ULB

ECOLE
POLYTECHNIQUE
DE BRUXELLES

Assessing and forecasting the performance of optimization-based design methods for robot swarms

Experimental protocol & pseudo-reality predictors

Thesis presented by Antoine LIGOT

in fulfillment of the requirements of the PhD Degree in Engineering and Technology
("Docteur en Sciences de l'Ingénieur et Technologie")
Année académique 2022-2023

Supervisor: Professor Mauro BIRATTARI
IRIDIA—Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Articifielle

European Commission

Horizon 2020
European Union funding
for Research & Innovation

In memory of my beloved grandfather Edmond Zorn

The thesis

Understanding that the reality-gap problem is akin to overfitting (as it is encountered in machine learning), enables more principled and accurate ways to evaluate automatic methods for the design of robot swarms.

Summary

Which off-line fully-automatic optimization-based design method produces control software that will yield the best performance once executed on a swarm of physical robots? This fundamental question cannot currently be answered due to the lack of two elements: i) an appropriate experimental protocol for the evaluation and comparison of fully-automatic design methods, and ii) a procedure that reliably predicts the real-world performance of control software. This dissertation focuses on addressing this void.

The literature on optimization-based design of robot swarms suffers from the absence of a clearly established state of the art. It is in fact, for the most part, a collection of feasibility studies, and little effort has been devoted to the systematic empirical evaluation and comparison of the methods or ideas proposed. Recent papers have formally characterized two approaches to the optimization-based design that were entangled in the literature: the fully-automatic and the semi-automatic approaches. In light of this novel categorization, we show that the experimental protocols employed so far for the evaluation and comparison of design methods do not respect the tenets of fully-automatic design, and we propose one that does.

One of the most challenging issues when designing control software off-line on the basis of a simulation model is the reality gap: the unavoidable discrepancies between the design model and reality. It is generally understood that, because of the reality gap, the design model overestimates the performance that control software eventually yields when executed on physical robots. As a result, conducting expensive and time consuming tests on physical robots is mandatory to reliably assess control software. We introduce the concept of pseudo-reality: a simulation model, different from the one used in the design, whose purpose is to evaluate control software. With this concept, we show via a series of experiments that the reality-gap problem is to be understood as a generalization problem, akin to the one encountered in machine learning. We also use it to conceive several simulation-only predictors of real-world performance, and we assess their accuracy with a large dataset of observations collected from previous studies. Results show that the pseudo-reality predictors we propose are more accurate than the current practice for predicting the expected performance of control software for robot swarms.

Original contributions

The following is a summary of the original contributions in this thesis:

Review of the literature on optimization-based design: We classify the relevant literature on the basis of the semi-automatic/fully-automatic classification. In particular, we focus on the elements of the existing works that classify them as belonging to the semi-automatic approach or the fully-automatic one, and on the experimental protocols used. We then reckon that works that belong to the semi-automatic approach do not compare the performance of design methods, and that works that belong to the fully-automatic one adopt experimental protocols that are appropriate when one aims at evaluating the performance of design methods for specific missions, but not for a whole class of mission, as it should be.

An experimental protocol for fully-automatic design: We propose an experimental protocol for the evaluation and comparison of fully-automatic design methods. This protocol is characterized by two notable elements: a way to define benchmarks for the evaluation and comparison of design methods, and a sampling strategy that minimizes the variance when estimating their expected performance.

MG 1, the first mission generator for swarm robotics: We illustrate the concept of mission generator by presenting one we named MG 1, short for *mission generator 1*. We use MG 1 in an illustrative study in which we compare two off-line fully-automatic design methods that were presented in previous publications. MG 1 is, to the best of our knowledge, the first generator of missions for swarm robotics. MG 1 is shared publicly.

A taxonomy for approaches to handle the reality gap: We propose a novel taxonomy for the classification of approaches proposed to limit the performance drops suffered by generated control software when going from simulation to reality. This taxonomy stands on the working hypothesis that drove the creation of each approach, and on the element of the design process that is targeted.

The concept of pseudo-reality: We introduce the concept of *pseudo-reality*, which refers to a secondary simulation model, different from the one used in the design

process, that is used for the evaluation of control software and hence plays the role of reality. In this thesis, we conduct experiments with swarms of e-puck robots and use the ARGoS3 simulator to simulate them. We consider pseudo-reality models that differ from the design model by the amount of noise injected to the sensors and actuators of the robots.

Disproof of the complexity assumption: We call the *complexity assumption* the common belief that emerges from the literature that claims that effects of the reality gap are due to the fact that the simulation conditions in which control software is designed are too simplistic with respect to the real-world environment in which it is eventually evaluated. We disprove the complexity assumption via experiments in which we create a simulation-only, artificial reality gap between the design model and a pseudo-reality we named M_B.

Three pseudo-reality predictors: We propose three simulation-only predictors of real-world performance of control software for robot swarms created on the basis of the concept of pseudo-reality. The first one, P_{M_B}, consists in the evaluation of control software on the fixed and handcrafted simulation model M_B. The second one, P_{R_1}, consists in the evaluation of an instance of control software on a single model sampled from a range R of possible ones. The third one, P_{R_k}, consists in the evaluation of an instance on $k > 1$ models sampled from R.

DS 1, the first dataset of reused control software for swarm robotics: We assess the accuracy of the three aforementioned pseudo-reality predictors. To perform a meaningful assessment, a large amount of instances of control software is necessary. Rather than generating enough instances of control software and evaluating them on physical robots ourselves, we collected and reused data from 7 previously published studies in off-line optimization-based design (i.e., a total of 1021 instances of control software generated by 18 different design methods for 45 missions). Reusing control software for a different purpose than the one it was originally produced is, to the best of our knowledge, a first in swarm robotics. We create the dataset DS 1—short for *dataset 1*—containing the data collected and the predictions of real-world performance made by our pseudo-reality predictors. DS 1 is shared publicly.

Statement

This thesis presents an original work that has never been submitted to the Université libre de Bruxelles or any other institution for the award of a Doctoral degree.
Some parts of this thesis are based on a number of peer-reviewed articles that the author, together with other co-workers, has published in the scientific literature. The classification of the optimization-based design methods according to the online/offline and semi-automatic/fully-automatic categorizations of Chapter 2 is based on:

> Birattari M., **Ligot A.**, Hasselmann K. (2020) Disentangling automatic and semi-automatic approaches to the optimization-based design of control software for robot swarms. *Nature Machine Intelligence* 2(9):494–499.

The experimental protocol proposed and illustrated in Chapter 3 is based on:

> **Ligot A.**, Cotorruelo A., Garone E., Birattari M. (2022) Towards an empirical practice in off-line fully-automatic design of robot swarms. *IEEE Transactions on Evolutionary Computation* 26(6):1236-1245.

The formal proof of the theorem presented in Chapter 3 reported here as Appendix A is based on:

> Cotorruelo A., **Ligot A.**, Garone E., Birattari M. (2021) Minimizing the variance in the estimation of the performance of a method for the fully-automatic design of robot swarms: a mathematical proof. Technical Report TR/IRIDIA/2021-007, IRIDIA, Université libre de Bruxelles, Belgium.

The review and taxonomy of the methods presented as solutions to the reality gap, as well as the introduction of the concept of pseudo-reality and the two-stage experiments used to disprove the complexity assumption of Chapter 4 are based on:

> **Ligot A.** and Birattari M. (2018) On mimicking the effects of the reality gap with simulation-only experiments. In Dorigo M. et al., editors, *Swarm Intelligence—ANTS 2018*, volume 11172 of *LNCS*, 109–122. Springer, Cham, Switzerland.

The description of the methodology to conceive pseudo-reality models and the associated experiments of Chapter 5 are based on:

Ligot A. and Birattari M. (2019) Simulation-only experiments to mimic the effects of the reality gap in the automatic design of robot swarms. *Swarm Intelligence* 14:1–24.

The definition of the pseudo-reality predictors of real-world performance of control software and the assessment of their accuracy in Chapter 6 are base on:

Ligot A. and Birattari M. (2022) On using simulation to predict the performance of robot swarms. *Scientific Data* 9:788.

Ideas developed in this thesis emerged from discussions or were supported by works that the author, together with co-authors, has published in the scientific literature but that are not part of the manuscript:

Birattari M., **Ligot A.**, Bozhinoski D., Brambilla M., Francesca G., Garattoni L., Garzón Ramos D., Hasselmann K., Kegeleirs M., Kuckling J., Pagnozzi F., Roli A., Salman M,. Stützle T. (2019) Automatic off-line design of robot swarms: a manifesto. *Frontiers in Robotics and AI* 6:59.

Hasselmann K., **Ligot A.**[‡], Ruddick J., Birattari M. (2020) Empirical assessment and comparison of neuro-evolutionary methods for the automatic off-line design of robot swarms. *Nature Communications* 12:4345.

Birattari M., **Ligot A.**, Francesca G. (2021) AutoMoDe: a modular approach to the automatic off-line design and fine-tuning of control software for robot swarms. In Pillay N. and Qu R., editors, *Automated Design of Machine Learning and Search Algorithms*, Natural Computing Series, Springer, Cham, Switzerland.

Ligot A., Hasselmann K., Birattari M. (2020) AutoMoDe-Arlequin: neural networks as behavioral modules for the automatic design of probabilistic finite state machines. In Dorigo M. et al., editors, *Swarm Intelligence—ANTS*, volume 12421 of *LNCS*, 271–281. Springer, Cham, Switzerland.

Salman M., **Ligot A.**, Birattari M. (2019) Concurrent design of control software and configuration of hardware for robot swarms under economic constraints. *PeerJ Computer Science* 5:e221.

Finally, implementations of the optimization-based design methods used in the experiments of Chapter 3, 4, and 5 have been released as open source projects, whose implementations are described in:

[‡]First co-author.

Ligot A., Hasselmann K., Delhaisse B., Garattoni L., Francesca G., and Birattari M. (2017). AutoMoDe and NEAT implementations for the e-puck robot in ARGoS3. Technical Report TR/IRIDIA/2017-002, IRIDIA, Université libre de Bruxelles, Brussels, Belgium.

Hasselmann K., **Ligot A.**, Francesca G., Birattari M. (2018) Reference models for AutoMoDe. Technical Report TR/IRIDIA/2018-002, IRIDIA, Université libre de Bruxelles, Brussels, Belgium.

Acknowledgements

This work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (DEMIURGE Project, grant agreement No 681872) and from Belgium's Wallonia-Brussels Federation through the ARC Advanced Project GbO—Guaranteed by Optimization.

I would like to express my deepest gratitude to Prof. Mauro Birattari. More than a supervisor, Mauro has been a true mentor. His passion, his work ethic, his vision, his attention to details and his audacity have inspired me to redouble the effort put in my work and to become a better researcher. His firm yet kind leadership, his guidance, his respect and his patience have inspired me to become a better person. I am extremely grateful for the many, precious things he thought me throughout the years, and forever indebted for the fascinating and enriching journey he offered me to be a part of. *Grazie mille Mauro!*

I also wish to express my deepest gratitude to my friend, my companion, Ken Hasselmann. This PhD would not have been the same without him. He was there during the lowest of lows, helped me through the frustrating times, and accompanied me during the most tiresome jobs. We joked and laughed a lot, and thanks to him, 'fun' is one of the first words that comes to my mind when thinking back to all these years. *Merci l'ami!*

I wish to thank all members of the IRIDIA laboratory. It was an honor and a privilege to work alongside Prof. Marco Dorigo, Prof. Hugues Bersini and Prof. Thomas Stützle, and a real pleasure to share this experience with all the other researchers. During my Master thesis at IRIDIA, the PhD students at that time (Lorenzo, Gianpiero, Alberto, Federico, Gaëtan, Leslie, Giovanni, Anthony, Volker and Marcolino) intimidated me by their status of researchers; I quickly realized that this intimidation was only the fruit of my imagination as they all warmly welcomed me to what immediately felt like a family, treating me like their peer—the same can be said about Marco, Hugues and Thomas. A special 'thank you' goes to the members of the DEMIURGE project (Ken, Salman, David, Jonas, Miquel, Fernando, Darko and Guillermo): the ideas bounced around each others during our meetings and discussions helped me defining my research and to mature as a researcher. I also wish to thank Prof. Emanuele Garone

and Dr. Andrès Cotorruelo of the Service d'Automatique et d'Analyse des Systèmes (SAAS) for redacting the proof of a key element of this thesis, and Prof. Andrea Roli of the Università di Bologna, Italy, for our fruitful collaborations.

Finally, I wish to thank my family. I am extremely grateful for their support and their investment in my education. They pushed me towards university and insisted I would give it a try when I doubted myself and would have instead settled for a different path. Thank you for giving me the confidence I lacked when I needed it the most.

The final words go to my loving wife, Maria Paula. She always made me believe in myself, she made sure I disconnected from work when necessary, and gave me the occasional much-needed kick in the butt when I lacked motivation. *Te amo.*

Antoine

Contents

Chapter 1

Introduction

The first robot, Unimate, was created in 1959. Unimate is a mechanical arm that quickly became popular in the industry as it was able to repeat tasks that required strength, rapidity, and precision over and over again. The novelty of Unimate with respect to the industrial machines that appeared in the 18th century was the fact that it was digitally programmable: the same machine could be used to accomplish multiple tasks. Unimate could not adapt itself to different tasks, but its behavior could be redefined by a human. It is with advancements in technologies such as sensors, manipulators, transistors, and artificial intelligence that robots gained the ability to sense their environment and act upon it in real time—to become autonomous. Today, robots have not only allowed to completely automate most assembly lines, they also spread beyond industry: they are now used in transportation, agriculture, construction, space exploration, health care, and surgery. Robots, in the form of automatic vacuum machines or lawn mowers, have also discreetly entered many households.

Roboticists have for vocation to create robotic systems that solve problems ever more complex; often to free humans from difficult, unpleasant, or dangerous tasks. Most of the research in robotics has been dedicated to push the capabilities of individual robots; to make them more competent, more intelligent, and more autonomous. In many applications however, a single robot is not appropriate. An example of such applications is search and rescue (e.g., extracting people from a collapsed building after an earthquake or from a sinking ship during a tempest). One can easily imagine that a single robot would have to be extremely agile, rapid, strong, and precise to accomplish alone search-and-rescue missions in hazardous conditions. In addition to the conception of this robot to be exceptionally challenging, this solution would not be viable because the robot represents a single point of failure of the whole rescue system. This solution is also not satisfactory and possibly unfeasible: if the number of imperiled persons exceeds the capabilities of the robot, not all will be saved. A new, better robot should then be devised. In this application, and in many others, the solution is a system composed of multiple robots that operate in parallel.

