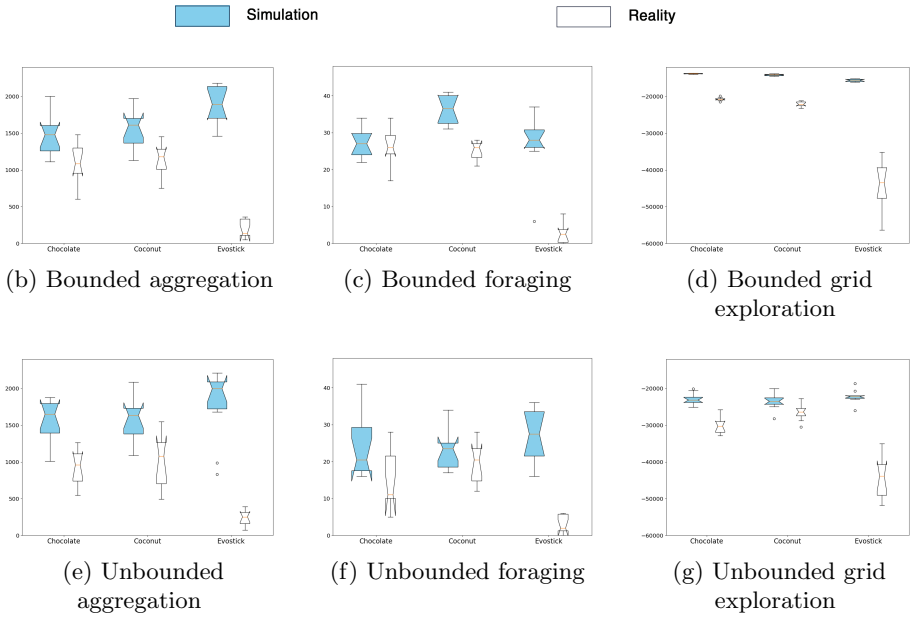
The evaluation of the performance on each mission is represented by notched box plots. For each mission, the score obtained in simulation and in reality for Coconut, Chocolate and Evostick is reported. Statements about the relative performance of the three methods on a specific mission are supported by the confidence intervals of those box plots. The evaluation of the aggregated performance over all of the missions is represented by a Friedman test. Once again, statements about the relative performance of the two methods are supported by the confidence intervals of this test. Any statement like "A performs *signi cantly* better/worse than B" means that the confidence intervals of the box plots of the scores obtained or the Friedman test for A and B do not overlap.

In order to interpret the observed performance of the automatic modular design methods, one needs to have some insight into the modules used by the two variants of AutoMoDe. We use two ways to measure the use of the different modules during a mission. The first one consists in counting, for each module, the proportion of instances of control software using this module in their finite-state machine. While this measurement gives some information about the finite-state machines and the behavior of the control software, it also shows modules that might not actually be used at runtime. Indeed, some states of the finite-state machines can be bypassed completely by high-probability transitions, making them useless. The second measurement is the average (across all of the robots of the swarm and all instances of control software of the mission) of the proportion of time each robot uses the behavior of each module. While this measure gives a better idea of the actual use of the different modules at runtime, it fails to differentiate important modules used for a short time and useless modules used as transitions. For that reason, the two measurements are compared.

5 Results

We present the qualitative analysis of the results. Demonstrative videos, code, and additional results are available in [34]. The performance of Chocolate, Coconut and Evostick on the two classes of missions are shown in Figure 3. For all missions, Evostick performs the same or better than both AutoMoDe methods in simulation but completely fails in reality. This is coherent with previous results obtained in the litterature [13, 18]. For Chocolate and Coconut, the results in simulation are close to those in reality for all the missions and for both methods although a small reality gap can be seen. An analysis of the exploration schemes used by Coconut for the different missions is shown in Figure 4. We observe

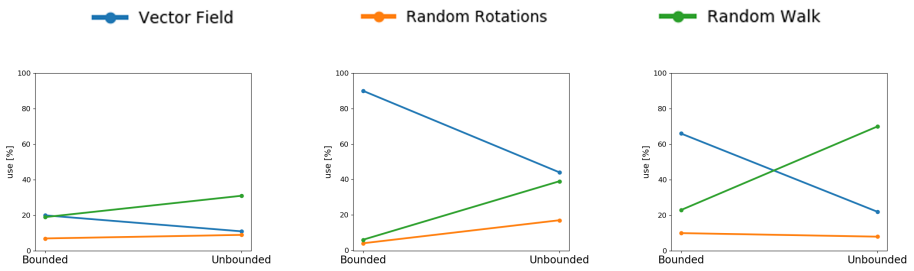
[2] This protocol has been used in [14, 13, 21, 22, 18] and is further discussed in [3]