Swarm robotics is the engineering discipline that studies how groups of autonomous robots can coordinate to solve a problem (Beni, 2004; Şahin, 2004; Dorigo et al., 2014;

Garattoni and Birattari, 2016; Hamann, 2018). In a swarm, robots have local sensing and communication capabilities, and they do not rely on a predefined leader or on external infrastructures. The robots are relatively simple with respect to the mission they must accomplish; they must cooperate to perform tasks that are beyond their individual capabilities. Robot swarms are meant to be robust to individual failure, to be able to operate in areas where communication capabilities are limited (i.e., below the ground or underwater), and to be scalable—that is, the system adapts itself seamlessly to the increase or decrease of robots. These characteristics make for robotics systems that are ideal solutions to problems that involve risks of damage and for which global communication might not be achievable, such as the aforementioned search and rescue.

The literature on swarm robotics provides many successful and fascinating demonstrations of swarms of different robots able to execute collective tasks (Dorigo et al., 2013; Rubenstein et al., 2014; Werfel et al., 2014; Garattoni and Birattari, 2018; Li et al., 2019; Zhou et al., 2022). These successes have propelled the domain to a notable position in the scientific literature, with more and more of these works being published in prestigious journals. Swarm robotics is now recognized as a technology that is likely to have a major impact on robotics in the near future (Yang et al., 2018).

Despite the growing enthusiasm around swarm robotics, the research domain is still in its infancy and robot swarms have yet to be applied to real-world scenarios. What prevents swarm robotics to be of a major impact on robotics *today* is the challenge that represents obtaining the desired collective behavior of a swarm, but most importantly, the lack of a general methodology to do so (Brambilla et al., 2014; Dorigo et al., 2020). The design problem in swarm robotics is indeed particularly difficult as it is not feasible to directly program the desired collective behavior of a swarm: only the behaviors of the individual robots can be programmed. Any collective behavior displayed by a swarm is the result of many interactions between robots, and between robots and the environment. Obtaining that a swarm behaves as desired requires the robots to correctly react to these interactions, which depend on how the system evolves, and are therefore unknown at design time.

A few principled manual design methods have been proposed, but, due to their working hypotheses and constraints, their application is limited to specific classes of missions (Hamann and Wörn, 2008; Berman et al., 2011; Brambilla et al., 2014; Reina et al., 2015). Similarly, traditional multi-robot systems and software engineering techniques, which rely on the formal derivation of the individual behaviors from specifications expressed at the collective level (Brugali, 2007; Di Ruscio et al., 2014; Bozhinoski et al., 2015; Schlegel et al., 2015), cannot be applied to swarm robotics, at least in the general case. Therefore, in most cases, designers proceed by trial and error, and the design process is costly, time consuming, and hardly repeatable.

Optimization-based design is a possible alternative. In this approach, the problem of designing the control software that determines a robot's behavior to perform a given mission is re-formulated into an optimization problem: an optimization algorithm searches for the instance of control software, among a finite space of possible

ones, that maximizes a mission-dependent measure of the collective performance of the swarm—or, at least, for one that scores sufficiently well with respect to the given performance measure. Optimization-based design, because it effectively bypasses the complex endeavor of deriving the individual rules from the desired collective behavior, is deemed to have the potential to become a general engineering methodology to conceive swarm behaviors (Dorigo et al., 2020).

Optimization-based design methods can be categorized according to two different, independent characteristics. The commonly adopted categorization stands on the nature of the design process and distinguishes between on-line methods and off-line ones. In on-line methods, the design process takes place after the robots are deployed in the target environment. A distributed optimization algorithm, running on the robots themselves, iteratively improves the control software on the basis of their performance while the robots operate on their mission in the target environment. In off-line methods, the design process takes place prior to the deployment, and it (most typically) relies on evaluations of control software in simulations that ought to reproduce relevant features of the target environment in which the robot will eventually operate.[1]

A second classification exists; it stands on the way a optimization-based method is utilized, and it distinguishes between semi-automatic methods and fully-automatic ones (Birattari et al., 2019, 2020). In semi-automatic design, the design method is used as a tool by a human expert who is allowed to adapt any part of the design process to the mission at hand so that the solution produced satisfies the requirements and the expectations. The adaptation of the process is typically performed after the expert evaluates the behavior of the swarm when executing control software produced, and a new design must then be initiated. Examples of elements of the design process that can be adjusted are the control software architecture, the parameters of the optimization algorithm, the simulation models, and the performance measure to be maximized. In fully-automatic design, the design method does not undergo any per-mission, manual modifications. For a design method to qualify as fully-automatic, one should conduct research reflecting the following tenets: the method should not be defined to solve a specific mission, and its expected performance should be assessed on a class of missions

[1] In principle, one can do off-line design with physical robots and therefore without the need for simulations—see, for example, the works of Regan et al. (2006) and Gongora et al. (2009). In fact, one can upload and execute on the robots the candidate instances of control software that are generated by the optimization algorithm that is itself running on a dedicated machine, and feed the observed performance back to the algorithm. However, this process is extremely time consuming given the several tens of thousands of evaluations commonly required to obtained the desired collective behaviors (Regan et al. (2006) and Gongora et al. (2009) adopted this process, but only made use of 600 and 3000 evaluations, respectively, and for single robots—doing so for swarms of robots would be considerably more time consuming). This design process would also be expensive and potentially dangerous as harmful behaviors (for the robots and/or the environment) are likely to be uploaded on the robots, especially in the early phases of the design process, if no damage prevention strategies are manually included (Jones et al., 2019; Kaiser and Hamann, 2022; Wahby and Hamann, 2015). The real interest of going off-line is to use simulations, which are (relatively) cheap, safe, and convenient. For these reasons, it comes natural to identify off-line design with simulation.

without providing any human intervention or modification to any phase of the design processes initialized for each mission of the class.

The two classifications can be crossed to give four categories of approaches: on-line semi-automatic, on-line fully-automatic, off-line semi-automatic, and off-line fully-automatic. It should be noted that the aforementioned classifications of approaches are not intended to be a rigid taxonomy. In fact, hybrids between on-line and off-line approaches are possible: one could conceive a design method that consists in the off-line conception of an instance of control software for which some parameters are then adjusted on-line, while the swarm operates. Because the semi-automatic/fully-automatic classification is based on the fashion a method is used, a same method could possibly be utilized sometimes in a semi-automatic way and sometimes in a fully-automatic one. We therefore expect achievements and progress in one approach to have an impact on the other, and that both will grow at a somewhat similar pace. Nonetheless, these classifications are conceptually important because they provide the categories to navigate the literature and the possible approaches proposed. They are also essential to the definition of the research questions and challenges relevant to each approach, as well as to the definition of the appropriate expectations of what they should achieve.

So far, off-line semi-automatic and fully-automatic approaches have received the most attention from the community. On-line methods that could possibly or have been demonstrated to operate directly on robot hardware have been proposed, but they do not appear to be the ultimate solution the design problem of robot swarms. Indeed, the relative small search space that can be explored, the risk of damaging robots and/or environments by suboptimal control software, and the fact that it can address only missions in which the robots can evaluate their own collective performance, limit their applications. Designing control software off-line on the basis of simulations does not suffer from the these drawbacks, and therefore appears to be a more effective and viable approach than on-line design.

Yet, for off-line optimization-based design to become a general methodology to conceive robot swarms, notorious issues that hinder its progress need to be overcome. To the best of our knowledge, Francesca and Birattari (2016) were the first to warn the community about the lack of a systematically applied empirical practice in the optimization-based design of robot swarms. In fact, in their review of the relevant literature, the authors highlighted the fact that it is composed of isolated studies, and identified a few issues that need to be addressed. Like the authors, we strongly believe that establishing a clear state of the art is the next crucial step that must be taken by the community for optimization-based design to become a mature research domain. This thesis is dedicated to providing tools that we deem necessary to make this step.

One of the issues highlighted by Francesca and Birattari (2016) is the absence of benchmarks on the basis of which the community should assess design methods. We go further in our critique towards the domain literature and show—both intuitively and formally—that none of the experimental protocols employed in the current pa-

pers respect the tenets of fully-automatic design. We propose one that does. Our experimental protocol is characterized by two elements: (i) the notion of mission generator that allows for the creation of benchmarks of missions that mimic those a design method will eventually have to solve, and (ii) a sampling strategy that minimizes the variance when estimating the expected performance of design methods. We illustrate the experimental protocol by comparing the performance of two off-line optimization-based design methods on the basis of 30 missions that were automatically generated.

Another issue in the current literature on optimization-based design highlighted by Francesca and Birattari (2016) is the lack of experiments performed with physical robots. Robot experiments are expensive and time consuming. The financial costs of running robot experiments include the price of the individual robots themselves, but also the cost of maintenance of the robots (including required tools, spare parts, and eventual shipping and repair services) and the cost of the equipment needed to operate robot swarms (including laboratory space, batteries and charging stations, and tracking cameras and desktop stations to record the positions of the robots and compute performance metrics). The time consuming aspect of running robot experiments includes the calibration of the robots and the setup of the environment, the supervision and monitoring of the execution of the swarm by at least one researcher, and the maintenance of the robots. These elements represent a heavy burden to the assessment of control software on physical robots, and, to avoid this burden, many studies skip robot experiments and only rely on the simulation model used during the design to evaluate automatically generated control software (Gomes and Christensen, 2018; Salman et al., 2019; Pagliuca and Nolfi, 2019; Cambier and Ferrante, 2022). By doing so, they overlook the most important and well-known problem faced by off-line design: the *reality gap* (Brooks, 1991; Jakobi et al., 1995; Silva et al., 2016; Hasselmann et al., 2021).

The reality gap is the difference between the simulations on the basis of which control software is produced, and the real environment in which the control software is eventually executed. These differences between simulation and reality might be subtle, but they are unavoidable (Brooks, 1992; Jakobi et al., 1995). Due to the reality gap, it is likely that robot swarms do not display the same behavior in simulation and in reality (Floreano et al., 2008). These behavioral discrepancies often translate in control software that performs poorly in reality despite giving good results in simulation (Hasselmann et al., 2021). The true issue with the reality gap is the relative nature of its effect: different instances of control software are prone to be affected by different degrees of performance drop when moving from simulation to reality. The relative nature of the reality gap might lead to a phenomenon of *rank inversion*: a phenomenon in which, out a pair of instances of control software, the one that performs best in simulation performs worst in reality. The possibility of observing a rank inversion is insidious because it discredits any conclusion made on the basis of results obtained on the simulation model used during the design. More importantly, it questions the validity of the off-line design process itself in that it relies on the assumption

that the higher the performance in simulation, the higher the performance in reality.

Several approaches have been proposed to limit the performance drops caused by the reality gap as much as possible, but none of them has been studied in detail, no extensive comparison has been produced, and none of them appears to be the ultimate solution. In the literature, effects of the reality gap are commonly explained by saying that reality is more complex than simulations—or equivalently, that simulations are too simplistic (Nolfi et al., 1994; Koos et al., 2013). We call this assumption the *complexity assumption*, and we challenge it. Our working hypothesis is that the reality-gap problem is somehow reminiscent of the generalization problem faced in machine learning, and that drops in performance from simulation to reality are caused by a sort of overfitting of the control software to the simulation model used in the design. In other words, we consider performance drops to be due to the inability of the control software to generalize to different contexts/conditions of executions. Following this working hypothesis, we contend that performance drops can be observed even if the simulation model under which control software is designed is not a simplistic version of the context/conditions under which the control software is eventually assessed; they only need to be sufficiently different.

We support our contention with a set of simulation-only experiments in which we create an artificial reality gap: we replace reality with a simulation model into which we insert discrepancies with respect to the model used in the design. We shall call any secondary model that we use for assessing control software—and that therefore plays the role of reality—a *pseudo-reality*. For these experiments, we created a pseudo-reality via trial and error with the goal of observing a performance drop that is qualitatively similar to the one previously observed on physical robots (Francesca et al., 2014b). Our experiments have the logical structure of a *reductio ad absurdum*: traversing the artificial reality gap we created in both directions—that is, generating control software on the basis of one of the two models considered and evaluating it on the other one, and vice versa—leads to similar effects of the (pseudo-)reality gap which, according to the complexity assumption, would mean that one of the two models used is simultaneously more and less complex than the other one; a contradiction. Our experiments therefore bring empirical evidence that the effects of the reality gap appear even in cases in which we can exclude that the evaluation is performed in a context that is more complex than the one in which control software is designed.

Beside shedding light on the nature of the reality gap, the fact that we were able to (re)produce a realistic performance drop with an artificial reality gap suggests that the concept of pseudo-reality could have practical implications. As discussed above, the reality-gap problem currently entails the necessity to conduct expensive and time consuming tests on physical robots in order to reliably assess control software; a simulation-only procedure to accurately predict real-world performance would be extremely valuable. We conceive various predictors on the basis of the concept of pseudo-reality, and compare them with the classical approach adopted to estimate real-world performance, which relies on the evaluation of control software on the sim-

ulation model used during the design. Results show that the pseudo-reality predictors considered yield a more accurate estimation of the real-world performance than the classical approach.

Overall, we believe that the two main contributions of this thesis—that is, the experimental protocol and the pseudo-reality predictors—will have a positive impact on the research domain of optimization-based design of robots swarms. On the one hand, the experimental protocol is a significant step towards addressing the current lack of objective comparisons of design methods and, consequently, the lack of a clearly identified state of the art. On the other hand, we foresee that the pseudo-reality predictors could considerably reduce the amount of tests with physical robots needed to validate ideas and new design methods, and would therefore facilitate the research in the off-line design of robot swarms. Together, the tools we discuss here enable researchers to answer the following question more appropriately and with more confidence than with the current practice: *Which off-line fully-automatic design method produces control software that will yield the best performance once executed on a swarm of physical robots?*[2]

The following is the structure of the thesis. In Chapter 2, we give an overview of the domain of swarm robotics, and review the literature that is dedicated to optimization-based design and to the reality-gap problem. In particular, we further discuss the two categorizations of the approaches to optimization-based design—that is, off-line versus on-line, and semi-automatic versus fully-automatic—and we classify the relevant literature according to the four categories that result from crossing these categorizations. We pay particular attention to the elements of the existing works that indicate whether they belong to the semi-automatic or to the fully-automatic approach, and to the experimental protocol employed. We also describe the functioning of the methods proposed to handle the reality gap and propose a novel taxonomy.