(b) Bounded aggregation (c) Bounded foraging (d) Bounded grid exploration

(e) Unbounded aggregation (f) Unbounded foraging (g) Unbounded grid exploration

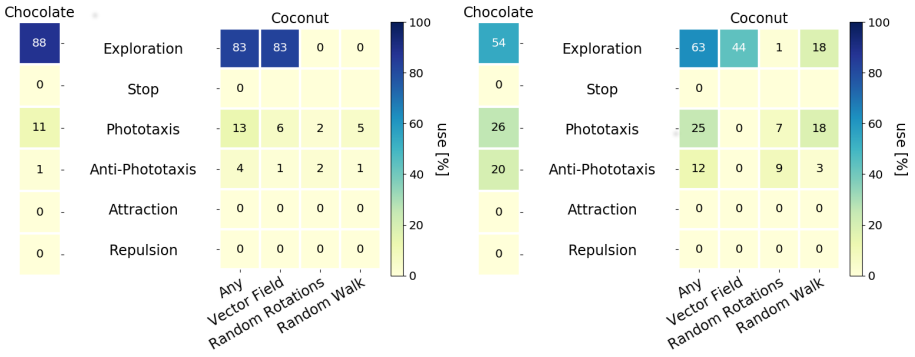
Fig. 3: Performance of Chocolate, Coconut and Evostick on aggregation, foraging and grid exploration, in bounded (top) and unbounded (bottom) workspaces. The higher the better.



(b) Aggregation (c) Foraging (d) Grid exploration

Fig. 4: Runtime use of the ballistic motion with vector field, ballistic motion with random rotations and random walk exploration schemes in control software designed by Coconut for aggregation, foraging and grid exploration, in bounded and unbounded workspaces.

that Coconut selects the ballistic motion for the bounded missions to promote exploration. Indeed, ballistic motion allows the robots to cover larger distances. For the unbounded missions, Coconut switches to random walk to promote exploitation. The random walk tends to keep robots in the same area and hence



(a) Bounded foraging (b) Unbounded foraging

Fig. 5: Runtime use of the modules in control software designed by Chocolate and Coconut for foraging in bounded (left) and unbounded (right) workspaces.

reduces the risks to lose robots. In this sense, the exploration scheme has an influence in the unbounded class of missions.

Performance-wise, Coconut performs similarly to Chocolate in most missions. Differences between Chocolate and Coconut can only be observed for the bounded versions of foraging and grid exploration. However, these differences do not result from the exploration capabilities of Coconut but rather from the difference between the search space size of both methods. Indeed, Coconut has a larger search space and hence explores more solutions. Eventually, Coconut can find a solution that Chocolate cannot produce. In particular, this is the case for the bounded foraging mission. On the contrary, Chocolate will explore fewer solutions and converge to an optimal solution faster than Coconut. Eventually, this can translate into a slightly better performance, like for the bounded grid exploration. All in all, there is no improvement from the addition of new exploration schemes, even for the unbounded class of missions for which relying only on ballistic motion is an apparent disadvantage.

Therefore, we analyze the finite-state machines produced by Chocolate and Coconut on bounded and unbounded missions. We focus here on foraging but the following observations can also be made on the other missions. We can see in Figure 5 that, in the unbounded mission, both Chocolate and Coconut decrease their use of the exploration module to rely more on the light (phototaxis and anti-phototaxis). The light is indeed at the opposite of the open part of the workspace and helps the robots to stay within the workspace. Considering that the performance of Chocolate and Coconut are similar, random walk does not help Coconut to achieve a better behavior. Therefore, we conjecture that Chocolate is able to adapt to different classes of missions by combining the modules at its disposal, without relying on different exploration schemes. In this sense, the exploration capabilities of Chocolate emerge from the interaction between its different modules rather than being the direct result of specific embedded exploration schemes.

6 Conclusions

We introduced Coconut, an automatic modular design method able to select different exploration schemes for each of its modules. We evaluated Coconut on three missions in bounded and unbounded workspaces. We observed that Coconut is prone to select exploration schemes that fit the requirements of the mission at hand. In bounded workspace, the control software produced uses mainly ballistic motion as it allows robots to cover bigger distances than random walks and promotes hence exploration. On the contrary, in unbounded workspaces, the control software produced uses mainly instances of random walk as it promotes exploitation behaviors that help maintaining the robots within the workspace. In this sense, the influence of the exploration scheme is only relevant for the class of missions in which the workspace is unbounded.