In Chapter 3, we propose an experimental protocol for the comparison of fully-automatic design methods. This protocol is characterized by two notable elements: a generator of missions to define benchmarks for the evaluation and comparison of design methods, and a sampling strategy that minimizes the variance when estimating their expected performance. We illustrate the experimental protocol by comparing the performance of two off-line fully-automatic design methods on 30 generated missions.

In Chapter 4, we disprove the aforementioned *complexity assumption*. We do so

[2]In this thesis, we focus on off-line fully-automatic design of control software for robot swarms. However, the contribution has also the potential to have an impact on on-line fully-automatic, on off-line semi-automatic, and on manual design. As the experimental protocol we propose aims at estimating the expected performance of a design method following the tenets of fully-automatic design, it is appropriate for both off-line and on-line ones, whereas the pseudo-reality predictors might facilitate the tedious and time-consuming process that represents the conception of control software via an off-line semi-automatic method. Also, human designers often use simulations to conceive control software—being able to test their solutions in pseudo-reality rather than on physical robots would also simplify their endeavor. Moreover, we believe that the contribution is also relevant to the optimization-based design of single-robot systems or of centralized multi-robot ones as they face issues that are similar in nature to the ones we face in swarm robotics.

via a simulation-only experiment organized in two stages. In the first stage, we automatically generate control software on the basis of M_A, a model that has been used in previous studies for the same purpose (Francesca et al., 2014b, 2015; Birattari et al., 2016). We evaluate the control software on the basis of another model, M_B, that we chose via trial and error so as to qualitatively replicate real-world performance obtained by two design methods on two missions (Francesca et al., 2014b). In the second stage, we interchange the role played by the two models: we use M_B as design model to generate control software, and we use M_A as pseudo-reality to assess it. Finally, we evaluate the control software produced on the basis of both M_A and M_B on physical robots to investigate whether they have an impact on the performance in reality.

In Chapter 5, we use multiple pseudo-reality models that we sample automatically, and we replicate the two-stage simulation-only experiment of Chapter 4. We propose multiple measures to quantify the difference between the design model and the pseudo-reality one—we refer to the difference between these two models as the width of the artificial reality gap they create. We study the correlation between the performance drop experienced due to a given artificial reality gap and the width of that gap.

In Chapter 6, we quantitatively assess the accuracy of multiple predictors of real-world performance of robot swarms. We consider (i) the design model M_A, (ii) the pseudo-reality model M_B found via trial and error, and (iii) the automatically sampled pseudo-reality models. To assess these predictors, we reuse 1021 instances of control software generated by 18 off-line design methods for 45 missions and their associated real-world performance that we collected from 7 previously published studies.

In Chapter 7, we conclude the thesis with a summary of the contributions and a discussion on future research directions.

Chapter 2

State of the art

This chapter gives an overview of the swarm robotics literature. A number of works have illustrated the principles of swarm robotics via fascinating demonstrations. For example, Dorigo et al. (2013) presented a swarm of robots of different capabilities that coordinate to localize, grab, and retrieve a book from a shelf. Rubenstein et al. (2014) presented a swarm of a thousand coin-sized robots able to self-organize into different complex shapes. Werfel et al. (2014) presented a swarm of robots able to carry foam bricks and coordinate to build castles, towers, and pyramids. Garattoni and Birattari (2018) presented a swarm of robots able to collectively determine a sequence of actions whose order is a priori unknown. Li et al. (2019) presented a swarm able to collectively move and push objects despite the fact that the individuals robots are themselves incapable of locomotion. Very recently, Zhou et al. (2022) presented a swarm of aerial robots able to fly through a dense forest while maintaining a formation or tracking a target. These works are particularly impressive because there is currently no methodology to follow to create these swarms; they are the products of ingenuity, expertise, and countless hours of labor.

Optimization-based design is a promising solution to the current lack of general methodology for the conception of robot swarms, and we focus our attention to this approach in this chapter. We refer the reader to Hamann (2018) for an in-depth introduction to the research domain of swarm robotics, to Brambilla et al. (2013) and Garattoni and Birattari (2016) for comprehensive reviews of swarm robotics from an engineering perspective, and to Schranz et al. (2020) for a recent survey of the swarm behaviors developed so far. Finally, we refer the reader to Dorigo et al. (2021) for a recent authoritative perspective on the future of swarm robotics.

2.1 Swarm robotics

Swarm robotics is an engineering discipline that originally found inspiration in swarm intelligence (Dorigo and Birattari, 2007; Dorigo et al., 2014), and whose ultimate goal is to design large groups of autonomous robots (Beni, 2004; Şahin, 2004; Hamann,

2018)—what we call *robot swarms*.

In a swarm, robots are completely autonomous and act on the basis of the principle of locality: they take decisions based solely on local information collected through their own sensors, or on information communicated by their neighboring peers. Local sensing and communication capabilities promote *scalability*, the ability to perform a task in different group sizes. In fact, as the robots are only aware of their nearby surroundings, they are oblivious of what the whole swarm is doing or of the number of robots in the swarm. Therefore, the addition or removal of robots does not influence the behavior of the system granted that it does not affect the robot density too dramatically.

Robot swarms typically tackle missions that are beyond the capabilities of the individual robots that compose the swarm; they therefore rely on cooperation and coordination to reach their goal. Missions are often divided into sub-tasks that the robots perform in parallel, switching from one sub-task to another depending on contingencies. Cooperation, parallel execution, and autonomous task allocation promote *flexibility*, the ability to cope with variations in the environment and to handle a large variety of missions.

A robot swarm behaves in a self-organized way: robots do not have predefined roles, nor is there a predefined leader that directs the others. No single robot is therefore indispensable to the success of the swarm. In addition, swarms are typically characterized by a high redundancy: many robots can perform any given sub-task. Autonomy, self-organization, and redundancy promote *fault-tolerance*, the ability to cope with damage or loss of individual robots.

The characteristics and properties of swarm robotics systems are particularly appealing for applications that involve a high risk of damage, that are subject to environmental and conditional changes, and that take place in areas where deploying a communication infrastructure is unfeasible. Common examples of applications are search-and-rescue in hazardous areas, surveillance, underwater/underground/exoplanet exploration or monitoring, demining, and toxic-waste disposal.

However, these characteristics and properties make for complex systems that are difficult to design. The design problem in swarm robotics is particularly challenging as it is not feasible to directly program the collective behavior of a swarm: only the individual-robot behaviors can be specified. Any collective behavior displayed by a swarm is the result of interactions between robots, and between robots and the environment. Obtaining the desired collective behavior requires therefore to master the complex "what, where, when, and how" of these many robot-robot and robot-environment interactions, which are unknown at design time.

Traditional multi-robot systems and software engineering techniques (Brugali, 2007; Di Ruscio et al., 2014; Bozhinoski et al., 2015; Schlegel et al., 2015), which rely on the formal derivation of the individual behaviors from specifications expressed at the collective level, cannot be applied to swarm robotics, at least in the general case, due to the aforementioned issues. A few principled manual design methods have been pro-

posed (Hamann and Wörn, 2008; Kazadi, 2009; Berman et al., 2011; Beal et al., 2012; Brambilla et al., 2014; Reina et al., 2015; Pinciroli and Beltrame, 2016), but their application is limited to specific classes of missions due to their working hypotheses and constraints. Therefore, experts in swarm robotics usually proceed by trial and error to obtain the desired collective behaviors (Garattoni and Birattari, 2016).

2.2 Optimization-based design

A promising alternative to manually designing the control software exists: the adoption of optimization-based design methods. With these methods, the design problem is reformulated into an optimization problem: an optimization algorithm explores the search space composed of all possible individual behaviors, with the objective of finding one that maximizes a performance measure expressed at the collective level. The optimization-based approach regroups different categories of design methods. In the domain literature, the commonly adopted classification distinguishes between on-line and off-line methods (Brambilla et al., 2013; Francesca and Birattari, 2016; Bredeche et al., 2018). A second classification distinguishes between (fully-)automatic and semi-automatic design methods (Birattari et al., 2019, 2020).

The on-line/off-line classification is widely used and understood by the community. For this reason, in Sections 2.2.1 and 2.2.2 we only focus on describing the two approaches and the main methods used. We refer the readers to recent reviews of the literature for more details about the applications and achievements of these methods (Francesca and Birattari, 2016; Bredeche et al., 2018). On the contrary, the classification between automatic and semi-automatic is very recent and deserves more attention.

The on-line/off-line and semi-automatic/automatic classifications are based on unrelated criteria and can be crossed to give four categories of approaches to the optimization-based design: semi-automatic on-line, semi-automatic off-line, automatic on-line, and automatic off-line. In Section 2.2.3, we review the literature on optimization-based design for which ideas and methods have been tested on physical robots and classify them according to these four categories.

2.2.1 On-line and off-line design

In on-line methods, the design process is distributed and operates on the physical robots while they perform their mission. The main appeal of on-line design methods is the autonomous and continuous adaptation of the system. In fact, the robots are able to modify their behaviors on the fly to cope with possibly unexpected circumstances, which makes the swarm intrinsically flexible. Another attractive aspect of on-line design is the fact that is does not require the modeling of the robot-robot and robot-environment interactions; on-line design methods are hence not subject to the reality gap problem like their off-line counterpart.

In on-line design, the search for the best possible instance of control software is distributed to all robots in the swarm. Each robot conducts its own exploration of the search space by executing a subset of instances of control software one after another for a fixed or variable amount of time. The process is initialized with a random subset of instances of control software. The robots evaluate their quality by assessing their own individual performance with respect to the mission at hand. The subset of instances is updated periodically according to rules that are proper to the optimization algorithm employed. When sufficiently close to one another, robots might exchange information about the optimization process their are conducting, such as the best instance found so far and its associated performance. Upon reception of instances evaluated by peers, robots might decide to incorporate them into their own subset of instances, possibly after some mutations.

Among the notable works belonging to the on-line optimization-based literature, the first is ascribed to Parker (1997) for L-ALLIANCE, a control architecture that allows for the on-line adaptation of some parameters. The most popular approach is called *embodied evolution* (Watson et al., 1999, 2002) and consists in the embodied optimization of artificial neural networks parameters (i.e., the synaptic weights) via an evolutionary algorithm. Recent embodied evolution frameworks include MEDEA (Bredeche et al., 2012), MONEE (Haasdijk et al., 2014), and odNEAT (Silva et al., 2015); the last one allows for the on-line evolution of the topology of the neural networks alongside the optimization of the synaptic weights. Methods that slightly distinguish themselves from the classical embodied evolution exist: König et al. (2009) optimized finite-state machines using a distributed evolutionary algorithm, and Di Mario and Martinoli (2014); Di Mario et al. (2015) optimized recurrent neural networks using a distributed particle swarm optimization algorithm.

On-line methods suffer from a number of drawbacks (Francesca and Birattari, 2016; Bredeche et al., 2018). In the early steps of the design process, the instances of control software executed are typically selected randomly. These instances are likely to be suboptimal, and might cause damage to the robots or the environment if no damage prevention strategies are manually included (Jones et al., 2019; Kaiser and Hamann, 2022; Wahby and Hamann, 2015). Moreover, only a small search space is likely to be explored so as to avoid wasting time and energy executing large samples of suboptimal instances. Finally, enabling the individual robots to assess the collective performance of the swarm is not always feasible, which limits the application of on-line design. Let us imagine a foraging mission in which a swarm of robots has to collect items from a source and gather them to a location we shall call the nest. If the swarm density is reasonably low, it is very likely that all robots are not able to know the total amount of items collected to the nest by the swarm in real time. To the best of our knowledge, all applications of on-line design use the performance of the individual robots to assess the performance of the instance of control software executed at a given time. In our foraging example, each robot would therefore aim at maximizing the number of items they retrieved rather than the number of items collected collectively. This

naturally promotes individualism, and cooperative behavior such as chain-based path formation (Nouyan et al., 2008), in which some robots stand still to indicate to others where the points of interests are located to increase efficiency, would never emerge.

In off-line methods, the design process is performed before the swarm is deployed in the target environment. To evaluate the performance of control software, these methods rely on computer-based simulations that reproduce relevant features of the target environment in which the robot will be eventually deployed. Because the assessment of performance is much faster and cheaper in simulation than on physical robots, off-line methods can explore a larger space of possible instances of control software in a relatively short time. In addition, simulation provides a god's-eye view of the swarm, which allows off-line methods to evaluate any possible performance metric. Finally, damaging robots or the environment is irrelevant when they are simulated entities.

Many off-line optimization-based methods have been presented in the literature. They resemble the ones studied in the on-line approach, the major difference being the centralization aspect of the optimization process. The most popular approach is neuroevolutionary (swarm) robotics (Lipson, 2005; Floreano et al., 2008; Trianni, 2008, 2014): robots are controlled by a neural network, the robot's sensor readings are fed to the neural network as inputs and the robot's actuator values are dictated by the network's output. An evolutionary algorithm is used to search for the best possible configuration of the neural network, that is, the parameters or synaptic weights, and possibly the topology. Other approaches, based on modularity, have been proposed: they generate control software by assembling low-level behavioral modules. Modules can be generated automatically (i.e., via neuroevolution) and combined manually (Duarte et al., 2015); conceived manually and combined automatically (Francesca et al., 2014b); or both generated and combined automatically (Duarte et al., 2014; Ligot et al., 2020a).

2.2.2 Semi-automatic and fully-automatic design

In semi-automatic design, a human designer utilizes an optimization algorithm as a tool that they operate using their intuition and previous experience to solve the problem at hand. Typically, the designer iterates through a series of steps, which include: the execution of the optimization process, the evaluation and analysis of the behavior produced using simulation and/or physical robot experiments, and the modification of the optimization process. This three-step procedure is repeated until the control software produced satisfies the designer and/or they feel that it cannot be improved any further.

In automatic design, on the other hand, the design method does not undergo any per-mission manual modifications.

Due to the difference in operational functioning between semi-automatic and fully-automatic design, they address different contexts of application—see Figure 2.1 for an illustration.

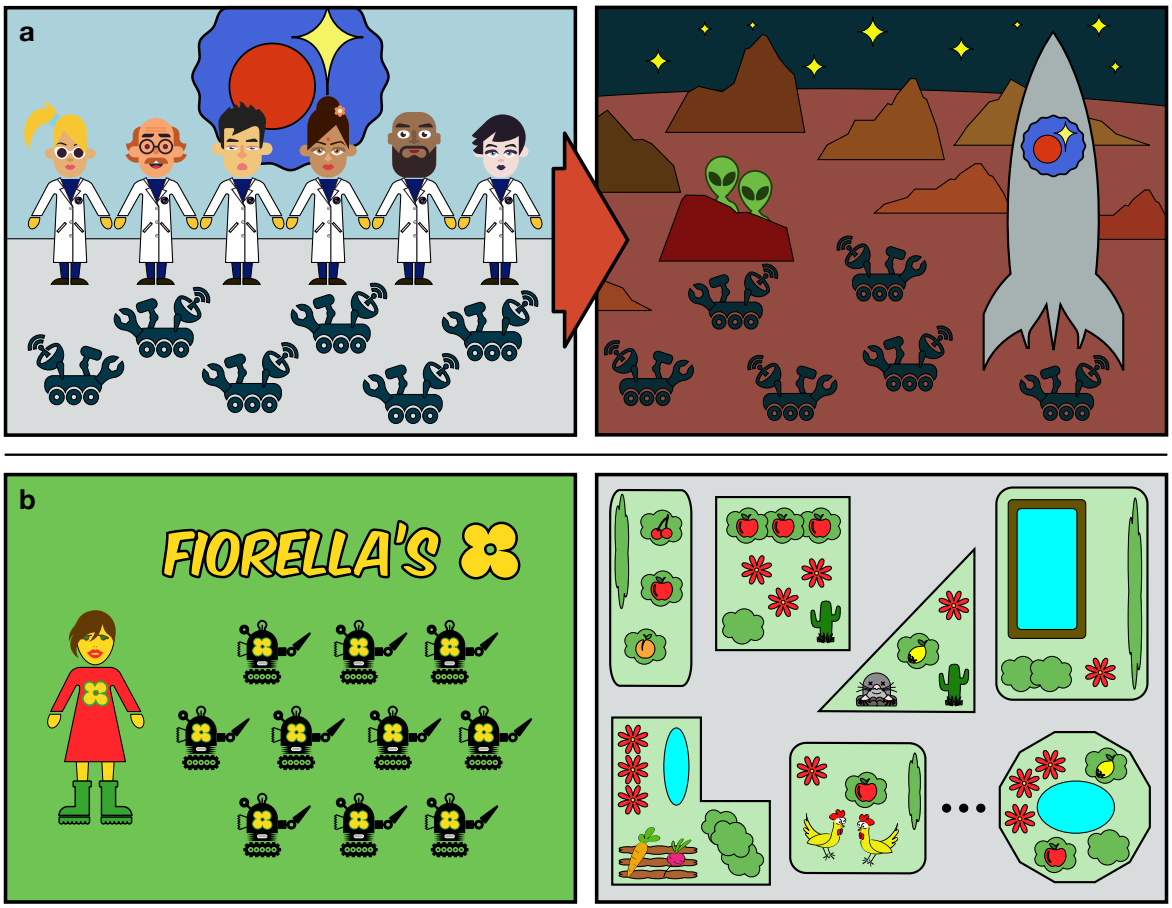


Figure created by Mauro Birattari to illustrate the content of
Birattari M, Ligot A, Hasselmann K (2020)
Nature Machine Intelligence 2(9):494–499, DOI 10.1038/s42256-020-0215-0

Figure 2.1: **Illustration of potential applications of semi-automatic and automatic design of robot swarms.** (**a**) Illustration for the semi-automatic approach inspired from the NASA's tests in the Atacama desert (Tabor, 2017). Consider a project aiming at deploying a swarm-based exploration and monitoring system on Mars. The mission is complex, the challenges are legion. One can imagine that many engineers are involved, and that sufficient time and resources are available for them to be able to test, adjust, and repeat the design process multiple times until the appropriate behavior is found. The multiple tests can be done in simulation, or in mock-up environments that reproduce the conditions that the robots will experience once deployed on Mars. (**b**) Illustration for the automatic approach inspired from the Fiorella's gardening swarm (Birattari et al., 2019). Consider a small one-person business that uses a swarm to provide a service, here a gardening one. Everyday, the swarm is deployed on different gardens. Each intervention is relatively simple, yet they are all unique and benefit from a tailored design to maximize efficiency. Time and monetary constraints do not allow for tests and manual interventions; Fiorella's business depends on a fully-automatic design method.

Semi-automatic design is best suited for complex, one-of-a-kind missions. To successfully accomplish such missions, it is reasonable to expect that sufficient time and resources can be allotted to allow human designers to iteratively adjust the functioning of a design method on the basis of evaluations of control software produced. The relative high cost of this human-in-the-loop process is justified by the exceptional nature of the missions to be solved.

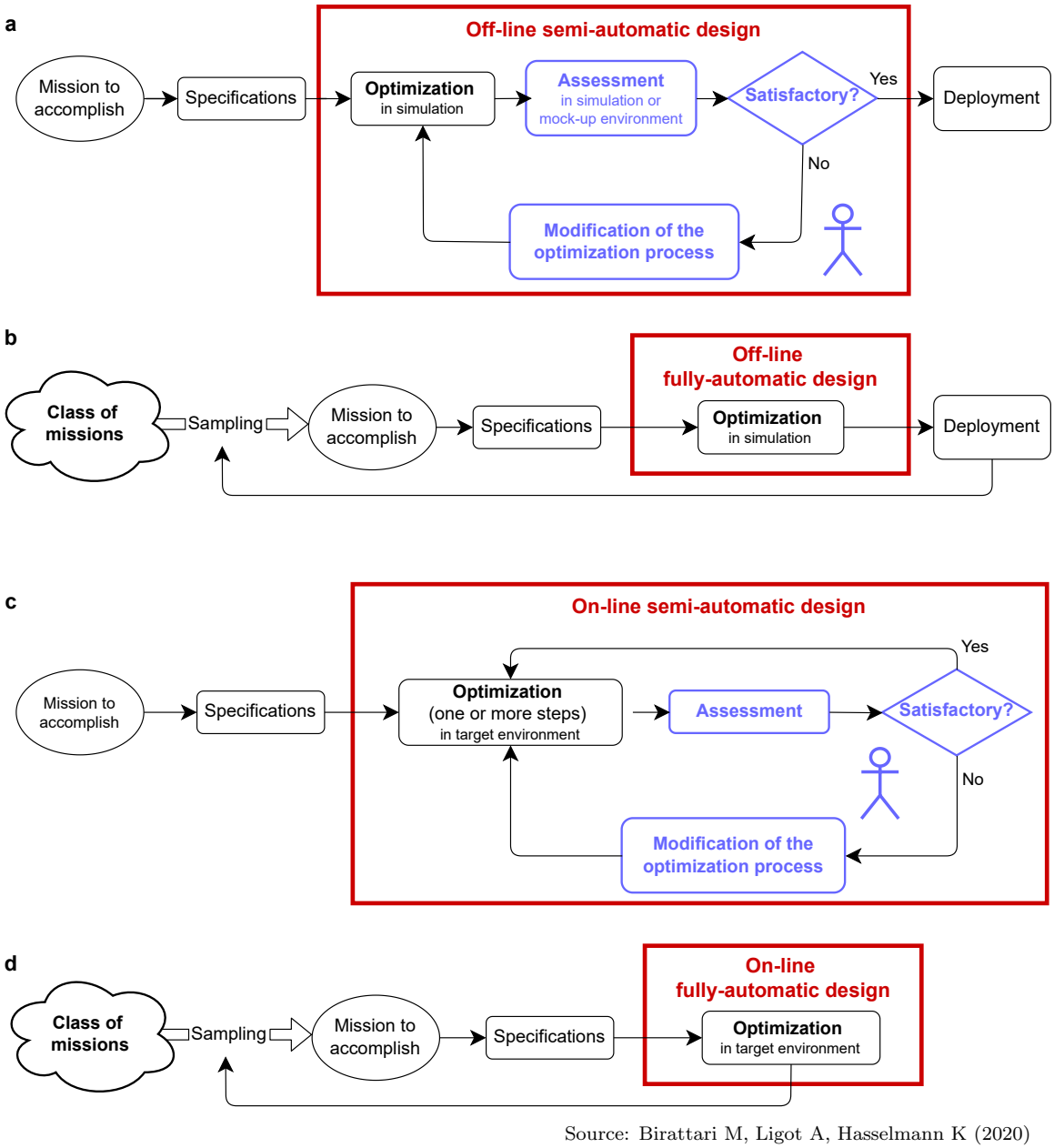
Fully-automatic is best suited when one must solve multiple missions repeatedly, one after another, in such a way that the presence of a human expert in the design loop would be unfeasible due to monetary and time constraints. In this fully-automatic context, a design method is therefore expected to be able to design control software for any mission belonging to a given *class of mission* without any intervention of a human designer on a per-mission basis. By class of missions, we mean a set of missions characterized by different goals, constraints, and/or configurations. Differences between two missions of a class might be qualified as minor, yet they can be sufficiently important to benefit from a tailored design.

2.2.3 Four crossed categories

Crossing the on-line/off-line and semi-automatic/automatic classifications gives four possible categories of approaches: off-line semi-automatic, on-line semi-automatic, off-line automatic, and on-line automatic. They are depicted in Figure 2.2. In this section, we review the existing works in optimization-based design, and we classify them according to these four categories—a summary of this categorization is given in Table 2.1. The distinction between on-line and off-line is straightforward as it depends on the nature of the design method itself. On the other hand, the distinction between semi-automatic and fully-automatic can be ambiguous as it depends on the way the design methods are used. A same design method can therefore be used in an semi-automatic or in a fully-automatic fashion.

Our review reveals that a majority of the works presented in the literature on neuroevolutionary swarm robotics belong in the semi-automatic approach. The authors of these works usually do not describe how they devised the neuroevolutionary setup they use (e.g., the configuration of the neural networks). However, it is commonly believed that many evolutionary setups are found in an ad-hoc fashion (Christensen and Dorigo (2006) and Doncieux et al. (2015) mention it explicitly in their respective papers), and we highlight elements of these works that confirm this belief. The element of the design process that is most often adapted manually after observing the executions of behaviors produced in preliminary runs is the performance measure[1] to be optimized (Floreano and Urzelai, 2000; Doncieux and Mouret, 2014). Typically, terms are added to reward the emergence of certain behaviors, to penalize those that exploit aspects of the simulation that are not correctly modeled and that therefore do

[1]This performance measure is usually referred to as *fitness function* in the neuroevolutionary jargon.



Source: Birattari M, Ligot A, Hasselmann K (2020)

Figure 2.2: **Flowcharts of the four approaches to the optimization-based design of robot swarms.** The design processes are the parts contained in the red boxes; the human interventions are depicted in blue.

not cross the reality gap satisfactorily, or to speed up the convergence of the optimization process. Besides the performance measure, some works report the need to adjust parameters of the simulations, to adjust the output of the control software prior to assigning it to actuators, to fine-tune parameters of the optimization algorithm, or to adapt the configuration of the swarm to obtain a collective behavior that successfully accomplish the mission they aim to solve.

In addition to the design process, we also pay attention to the protocols used during the experiments described in the works belonging to the off-line optimization-based design of robot swarms. An experimental protocol in off-line design consists in the missions to be solved; the design methods used to solve them; the number of executions of each of these design methods, with each execution typically resulting in the production of one instance of control software; and the number of execution of the produced instances of control software on the robots. The experimental protocols described in works we classify as semi-automatic should give the reader the feeling that the authors are indeed interested in solving a specific mission or to show that a given methodology is able to do so, and that they are not interested in estimating the expected performance of the method.

We mentioned above the possible ambiguity when classifying works to be either considered as belonging to the semi-automatic or fully-automatic approach. As a matter of fact, we were not able to confidently classify several works. These works include off-line design (Baldassarre et al., 2007; Hauert et al., 2009; Trianni and Nolfi, 2009; Gauci et al., 2014a,b) and on-line design (Watson et al., 2002; Usui and Arita, 2003; Di Mario and Martinoli, 2014), and we placed them in a gray area in Table 2.1. The incertitude lies in the fact that despite the authors do not describe any human intervention within the design process, which by itself would lean towards a classification as fully-automatic, the methods described were tested on a single mission, which does not provide evidence that they could be employed to generate satisfactory solutions for other missions without the need to undergo per-mission, manual adaptations (that is, in a fully-automatic fashion).

Off-line semi-automatic design

Quinn et al. (2003) were the first to use neuroevolutionary robotics to produce control software for a robot swarm. They generated a formation-movement behavior in which 3 robots move as a group for a distance of 1 meter in any direction. To shape the behavior they want to obtain, the authors use a performance measure that contains a term that penalizes robots when they go outside of sensor range and one that penalizes robots collisions. The authors instantiated 10 design processes; they terminated 4 of them early as they judged them to be unpromising, the 6 others were terminated when they did not show any sign of possible further improvement. The 6 resulting instances of control software were tested in 5000 simulated runs; the highest-scoring one was then evaluated in 100 trials on robots. There was, according to the authors, no evidence of degradation of the performance between simulated runs and physical

Table 2.1: **Classification of the works in optimization-based design of robot swarms**. The gray area contains works for which no sufficient evidence allows us to confidently place them in either category.

	Off-line	On-line
Semi-automatic	Quinn et al. (2003) Dorigo et al. (2003) Ampatzis et al. (2006) Christensen and Dorigo (2006) Trianni (2008) Ampatzis et al. (2009) Duarte et al. (2016) Jones et al. (2018)	Bredeche et al. (2012) Jones et al. (2019)
	Baldassarre et al. (2007) Hauert et al. (2009) Trianni and Nolfi (2009) Gauci et al. (2014a) Gauci et al. (2014b)	Watson et al. (2002) Usui and Arita (2003) Di Mario and Martinoli (2014)
Fully-automatic	Waibel et al. (2009) Francesca et al. (2014b) Francesca et al. (2015) Hasselmann et al. (2018b) Kuckling et al. (2018) Garzón Ramos and Birattari (2020) Ligot et al. (2020a) Spaey et al. (2020) Hasselmann et al. (2021)	

ones, but no quantitative results were reported. The 5 other instances were evaluated on physical robots as well, although not in the same fashion as the highest-scoring one, and no details is given about the procedure. The authors report that 2 of these instances transferred well to the robots, the 3 others displayed noticeable performance drop.