We also compared Coconut to Chocolate, the state-of-the-art automatic modular design method. Performance-wise, we could not observe a conclusive difference between the control software produced by Chocolate and Coconut, even in unbounded workspaces. Other exploration schemes do not improve the performance of the swarm as we expected but the results are still interesting as they allow us to make the following observations.

Even though Chocolate could only rely on ballistic motion as exploration scheme, it yielded a similar performance to Coconut in unbounded workspaces. Chocolate was hence able to design a control software preventing the robots to leave the workspace by combining its different modules. This means that exploration capabilities come from the interaction between atomic behaviors and not only from the exploration schemes embedded in the modules. In this sense, we saw that AutoMoDe adjusts to produce appropriate exploration strategies for the task at hand.

For the class of missions conceived so far, ballistic motion has proven to be a sufficiently appropriate exploration scheme. Still, whether random walk exploration could be a suitable solution in other contexts needs to be further explored.

Acknowledgements

The project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 681872). Mauro Birattari acknowledges support from the Belgian Fonds de la Recherche Scientifique – FNRS. David Garzón Ramos acknowledges support from the Colombian Administrative Department of Science, Technology and Innovation – COLCIENCIAS.

References

1. Beni, G.: From swarm intelligence to swarm robotics. In: International Workshop on Swarm Robotics. pp. 1–9. Springer (2004)

2. Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K.: A racing algorithm for configuring metaheuristics. In: Langdon, W., et al. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO. pp. 11–18. Morgan Kaufmann, San Francisco CA (2002)
3. Birattari, M.: Notes on the estimation of the expected performance of automatic methods for the design of control software for robot swarms. Tech. Rep. TR/IRIDIA/2020-010, IRIDIA, Université Libre de Bruxelles, Belgium (2020)
4. Birattari, M., Ligot, A., et al.: Automatic off-line design of robot swarms: a manifesto. Frontiers in Robotics and AI **1**(1), 1 (2019)
5. Bozhinoski, D., Birattari, M.: Designing control software for robot swarms: Software engineering for the development of automatic design methods. In: Robotics Software Engineering, RoSE. pp. 33–35. ACM, New York (2018)
6. Brambilla, M., Ferrante, E., Birattari, M., Dorigo, M.: Swarm robotics: a review from the swarm engineering perspective. Swarm Intelligence **7**(1), 1–41 (2013)
7. Dimidov, C., Oriolo, G., Trianni, V.: Random Walks in Swarm Robotics: An Experiment with Kilobots. In: Swarm Intelligence, vol. 9882, pp. 185–196. Springer, Cham, Switzerland (2016)
8. Doncieux, S., Bredeche, N., Mouret, J.B., Eiben, A.E.G.: Evolutionary robotics: what, why, and where to. Frontiers in Robotics and AI **2**, 4 (2015)
9. Dorigo, M., Birattari, M., Brambilla, M.: Swarm robotics. Scholarpedia **9**(1), 1463 (2014)
10. Duarte, M., Oliveira, S., Christensen, A.: Evolution of hierarchical controllers for multirobot systems. In: Artificial Life Conference Proceedings 14. pp. 657–664. MIT Press (2014)
11. Feynman, R.P., Leighton, R.B., Sands, M.L.: The Feynman lectures on physics. volume 1: Mainly mechanics, radiation, and heat. Basic Books, New York (2011)
12. Francesca, G., Birattari, M.: Automatic design of robot swarms: achievements and challenges. Frontiers in Robotics and AI **3**(29), 1–9 (2016)
13. Francesca, G., Brambilla, M., Brutschy, A., Garattoni, L., Miletitch, R., Podevijn, G., Reina, A., Soleymani, T., Salvaro, M., Pinciroli, C., Birattari, M.: AutoMoDe-Chocolate: automatic design of control software for robot swarms. Swarm Intelligence **9**(2/3), 125–152 (2015)
14. Francesca, G., Brambilla, M., Brutschy, A., Trianni, V., Birattari, M.: AutoMoDe: A novel approach to the automatic design of control software for robot swarms. Swarm Intelligence **8**(2), 89–112 (2014)
15. Garattoni, L., Francesca, G., Brutschy, A., Pinciroli, C., Birattari, M.: Software infrastructure for e-puck (and TAM). Tech. Rep. TR/IRIDIA/2015-004, IRIDIA, Université libre de Bruxelles, Belgium (2015)
16. Gutiérrez, Á., Campo, A., Dorigo, M., Donate, J., Monasterio-Huelin, F., Magdalena, L.: Open e-puck range & bearing miniaturized board for local communication in swarm robotics. In: IEEE International Conference on Robotics and Automation, ICRA. pp. 3111–3116. IEEE Press, Piscataway NJ (2009)
17. Hasselmann, K., Ligot, A., Francesca, G., Birattari, M.: Reference models for AutoMoDe. Tech. Rep. TR/IRIDIA/2018-002, IRIDIA, Université libre de Bruxelles, Belgium (2018)
18. Hasselmann, K., Robert, F., Birattari, M.: Automatic design of communication-based behaviors for robot swarms. In: Dorigo, M., et al. (eds.) Swarm Intelligence, ANTS, LNCS, vol. 11172, pp. 16–29. Springer, Cham, Switzerland (2018)
19. Jakobi, N., Husbands, P., Harvey, I.: Noise and the reality gap: the use of simulation in evolutionary robotics. In: Morán, F., et al. (eds.) Advances in Artificial Life. LNCS, vol. 929, pp. 704–720. Springer, London, UK (1995)