Dorigo et al. (2003) used neuroevolution to obtain aggregation and coordinated motion behaviors. The authors used a different neuroevolutionary setup for each task: for aggregation, neural networks have 12 sensory neurons connected; for coordinated motion, neural networks have 5 sensory neurons. They added a term to the fitness function devised to evolve aggregation behaviors to prevent the robots from turning on the spot, a behavior that the authors judged to be undesirable. They instantiated 20 design processes resulting in 20 instances of control software, which they evaluated on two simulation models: a 'simple' one that was used during the design, a more

'detailed' one specifically made for testing. According to the authors, the produced control software was robust to the differences between the two simulation models; the performance drop between the two is described as tolerable.[2]

Ampatzis et al. (2006) used neuroevolution to generate control software for two robots to perform phototaxis and get as close to a light source as possible. Two environments are considered: one allows the robot to reach the light source, the other does not. During the design process, each candidate solution is evaluated 15 times: 12 times on the environment that allows to robot to reach the light source, 3 times on the one that does not. The design choice originates from a previous research in which the authors studied the impact of the distribution of the two environments during the design on the resulting performance (Tuci et al., 2004). The fitness function comports a term that rewards the proper use of communication between the robots. 10 design processes were performed, resulting in 10 instances of control software. Only 3 of these instances performed the task successfully. One of them was then ported to physical robots for 40 evaluation trials; 20 on each environment. To select the instance of control software to be ported to the robots, the authors report having performed various tests on robots and made their decision on the basis of what they considered to be 'good' sensory-motor coordination and 'effective' communication.

Christensen and Dorigo (2006) used neuroevolution to generate a phototaxis behavior with hole-avoidance in a group of physically connected robots. Contrarily to the vast majority of works in neuroevolutionary robotics, the authors are very explicit in their description of what elements of the evolutionary setup they sequentially tested to eventually reach one that produced satisfactory results. For example, they report that without incorporating in the fitness function a term that explicitly penalizes robots for falling into a hole, the results were disappointing. The authors also report that they added a term that promotes the minimization of the traction between the robots, which they previously identified to be a determining factor to obtain coordinated motion (Trianni et al., 2004). Experiments with 3 connected robots were performed with the best instances of control software generated. However, no details about how these instances were selected are given, and the authors only state that the robots performed the task successfully without reporting quantitative results.

Trianni and Dorigo (2006) used neuroevolutionary robotics to generate hole-avoidance behaviors in physically connected robots. The authors studied and compared three approaches to communication between the robots. The three different communication strategies are obtained via different evolutionary setups, namely different neural network configurations. The fitness function (the same for the three communication strategies) is composed of different terms, one of which promotes coordinated motion by minimizing traction between robots, another one penalizes turn-on-the-spot be-

[2]Although the authors did not evaluate the performance of the generated control software on physical robots, we included their work in our review of the literature as their experimental methodology is relevant to the content of Chapter 4. In particular, the fact that the authors use the adjectives 'simple' and 'simplified' (with respect to the model used for evaluation) to describe the simulation model used during the design reflects the *complexity assumption* that we challenge in Chapter 4

haviors and emphasizes small differences of speed between the robots' wheels. The authors performed 10 designs per communication strategy, resulting in the generation of 10 instances of control software for each of them, and the best one is evaluated 30 times on 4 robots. The authors selected the best instances of control software on the basis of a different performance metric than the one used during the design. This performance metric only considers the distance covered by the connected swarm, and is described by the authors to be "a more informative measure of the controller's quality [with respect to the fitness function used to generate it]". When executed on the physical robots, two functions adjust the outputs of the neural networks to make the system less reactive to external stimuli; according to the authors, these modifications are necessary to prevent stress on the motors. Quantitative results show relatively small differences of performance between simulation and reality.

In his book dedicated to neuroevolutionary swarm robotics, Trianni (2008) describes several uses of neuroevolution to obtain control software to perform different tasks, such as self-organized aggregation, coordinated motion, hole avoidance, self-organized synchronization, and decision making. Taken separately, some of the applications of the neuroevolutionary approach might be qualified as fully-automatic (Baldassarre et al., 2007; Trianni and Nolfi, 2009). However, when compared to one another, the evolutionary setups of each case study appear to be different one from the others, and thus seem to be tailored to each of the tasks at hand. For what concerns the genetic algorithm, we noticed differences in the number of generations (100, 200, 500, or 5000), the number of offsprings generated during reproduction (either 4 or 5), the mutation probability (either 0.03 or 0.05), and the number of evaluations of the candidate solutions per generation (5, 8, 10, 12, or 16). For the two decision making tasks, the robots were controlled by continuous-time recurrent neural networks (for one of the tasks, the neural network has a multi-layer topology); for the other tasks, they are controlled by single-layer perceptrons.

Ampatzis et al. (2009) used neuroevolutionary robotics to generate self-assembly behaviors for two autonomous robots. The fitness function used to guide the evolution does not only reward the robots when they are able to connect to one another: it also contains a term that promotes aggregation; one that penalizes collisions and promotes straight displacements when robots approach one another, which is reported to improve transferability to physical robots; and one that rewards the robots when they are able to sense the presence of another robot with their special assembly sensor, which is reported to bootstrap the design. This last term comprises a constant whose value is not disclosed by the authors. The authors performed 20 designs and selected the best performing instance of control software to be evaluated on physical robots. The evolved solution showed good performance in reality.

Duarte et al. (2016) used neuroevolutionary robotics to generate control software for 4 tasks: homing, dispersion, clustering, and monitoring. The authors executed 10 neuroevolutionary processes for each task; suitable instances of control software were found after 100 generations for homing, dispersion, and monitoring, whereas for

clustering 400 generations were required to generate solutions that met the author's expectations. For each task, the three instances that obtained the highest score in simulation were selected to be executed on a swarm of 10 aquatic surface robots in a semi-enclosed waterbody. Results showed good performance in all 4 tasks. The authors eventually performed a proof-of-concept experiment in which the swarm executes the behaviors sequentially to perform a water temperature monitoring mission. The authors report the difficulty they faced to conceive a fitness function for the generation of a behavior to perform the complex monitoring mission. Their solution was to adopt a modular strategy: they divided the complex mission into the four aforementioned tasks, generated behaviors for each of them, and finally combined them.

Jones et al. (2018) proposed an evolutionary algorithm to generate control software in the form of behavior trees, which they used to solve a foraging mission. For their method to operate, one must implement an interface between the behavior trees and the robot, which include a trade-off between hard-coded capabilities and automatically evolved one. This interface is therefore mission specific. The authors conducted 25 design processes; the instance of control software that produced the highest performance in simulation out of the 25 ones produced by the different processes was uploaded to real robots. The authors observed a statistically significant performance drop between simulation and reality, yet the physical robots were able to forage effectively.

On-line semi-automatic design

Bredeche et al. (2012) proposed the mEDEA algorithm and validated it on a swarm of robots. The authors focused on the emergence of consensus behaviors in an experiment called "two-suns". They list a number of technical issues that they had to address in order for their algorithm to work on the physical robots. In particular, they had to modify values of parameters with respect to those used in simulation experiments, including the topology of the neural networks controlling the robots, the mutation rate, and the number of candidate solutions each robot could store. They also had to introduce a restart procedure in their algorithm to avoid robots to go idle. The authors report running the conduction of 21 initial experiments to determine which configuration of the swarm (e.g., number of robots, communication radius) would give better results. The authors then studied in details the performance of their mEDEA algorithm using the best configuration of the swarm on 8 runs.

Jones et al. (2019) used a distributed evolutionary algorithm to evolve behavior trees to perform a task in which robots have to move a frisbee in a predefined direction. In this approach, each robot performs its own design process in simulation while executing in reality the best instance of behavior trees available. The robots also periodically share information about the candidate solutions found to theirs peers, as it is done in the classical embodied evolution approach. The authors report having performed a few preliminary experiments with physical robots to adjust parameters of the simulator used by the robots. Out of their observations, they acknowledge the difficulty of simulating collisions correctly, and they decided to include a built-in collision

avoidance mechanism within all behavior trees generated. The authors also included a term that penalizes no movements of the frisbee to bootstrap the evolution.

Off-line fully-automatic design

Waibel et al. (2009) compared and studied the ability of four neuroevolutionary robotics methods to solve three variants of a foraging mission. In these tasks, robots have to push objects of different sizes to one side of the environment in which they operate. In the first variant, the environment contains light objects that can be pushed by a single robot; in the second variant, the environment contains heavy objects that can be pushed by two robots; in the third variant, the environment contains both light and heavy objects. The four neuroevolutionary methods differ from one another by the team composition and the level of selection of the swarm. The authors executed each configuration of the neuroevolutionary method 20 times on each mission variant, the resulting control software was then tested 10 times on robots. Results shows that the different configurations of the method lead to significant performance differences but that no configuration surpassed the others for all three variants of the mission considered.

Francesca et al. (2014b) introduced the AutoMoDe approach that consists in automatically selecting, combining, and fine-tuning predefined modules into a given control software architecture. The authors compared a proof-of-concept implementation of their approach, called AutoMoDe-`Vanilla`, to an implementation of neuroevolutionary robotics called `EvoStick` on two missions. For the two missions, the authors evaluated the ability of the design methods to produce control software using three levels of design budget—that is, the maximal number of simulated evaluations that the optimization algorithm is allowed to perform. For each mission and each design budget, the authors performed 20 executions of both `Vanilla` and `EvoStick` and kept the best performing control software obtained after each execution. They then executed all the obtained instances of control software once in simulation, and once on physical e-puck robots. The authors did not adapt the methods to the missions nor to the design budget they considered. Results showed that the method the authors presented, `Vanilla`, outperformed the neuroevolutionary one.

In a follow up work, Francesca et al. (2014a) compared the same, unmodified two automatic design methods `Vanilla` and `EvoStick` to two manual methods, namely `U-Human` and `C-Human`. In `U-Human`, the human designer can conceive the control software freely, whereas in `C-Human` they are constrained to use the `Vanilla`'s modules and combine them into `Vanilla`'s control architecture under the same constraints. The experimental protocol adopted by the authors is elaborate: 5 human designers were involved, and each had to (1) define a mission, (2) solve a mission under `U-Human`, and (3) solve another mission under `C-Human`. This way, the authors obtained 2 manually conceived instances of control software for each of the 5 missions defined (one of these instances was obtained via `U-Human`, the other via `C-Human`). They executed `Vanilla` and `EvoStick` once on each mission as well, to obtain 2 automatically pro-

duced instances per mission. All instances were then evaluated 10 times per mission on a swarm of 20 robots. Results show that the control software produced by the human designers under `C-Human` were the best performing one, and that `Vanilla` outperformed `EvoStick`, which confirmed they previous observations (Francesca et al., 2014b). The fact that `C-Human` outperformed `Vanilla` drove the authors to introduce a second implementation of AutoMoDe they named `Chocolate` (Francesca et al., 2015). The only difference between `Vanilla` and `Chocolate` lies in the optimization algorithm they use: `Vanilla` uses F-race (Birattari et al., 2002), whereas `Chocolate` uses an improved, more powerful version called Iterated F-race (Balaprakash et al., 2007; Birattari et al., 2010; López-Ibáñez et al., 2016). `Chocolate` outperformed both `C-Human` and `Vanilla` when applied to the 5 missions defined by the human designers.

The success of `Chocolate` has fueled the creation of more implementation of AutoMoDe methods (Hasselmann et al., 2018b; Kuckling et al., 2018; Salman et al., 2019; Garzón Ramos and Birattari, 2020; Kuckling et al., 2020; Ligot et al., 2020a; Spaey et al., 2020). These implementations typically differ from `Chocolate` or from one another by a single element of the design process and comparisons of control software produced by different implementations are performed to study the impact that these elements have on performance. In the following, we review the studies that made use of experiments on physical robots to evaluated the control software produced by the different implementations of AutoMoDe. In all these studies, each design method involved in the studies were executed 10 or 15 times on each mission and the best instance of control software found during each design process was then evaluated once on physical robots.

Hasselmann et al. (2018b), Garzón Ramos and Birattari (2020), Ligot et al. (2020a), and Spaey et al. (2020) proposed methods that differ from `Chocolate` by the set of modules available for combination. Hasselmann et al. (2018b) introduced `Gianduja`, an implementation of AutoMoDe that, in comparison with `Chocolate`, enables the robots to broadcast a single infrared message whose semantics is not defined a priori. They also introduced `EvoCom`, an extension of the neuroevolutionary design method `EvoStick` with the same communication capabilities as `Gianduja`. The authors compared `Gianduja`, `Chocolate` and `EvoCom` on the basis of three missions, and the results showed that `Gianduja` outperformed the other two methods. Similarly, Garzón Ramos and Birattari (2020) introduced `TuttiFrutti` that also enables the robots with communication capabilities, via RGB LEDs this time. The authors compared this implementation of AutoMoDe with the neuroevolutionary robotics counterpart `EvoCol` on the basis of three missions, and the results showed that `TuttiFrutti` produced control software that efficiently used the color-based information, and that it outperformed `EvoCol`. In Ligot et al. (2020a), we introduced `Arlequin` that differs from `Chocolate` by the nature of the modules it combines. In fact, in `Chocolate` the low-level behaviors are hand-crafted; in `Arlequin` we replaced them with neural networks obtained via the neuroevolutionary methods `EvoStick`. We compared `Arlequin`, `Chocolate`, and `EvoStick` on the basis of 2 missions, and the results showed that `Arlequin` outper-

formed EvoStick but was outperformed by Chocolate. Finally, Spaey et al. (2020) introduced Coconut that can produce control software with multiple configurable exploration schemes; Chocolate only has one exploration scheme at its disposal. The authors compared Coconut, Chocolate, and EvoStick on the basis of two variants of three missions. In the first variant of each mission, the robots operate in a bounded environment, like it is usually done in swarm robotics. In the second one, the environment is unbounded: 25% of the walls of the environment are removed, allowing the robots to wander off the area where the mission is taking place. Results showed that Coconut and Chocolate performed similarly, suggesting that the exploration scheme of Chocolate is appropriate, at least for the missions studied.

In Kuckling et al. (2018), we studied the impact of the control architecture in which the modules are assembled on the performance of the control software produced. In Chocolate or any other implementations of AutoMoDe mentioned above, modules are combined into probabilistic finite-state machines. We proposed Maple, an implementation of AutoMoDe that combines the same modules used in Chocolate into behavior trees (Champandard, 2007; Colledanchise and Ögren, 2018). When used to generate control software for two missions, Maple and Chocolate performed similarly and outperformed the evolutionary robotics method EvoStick.

In Hasselmann et al. (2021), we assessed and compared the ability of the most advanced neuroevolutionary robotics methods to generate control software for a swarm of 20 robots for 5 missions. The neuroevolutionary robotics methods included two configurations of CMA-ES (Hansen and Ostermeier, 2001), two configurations of xNES (Glasmachers et al., 2010), four configurations of NEAT (Stanley and Miikkulainen, 2002), and the aforementioned EvoStick. The empirical study also included Chocolate and a primitive behavior in which the robots move randomly in the environment—these two were used as baselines. Each design method was applied 10 times on each mission, and the best instance of control software after each execution was then evaluated once on physical robots; the primitive behavior was evaluated 10 times on the robots for each mission. Results showed that all neuroevolutionary methods suffered from an important performance drop when comparing their performance in simulation to the one obtained in reality, and that they produced control software that performed only marginally better than the primitive behavior.

2.3 The reality gap

One of the most challenging issue to be faced by off-line methods is the so-called *reality gap*: the difference between simulation and reality, which might be subtle but is unavoidable (Brooks, 1992; Jakobi et al., 1995). Due to the reality gap, it is likely that robot swarms do not display the same behavior in simulation and in reality (Floreano et al., 2008)—see Figure 2.3. This difference of behavior usually results in a drop of real-world performance with respect to the one observed in simulation. An issue that is often overlooked is that the occurrence of performance drops due to the reality gap

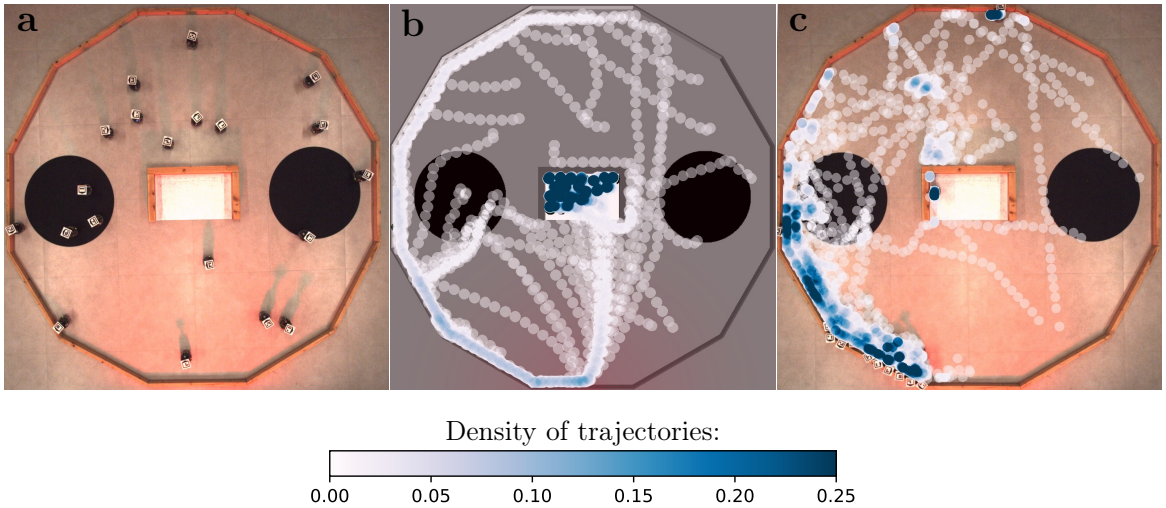


Figure 2.3: **Illustration of the effects of the reality gap.** (**a**) SHELTER mission in its initial configuration. The goal of the mission is for a maximal number of robots to aggregate in the white area as fast as possible. The white area is surrounded by walls on three of its sides. A source of light is placed outside the arena, facing the open side of the shelter, and can be used by the robots to orient themselves. (**b** and **c**) Traces of the trajectories of the robots when executing the same instance of control software generated by the neuroevolutionary method `EvoStick` in simulation and in reality, respectively. The darker the color of the spots, the longer a robot spend on that position. The strategy of the instance of control software, which is clearly visible in simulation (**c**) is to make the robots follow the walls in a anti-clock fashion until they reach the light, then go away from the light source to hopefully reach the shelter. The real robots are unable to properly execute the strategy: they follow the walls, but remain on the left hand side of the light source (**b**). The difference of behaviors displayed in the two environments results in an important performance drop from simulation to reality. The occurrence of the effects of the reality gap is a relative problem: other instances of control software might be able to display the same behaviors (i.e., trajectories) in simulation and in reality.

Table 2.2: **Taxonomy of the approaches dedicated to handle the reality gap**, that is, to produce control software that suffers from the smallest possible drop between the performance obtained in simulation and the one observed on physical robots.

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

is a relative problem: each instance of control software might be affected to a different extent. This relative nature can lead to a situation in which an instance of control software CS_A outperforms another instance CS_B in simulation, but CS_B outperforms CS_A when they are executed on physical robots. This phenomenon, which we call a *rank inversion*, has been observed when comparing instances of control software produced by different design methods (Koos et al., 2013; Francesca et al., 2014b, 2015), or by the same one at different steps along the optimization process (Birattari et al., 2016). Indeed, Birattari et al. (2016) observed a phenomenon that they called overdesign: past an optimal number of steps of the optimization process, the performance obtained in reality diverges from the one obtained in simulation. The phenomenon of rank inversion is thus particularly insidious as it questions the validity of the off-line design process in that it relies on the assumption that the higher the performance in simulation, the higher the performance in reality.

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

In the remainder of this section, we describe the functioning of the existing approaches and propose a taxonomy, which is summarized in Table 2.2. Note that here we do not only consider the literature on swarm robotics. Approaches to cross the reality gap have mainly been proposed and tested in the context of neuroevolution-

ary robotics for single robots. Nonetheless, they are typically general enough to be relevant to any design method based on off-line simulation, both for single- and multi-robot systems. Some approaches aim at reducing the differences between simulation and reality as much as possible (Miglino et al., 1995; Jakobi et al., 1995; Bongard and Lipson, 2004; Zagal et al., 2004; Koos et al., 2013), whereas others aim at making control software robust to these differences (Floreano and Mondada, 1996; Jakobi, 1997, 1998; Boeing and Bräunl, 2012; Francesca et al., 2014b). The first group of approaches appears to be driven by the hypothesis that the more accurate the simulation, the smoother the transition to reality; the second one, by the hypothesis that a fully accurate simulation is impossible and overfitting is always a risk. The two groups can be further detailed according to which element of the off-line design process is targeted by the approach: either the simulation models (Miglino et al., 1995; Jakobi et al., 1995; Jakobi, 1997; Bongard and Lipson, 2004; Zagal et al., 2004; Boeing and Bräunl, 2012) or the design method (Floreano and Mondada, 1996; Koos et al., 2013; Francesca et al., 2014b). Among the methods that focus on the simulation models, some aim at increasing their realism. Others use them to enhance the robustness of the design process. Among the methods that focus on the design process, some aim at conceiving a design process that avoids exploiting features of simulation that do not match reality. Others aim at making the design process intrinsically more robust. We describe these methods in the following paragraphs.

Focus on simulation models to reduce differences between simulation and reality

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

Bongard and Lipson (2004) proposed an approach they called *estimation-exploration*, which consists in the simultaneous evolution of control software and simulation models. The authors generated control software for a quadrupedal robot so that it

could walk the longest distance possible. The behavior produced crossed the reality gap successfully. Zagal et al. (2004) proposed a similar approach they named *back to reality*. The method was validated with a Sony AIBO robot on two tasks: gait optimization and ball-kicking (Zagal and Ruiz-del Solar, 2007). The two aforementioned methods rely on periodic evaluations of instances of control software on the target robot, but differ in the data used to improve the simulation models: *estimation-exploration* uses sensor samples from the physical robot; whereas *back to reality* uses the performance drop experienced in reality.

Focus on simulation models to enhance robustness of control software

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

Although they do not directly cite the work of Jakobi, Peng et al. (2018) and Andrychowicz et al. (2020) reintroduced his idea of applying random variation in the simulation models. They proposed a technique called *domain randomization* in the context of the application of reinforcement learning to robotics. Specifically, they showed that, thanks to this technique, control software developed in simulation can be successfully ported to physical robots: Peng et al. (2018) applied the technique to a robotics arm that was required to push an object on a table; Andrychowicz et al. (2020) to a five-fingered humanoid hand that was required to manipulate a cube.

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

Focus on design methods to reduce differences between simulation and reality

Koos et al. (2013) proposed what they called the *transferability approach*. In this approach a bi-objective algorithm optimizes: (i) a mission-dependent performance

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

Focus on design methods to enhance robustness of control software

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

Francesca et al. (2014b) have also deviated from traditional neuroevolutionary robotics. They started from the conjecture that neuroevolutionary robotics is particularly affected by the reality gap due to the excessive representational power of neural networks. As a result, neuroevolutionary robotics is likely to overfit the conditions experienced during the design process. Guided by the notion of bias-variance tradeoff (Geman et al., 1992), the authors introduced a novel approach to the off-line automatic design of robot swarms: AutoMoDe. In this approach, robots are controlled by a modular software architecture (e.g., a finite state machine, a behavior tree) automatically generated by assembling and fine-tuning predefined, hand-crafted modules. Compared to the neural networks used in neuroevolutionary robotics, the control software generated by AutoMoDe features a lower representational power: it is restricted to what can be obtained by assembling the predefined modules. The original method—AutoMoDe-`Vanilla`—has shown to produce control software that crosses the reality

gap more satisfactorily than those produced by `EvoStick`, an implementation of the traditional neuroevolutionary robotics (Francesca et al., 2014b). These results were further confirmed by follow-up studies (Francesca et al., 2015; Kuckling et al., 2018; Hasselmann et al., 2018b; Ligot et al., 2020a). In particular, in Ligot et al. (2020a) we assessed whether the conjecture of Francesca et al. on the bias/variance tradeoff also holds true when the predefined behavioral modules are generated automatically via neuroevolution or whether it is due to the fact that these modules are created by hand like it is the case for AutoMoDe-`Vanilla` and all subsequent implementations of AutoMoDe. The method we created—AutoMoDe-`Arlequin`—corroborated the conjecture as it also outperformed the neuroevolutionary method `EvoStick` in reality despite being the other way around in simulation.

2.4 Discussion

The literature on the optimization-based design of control software for robot swarm lacks a well-established and consistently applied empirical practice. In our review, we highlighted the various experimental protocols adopted throughout the published studies. What differentiate them from one another is the number of missions considered (from 1 up to 5), the number of design process executed (from 1 up to 25), the number of instances of control software resulting from the design processes that are then executed on the physical robots (from a subset of one to all of them), and the number of executions of these instances on the robots (from 1 to 100). Up until recently, comparisons of design methods and ideas were almost nonexistent (Francesca and Birattari, 2016). The fact that the first empirical comparison based on experiments performed with real robots of off-line methods belonging to the neuroevolutionary robotics approach (Hasselmann et al., 2021)—the most popular and most studied approach of optimization-based design—came almost two decades after the first use of this approach in swarm robotics (Quinn et al., 2003), is a blatant example of this observation.

The categorization between semi-automatic and fully-automatic design is the first step towards the establishment of a clear state of the art as it contributes to highlighting the research questions relevant to each approach, to setting appropriate expectations of what each should achieve, and to defining the challenges to be faced by each of them. The two approaches are both relevant to the development of swarm robotics and we expect them to address different contexts of applications.

We reckon that a semi-automatic approach is relevant when one has to solve a complex, one-of-a-kind mission for which reasonably large resources (time and budget) are available. Our review of the works that we classify as belonging to this category showed that, mainly thanks to the results of neuroevolutionary robotics obtained in both on-line and off-line design, the semi-automatic approach is an effective way to conceive robot swarms. However, none of the works in semi-automatic design comport comparisons of design methods or of neuroevolutionary setups (objective comparisons

is indeed problematic when human decisions intervene in the design process) and no directives or guidance other than the *ad hoc*, trial-and-error approach emerges from the current literature.

We reckon that a fully-automatic approach is relevant when one has to repeatedly solve missions belonging to a given class without the possibility for a human to supervise the design process or to verify its output prior to the deployment of the robots. When used in a fully-automatic off-line fashion, neuroevolutionary methods have shown to suffer heavily from the reality gap; they produce control software that perform very well in simulation, but poorly in reality. The aforementioned empirical study showed that the reality gap has a devastating effect on all the most advanced neuroevolutionary robotics methods: results were at most only marginally superior to a random walk behavior. All the works we classified as belonging to the fully-automatic approach compared the ability of multiple design methods to produce control software to solve between 2 and 5 missions or configurations of a mission. In most of these works, authors evaluate the expected performance of a design method by executing it multiple times on the missions considered, and by evaluating the produced control software once on the physical robots. This experimental protocol has been demonstrated to minimize the variance of the estimated performance for the specific mission considered (Birattari, 2004, 2020). However, this protocol fails to consider one of the corner stones of the fully-automatic approach: the class of mission. It is therefore not appropriate to estimate the expected performance of a fully-automatic design method. In Chapter 3, we propose one that does.

The reality-gap problem has been the center of attention of many studies. Unfortunately, the ideas proposed to solve it or mitigate its effects have not been sufficiently studied nor compared. Moreover, with the exception of the AutoMoDe approach, none of them has been applied to swarm robotics. It remains the most important problem to be faced in off-line optimization-based design, and it entails the conduction of expensive and time consuming robot experiments to reliably assess control software. In Chapter 4 and 5, we elaborate on simulation-only methodologies that could be used to predict real-world performance; in Chapter 6 we perform a large-scale empirical assessment of these methodologies.

Chapter 3

A protocol for fully-automatic design

In this chapter, we propose an experimental protocol for the comparison of off-line fully-automatic design methods.

A fully-automatic design method is expected to be able to produce control software for a whole class of mission without undergoing any human, per-mission adaptation. Our review of the literature on optimization-based design in Chapter 2 revealed the adoption of a variety of distinct experimental protocols. We argue that none of them is appropriate for evaluating the expected performance of fully-automatic design methods.