20. Kegeleirs, M., Garzón Ramos, D., Birattari, M.: Random walk exploration for swarm mapping. In: Towards Autonomous Robotic Systems. TAROS 2019. LNCS, vol. 11650, pp. 211–222. Springer, Cham, Switzerland (2019)
21. Kuckling, J., Ligot, A., Bozhinoski, D., Birattari, M.: Behavior trees as a control architecture in the automatic modular design of robot swarms. In: Dorigo, M., et al. (eds.) Swarm Intelligence, ANTS, LNCS, vol. 11172, pp. 30–43. Springer, Cham, Switzerland (2018)
22. Ligot, A., Birattari, M.: On mimicking the effects of the reality gap with simulation-only experiments. In: Dorigo, M., et al. (eds.) Swarm Intelligence, ANTS, LNCS, vol. 11172, pp. 109–122. Springer, Cham, Switzerland (2018)
23. López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., Stützle, T.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives **3**, 43–58 (2016)
24. Miglino, O., Lund, H., Nolfi, S.: Evolving mobile robots in simulated and real environments. Artificial life **2**(4), 417–434 (1995)
25. Mondada, F., Bonani, M., Raemy, X., Pugh, J., et al.: The e-puck, a robot designed for education in engineering. In: Gonçalves, P., Torres, P., Alves, C. (eds.) Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions. pp. 59–65. Instituto Politécnico de Castelo Branco, Portugal (2009)
26. Nolfi, S., Floreano, D.: Evolutionary Robotics. MIT Press, Cambridge MA (2000)
27. Pasternak, Z., Bartumeus, F., Grasso, F.W.: Lévy-taxis: a novel search strategy for finding odor plumes in turbulent flow-dominated environments. Journal of Physics A: Mathematical and Theoretical **42**(43), 434010 (2009)
28. Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., et al.: ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. Swarm Intelligence **6**(4), 271–295 (2012)
29. Quinn, M., Smith, L., Mayley, G., Husbands, P.: Evolving controllers for a homogeneous system of physical robots: structured cooperation with minimal sensors. Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences **361**(1811), 2321–2343 (2003)
30. Ramachandran, R.K., Kakish, Z., Berman, S.: Information correlated Lévy walk exploration and distributed mapping using a swarm of robots. arXiv (2019)
31. Renshaw, E., Henderson, R.: The correlated random walk. Journal of Applied Probability **18**(02), 403–414 (1981)
32. Şahin, E.: Swarm robotics: From sources of inspiration to domains of application. In: International workshop on swarm robotics. pp. 10–20. Springer (2004)
33. Silva, F., Duarte, M., Correia, L., Oliveira, S., Christensen, A.: Open issues in evolutionary robotics. Evolutionary Computation **24**(2), 205–236 (2016)
34. Spaey, G., Kegeleirs, M., Garzón Ramos, D., Birattari, M.: Evaluation of alternative exploration schemes in the automatic modular design of robot swarms: Supplementary material. http://iridia.ulb.ac.be/supp/IridiaSupp2020-002 (2020)
35. Trianni, V.: Evolutionary Swarm Robotics. Springer, Berlin, Germany (2008)
36. Trianni, V., López-Ibáñez, M.: Advantages of task-specific multi-objective optimisation in evolutionary robotics. PloS one **10**(8) (2015)
37. Watson, R., Ficici, S., Pollack, J.: Embodied evolution: distributing an evolutionary algorithm in a population of robots. Robotics and Autonomous Systems **39**(1), 1–18 (2002)
38. Zaburdaev, V., Denisov, S., Klafter, J.: Lévy walks. Reviews of Modern Physics **87**(2), 483–530 (2015)