The protocol we propose here is characterized by two notable elements: a way to define benchmarks for the evaluation and comparison of design methods, and a sampling strategy that minimizes the variance when estimating their expected performance. Benchmarks are decisive tools in the identification of strengths and weaknesses of a method, and they promote the consistent application of meaningful, coherent, and well-defined evaluation criteria. Conceptually, a benchmark for the evaluation of fully-automatic design methods is a (possibly infinite) class of missions: a set of missions associated with a probability measure that determines their relative frequency of appearance. A class of missions might comprise both missions that are of different types (i.e., that differ by the nature of their goals), and missions of the same type that differ by minor variations. These minor variations can be at the level of the environment in which the swarm operates (e.g., different density of robots, presence of a reference point, number of points of interest), or at the level of the swarm itself (e.g., number of robots, initial configuration of the swarm). For example, two missions are of different type if the goal of one is for the robots to aggregate at a point of interest, and the one of the other is to gather objects initially scattered in the environment. For example, two missions of the same type differ in minor variations if, *ceteris paribus*, a task is to be accomplished in an environment in which a reference point such as a light source is present, and in an environment without a reference point. Although the

difference between the two given missions might be qualified as minor, it can be sufficiently important that the two missions benefit from a tailored design. In the example, a design method is likely to exploit the reference point to produce control software that enables the robots to orient themselves in the environment and find the points of interest faster than in the case where no reference point is present. An operational definition of a class of missions can be given in the form of a *mission generator*: a computer program that generates missions belonging to the class at hand. In other terms, running a mission generator is effectively a way to sample the corresponding class of missions—that is, selecting one of the missions of the set, according to the associated probability measure, and independently of the design method to which it applies. In this chapter, we illustrate the concept of mission generator by presenting one we named MG 1, short for *mission generator 1*. We use MG 1 in an illustrative study in which we compare two previously proposed design methods.

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

It is our contention that the protocol we propose here is crucial for the development of the optimization-based design of robot swarms into a mature scientific domain: it will contribute to make clear and objective comparisons between different methods that will allow to establish an objective state of the art. Eventually, this will promote

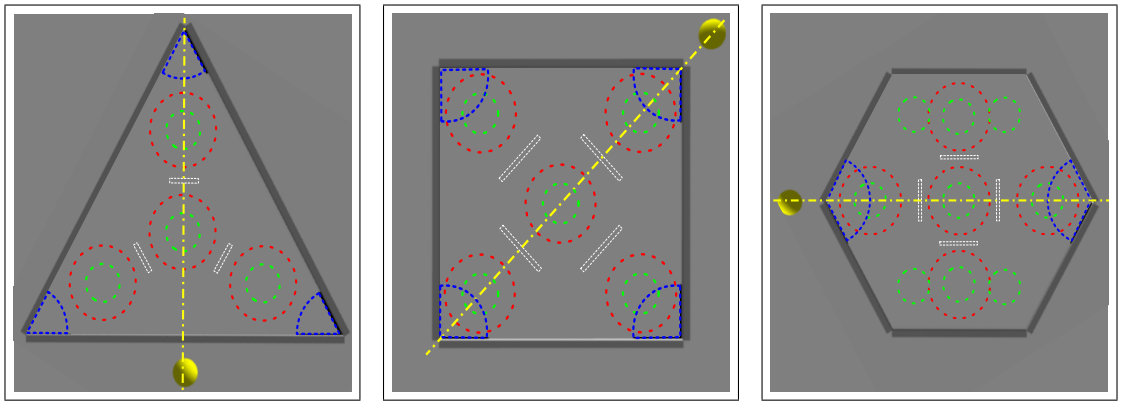
the best ideas proposed so far and the elaboration of new ones. This protocol is to be adopted when one evaluates design methods in the fully-automatic context and thus wants to estimate their performance over a whole class of missions rather than over specific ones (Birattari et al., 2019, 2020). The two components of the protocol are fundamentally complementary: the sampling strategy recommends to evaluate a design method on the maximal number of missions possible, and the notion of mission generator allows one to sample as many missions as desired.

The remainder of the chapter is organized as follows: in Section 3.1, we describe the implementation of the mission generator MG 1; in Section 3.2, we elaborate on the sampling strategy that minimizes the variance of the estimated performance of a fully-automatic design method; and in Section 3.3, we present an illustrative experiment. In the illustrative experiment presented in Section 3.3, we demonstrate the concepts introduced by evaluating and comparing two previously proposed fully-automatic design methods on 30 missions. We evaluate the two methods using the sampling strategy described in Section 3.2 on the 30 missions generated by the mission generator MG 1 described in Section 3.1.

3.1 A mission generator

In this section, we present MG 1, a generator of missions for robot swarms. A mission generator samples missions from a class of missions according to the associated probability measure. The class of missions is defined on the basis of the capabilities of a robotic platform, which are described by a *reference model* that formally characterizes the sensors and actuators of the platform (Francesca et al., 2014b). Indeed, it would not be reasonable to consider a class of missions containing ground missions if aquatic robots are expected to perform samples of this class, or missions that require sorting objects according to their color by robots that are unable to distinguish colors. The probability measure can be tuned so as to mimic a realistic frequency of appearance of deployment conditions. For example, based on previous rescue missions at sea, it could be determined that 80% were performed when the sea swell was high, 15% when it was moderate, and 5% when it was low. These probabilities can be used to define a generator so that sampled missions reflect the conditions that the robot swarm will face when deployed, and therefore enable a realistic estimation of the expected performance of a design method.

MG 1 samples missions defined on the basis of the capabilities of an enhanced version of the e-puck robot (Mondada et al., 2009). This version of the e-puck robot is capable of perceiving a light source, the presence of nearby obstacles, and the gray-scale color of the ground directly below its body. It can also detect the presence and estimate the relative position of neighboring peers. These capabilities are formally described by the reference model RM 1.1 (Hasselmann et al., 2018a) reproduced in Table B.1 for the reader's convenience. The missions take place in an enclosed area surrounded by walls—what we call an *arena*. The ground of the arena is gray, with



Source: Ligot A, Cotorruelo A, Garone E, Birattari M (2022)

Figure 3.1: **Schemes of the possible arenas that can be generated by MG 1.** Dotted circles, portions of circles, and rectangles represent placeholders for the possible environmental elements. The triangle arena covers an area of 2.5 m^2; the square one, an area of 3.76 m^2; and the hexagonal one, an area of 4.3 m^2. The dotted circles and portions of circles represent placeholders for possible black or white areas. The red circles are placeholders for circular areas of 0.3 m radius; the green ones, for circular areas of 0.15 m radius; and the blue ones are placeholders for portions of circular areas of 0.40 m radius. The white dashed rectangles represent placeholders for possible obstacles whose height and width are fixed to 0.07 m and 0.026 m, respectively. The length of these obstacles depend on the shape of the arena: 0.25 m, 0.45 m, and 0.35 m for the triangle, square, and hexagonal one, respectively. The yellow sphere represents the light source. For missions of type FORAGING, the possible nest areas are restricted to the circles positioned on the axis of the light source pictured by the yellow dashed-and-dotted line.

some areas being black or white. Obstacles can be present within the arena, and a unique source of light is positioned right outside the arena's walls. The light source is either on or off for the whole duration of the mission. Figure 3.1 illustrates the possible arenas that can be generated by MG 1 and provides details about the possible positions and sizes of the colored areas (i.e., black or white) and the obstacles.

MG 1 can instantiate missions of three types: FORAGING, HOMING, and AGGREGATIONXOR. To ensure the soundness of the instances created, we implemented within MG 1 several mission-specific conditions that guide the configuration of the arenas. Independently of the type of mission to be accomplished, MG 1 selects the number of robots in the swarm, their initial distribution, and the duration of the mission. We considered three families of initial random distribution for the robots: (i) uniform, the robots are deployed anywhere in the arena; (ii) one-side, the robots are deployed on one half of the arena, either close or far from the light and independently of whether the latter is on or not; and (iii) not-on-colored-areas, the robots are deployed anywhere in the arena, but not on the black or white areas. Table 3.1

Table 3.1: **The parameters, their possible values, and the corresponding probability distributions in MG 1**. The swarm can comprise 15 or 20 robots. There is a total of 5 different objective functions that can be selected by MG 1: one for FORAGING, and two for both HOMING and AGGREGATIONXOR. For the 5 objective functions to appear with equal probability, MG 1 selects HOMING and AGGREGATIONXOR with double the probability of FORAGING. See Sections 3.1.1 to 3.1.3 for descriptions of the types of missions.

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

summarizes the main parameters of MG 1; the types of missions and their respective conditions are described in the following subsections.

We created MG 1 as an open-source library[1] for the ARGoS3 simulator (Pinciroli et al., 2012). It should be noted that the library we share does not only implement MG 1, but a whole family of mission generators. Indeed, by modifying the parameters of MG 1, such as the frequency of appearance of the missions, one can create a different mission generator that would sample a different class of missions.

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

3.1.1 Foraging

The robots must retrieve virtual items from food sources to nest areas. A robot is deemed to pick up an item when it enters an area representing a food source, and drop the item when it then enters an area representing a nest. There may be up to two nests and up to three food sources. The nests areas may only be placed on the axis perpendicular to the light source depicted as the yellow dotted-and-dashed line in Figure 3.1. The food sources may be placed all around the arena. The size and positions of the areas are selected randomly with equal probability. MG 1 ensures that the areas representing the nests and the ones representing the food sources are of different color (i.e., if the nest is white, the food sources are black, and vice versa). The objective function to be maximized is $F_{\text{FORAGING}} = I$, where I is the number of items dropped in the nest areas after the duration of the mission.

3.1.2 Homing

The robots must aggregate on a black or white area designated as their home. MG 1 places between one and three areas in the arena: one for the home, and possibly two others to serve as distractions. MG 1 ensures that the area designated as the home is large enough so as to accommodate all the robots, and that the color of the distraction areas differs from that of the home area. There are two possible objective functions for this type of mission: *anytime* and *endtime*. The two objective functions are to be maximized and depend on N_{home}, the number of robots located in the aggregation area. With *anytime*, the performance is measured by the objective function $F'_{\text{HOME}} = \sum_{t=1}^{T/100\,\text{ms}} N_{home}(t)$, where T is the duration of the mission (in seconds) and 100 ms is the period at which N_{home} is evaluated. With *endtime*, the performance is measured once, at the end of the mission, and the objective function is $F''_{\text{HOME}} = N_{home}$.

3.1.3 AggregationXOR

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

3.2 Sampling strategy for performance estimation

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

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

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

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

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

In a general case a sampling strategy requires that one defines the number of missions, the number of design processes for each mission, and eventually the number of executions per design. Under the assumption of a lack of previous knowledge of ranges and variances of the performance, there is no reason to run a different number of design processes per mission and/or a different number of executions per design. As a consequence, a sampling strategy for estimating the expected performance of a design method on a class of missions, given that a maximum number N of executions can be performed, can be formally described by a triple $\langle n_m, n_d, n_x \rangle$, with $n_m \cdot n_d \cdot n_x \leq N$. The expected performance is estimated on the basis of n_m missions, n_d design processes per mission (to generate n_d instances of control software per mission), and n_x executions

of each of them. It has to be noticed that any triple $\langle n_m, n_d, n_x \rangle$ yields an unbiased estimate of the expected performance. Yet, different triples might differ for what concerns the variance of the estimate they yield, and it is thus of interest to understand which triple minimizes such variance.

In statistics, similar problems were studied since the early 1940s (Ganguli, 1941) under the name of *nested sampling*.[2] Classical results are based on the assumption that the sampled variable can be written as a sum of three mutually independent random variables—e.g., see (Hardeo and Ojeda, 2005b,a). Nested sampling has been used in numerous and very diverse fields—e.g., food quality control (Marcuse, 1949), agriculture (Kerry et al., 2010), blood pH of female mice (Sokal et al., 1995), premium of insurance contracts (Bühlmann, 1967), estimation of income (Fay III and Herriot, 1979).

The theoretical foundation of this chapter is summarized in Theorem 1, which shows that, given a maximum number of executions, the best strategy is to maximize the number of missions to be considered. Note that, unlike the results available in the literature (Ganguli, 1941; Hardeo and Ojeda, 2005b,a), this theorem does not require that the sampled random variable is expressed as the sum of three mutually independent random variables.

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

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

$$\mathbb{E}\left[(\hat{\mu}_{\mathcal{E}} - \mu)^2\right] = \frac{\sigma^2_{AM}}{n_m} + \frac{\bar{\sigma}^2_{AD}}{n_m\, n_d} + \frac{\bar{\sigma}^2_{WM}}{n_m\, n_d\, n_x}, \tag{3.1}$$

where σ^2_{AM} is the *across-mission variance* and indicates how missions differ from one another, $\bar{\sigma}^2_{AD}$ is the *expected across-design variance* and indicates how designs differ from one another within a same mission (averaged across all possible missions), and $\bar{\sigma}^2_{WM}$ is the *expected within-mission variance* and indicates how scores differ from one another within a same mission (averaged across all possible missions). Formal definitions of these three variances and a formal proof of Equation 3.1 are given in Appendix A. Clearly, to minimize the variance of the estimator, the denominators need to be chosen so as to be as large as possible. It is straightforward to conclude that this happens when $n_m = N$, $n_d = n_x = 1$ under the constraint $n_m \cdot n_d \cdot n_x \leq N$. □

The same conclusion (i.e., that the triple $\langle N, 1, 1 \rangle$ is the one that minimizes the variance) is relevant also in the case one wishes to compare the expected performance of two design methods—the reasoning can be generalized to more than two design

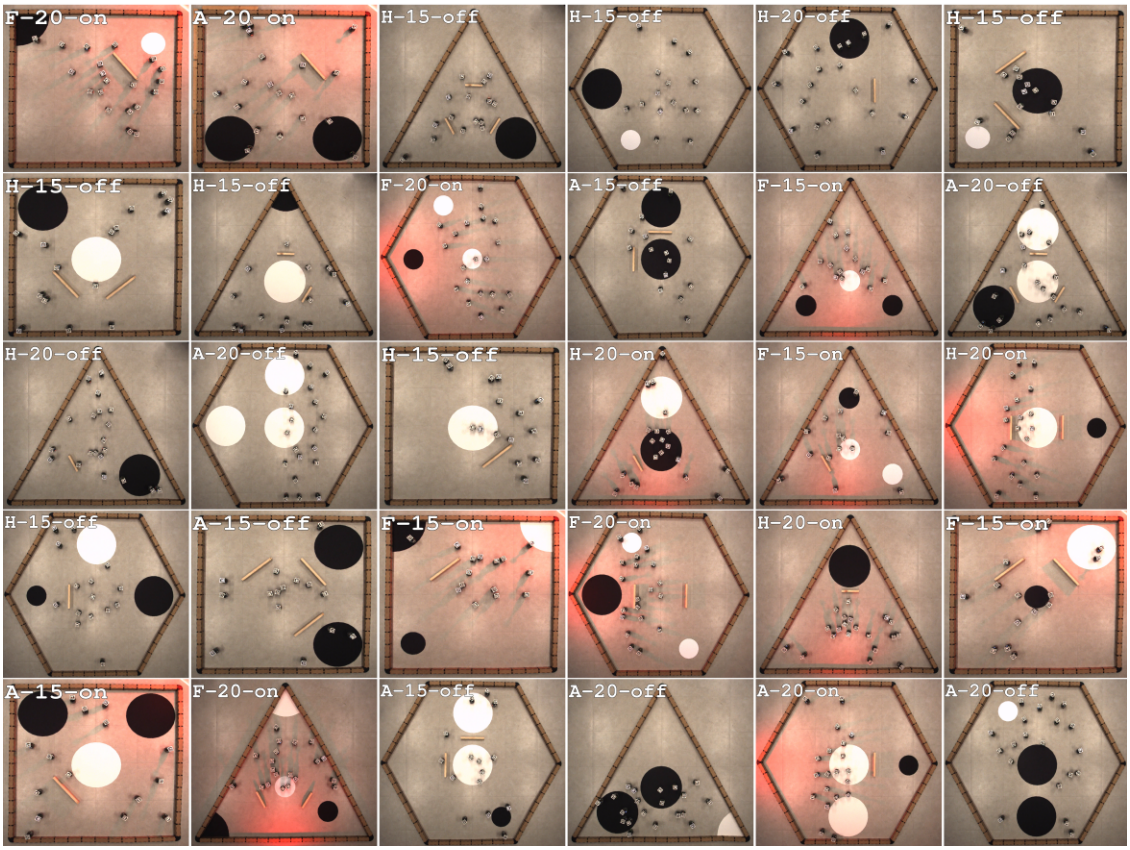
[2] Currently, the term *nested sampling* is in use in Bayesian statistics and refers to a completely different and unrelated technique.

methods, as well. When two methods are considered, the reasoning presented above applies to the estimation of the expected value of the difference between the performance of the control software produced by the two methods under analysis.

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

3.3 Illustrative experiment

In this section, we assess and compare the performance of two previously proposed fully-automatic design methods following the sampling strategy described in Section 3.2: we consider 30 missions generated with MG 1, run each design method once on each mission, and execute each instance of control software produced once on the



Source: Ligot A, Cotorruelo A, Garone E, Birattari M (2022)

Figure 3.2: **Pictures of the initial configurations of the missions generated by MG 1**. Each image is labeled with the initial letter of the mission to be accomplished (that is, F for FORAGING, H for HOMING, and A for AGGREGATIONXOR), the number of robots in the swarm (that is, 15 or 20), and the status of the light (that is, on or off).

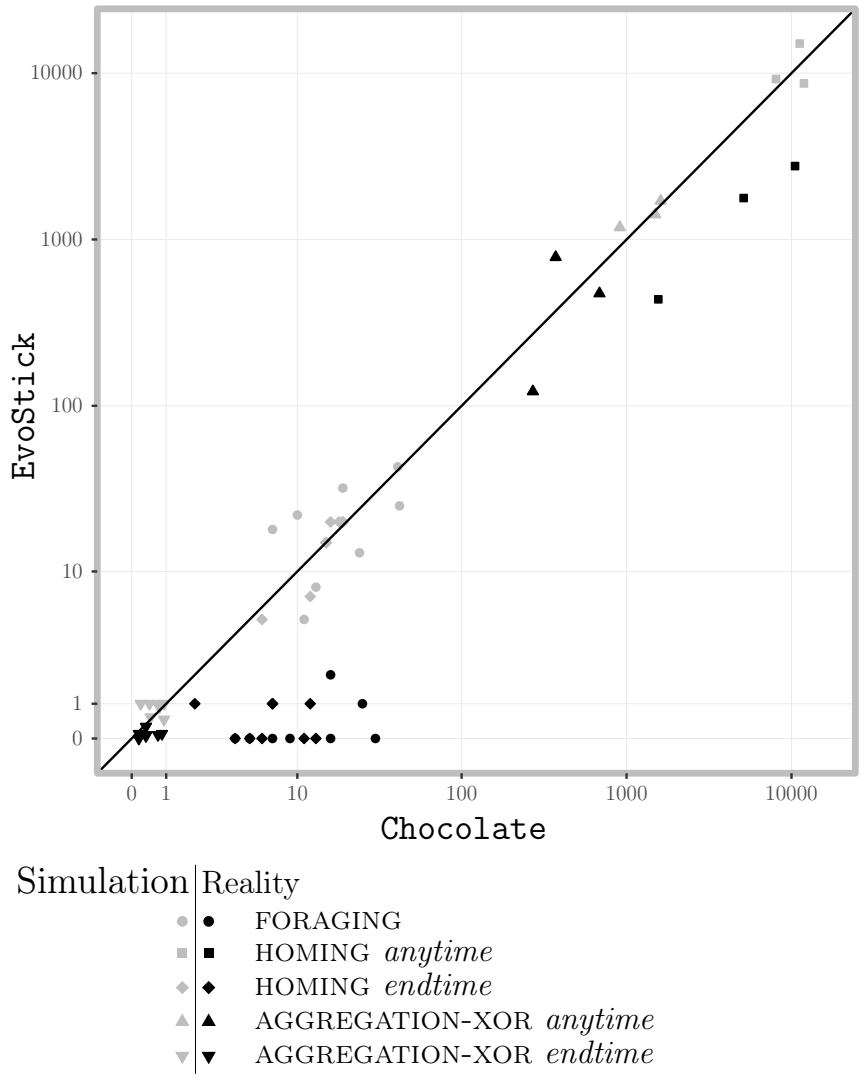
physical robots. We allocated a design budget of 100 000 simulation executions for each method on each mission: that is, each design process cannot exceed 100 000 simulation runs. To evaluate the intrinsic robustness of the methods, we also assess each instance of control software produced in simulation under the same initial conditions of the execution on the physical robots. Figure 3.2 shows pictures of the 30 arenas generated by MG 1.

We used `EvoStick` (Francesca et al., 2012, 2014b) and `Chocolate` (Francesca et al., 2015) to design control software for the 30 missions in a fully-automatic off-line way. `EvoStick` is a neuro-evolutionary robotics method that produces control software in the form of fully-connected neural networks without hidden layers. The 25 input nodes of these neural networks are fed with the readings of the sensors, as formally described by the reference model RM 1.1 described in Table B.1. The 2 output nodes

determine the velocity of the wheels. The 50 synaptic weights that connect the input nodes to the output nodes are real numbers in $[-5, 5]$, and are optimized by a genetic algorithm with a population size of 100 individuals and 10 evaluations per generation. The design budget of 100 000 simulation executions allocated to the genetic algorithm corresponds to 100 generations. `Chocolate` is a modular method that belongs to the AutoMoDe framework. It produces control software in the form of probabilistic finite state machines by selecting, fine-tuning, and combining pre-defined modules. The modules are programmed by hand *a priori* in a mission-agnostic way. They include 6 low-level behaviors to be used as states of the probabilistic finite state machines, and 6 conditions to be used as transitions between states. The probabilistic finite state machines produced can contain up to 4 states and 4 outgoing transitions per state, and are optimized by the optimization algorithm *Iterated F-race* (Birattari et al., 2010; López-Ibáñez et al., 2016). It is important to notice that the modules of `Chocolate`, although programmed by hand, are defined once and for all in a mission-agnostic way, and are not manually modified during the design process, which rightfully qualify `Chocolate` as a fully-automatic design method. We refer the reader to the original papers for further details on the methods Francesca et al. (2014b, 2015).

We chose `EvoStick` and `Chocolate` because the two methods have already been compared in several studies (Francesca et al., 2015; Ligot et al., 2020b,a), and we wish to keep the focus of this chapter on the experimental protocol used rather than on the outcome of a novel comparison of design methods. In these previous studies, results were always similar: `EvoStick` outperformed `Chocolate` in simulation, but `Chocolate` outperformed `EvoStick` in reality—in both cases, differences were significant with a confidence level of at least 95%. We expect to obtain similar results in our illustrative experiments. The novelty of the comparison we present here lies in the experimental protocol adopted. In the previously presented experiments, the experimental protocol considered 2 (Ligot et al., 2020b,a) and 5 (Francesca et al., 2015) missions selected by the experimenters, on which each method was executed 10 (Ligot et al., 2020b,a) or 20 (Francesca et al., 2015) times. These experimental protocols are not wrong, and the results obtained should not be disregarded. In fact, the adopted sampling strategy minimizes the variance of the expected performance for the specific missions considered (Birattari, 2004, 2020). However, as discussed in Section 3.2, the results of these previous experiments strongly depend on the missions chosen, which, due to specificities that might be unknown to the experimenters, could favor a given design method over another. The protocol proposed in this chapter aims at evaluating and comparing the performance of design methods over a whole class of missions rather than over specific ones, and should therefore be adopted when one evaluates design methods in the fully-automatic context (Birattari et al., 2019, 2020).

In simulation, `EvoStick` outperformed `Chocolate` in 18 out of the 30 considered missions, and the two methods obtained the same score in 3 missions. In reality, `Chocolate` outperformed `EvoStick` in 29 missions, and was outperformed only in one—see Fig. 3.3. Like in previous studies, `EvoStick` indeed produced control software that



Source: Ligot A, Cotorruelo A, Garone E, Birattari M (2022)

Figure 3.3: **Scatter plot of the performance obtained for each mission**. Gray points represent the performance obtained in simulation, black ones represent the performance obtained in reality. The performance is given in logarithmic scale. A point on the diagonal indicates that the two methods performed similarly on a given mission; a point below the diagonal indicates that `Chocolate` performed better than `EvoStick`, and inversely.

suffered from larger performance drop than the one produced by `Chocolate`. The configurations of the 30 missions generated, the instances of control software produced, the raw data obtained, and videos of the physical robots executing the missions are available as on-line supplementary material (Ligot and Birattari, 2022c).

Aggregating the performance observed on several different missions is not trivial

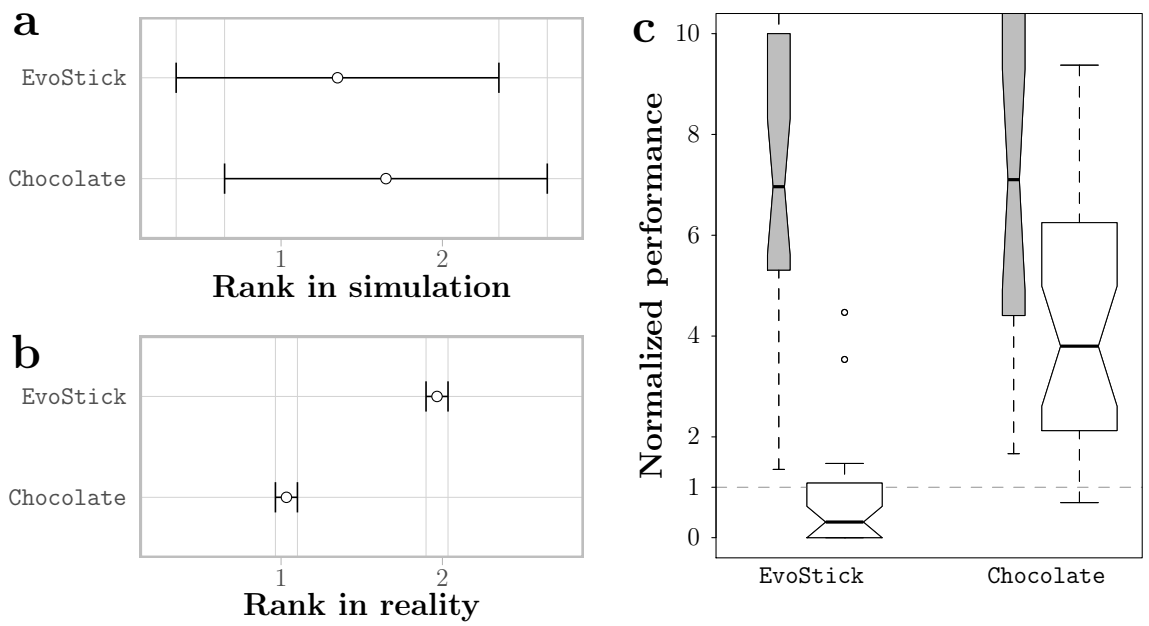


Figure 3.4: **Aggregated performance obtained in simulation and in reality**. (**a**) and (**b**) Expected rank and 95% confidence interval of the expected rank in simulation and in reality, respectively. The lower the expected rank, the better the performance. If two segments do not overlap, the expected ranks of the corresponding methods are significantly different with a confidence level of at least 95%. (**c**) Box-and-whisker plots of performance across all missions. Notches on the box represent the 95% confidence interval on the median. If notches on different boxes do not overlap, the medians of the corresponding methods differ significantly, with a confidence of at least 95%. To aggregate across missions, we normalized the performance with the performance obtained in simulation by a random walk behavior: they are computed as $\frac{P_{rw}(i)-P(i)}{P_{rw}(i)}$ for each mission i, where P is the performance of an instance of control software generated to solve mission i and P_{rw} is the performance of the random walk behavior on that mission. The horizontal dashed line represent the normalized performance of the random walk behavior. The higher, the better.

because the range of performance for different missions might vary greatly. Therefore, naively averaging the performance could give misleading results as missions for which the performance range is large would overshadow those for which it is small. Some form of normalization is needed to aggregate the performance observed on different missions. Here, for the purpose of this illustrative experiment, we address this issue in two ways. The first one consists in aggregating the performance observed on the different missions by reporting an estimation of the expected rank together with a 95% confidence interval—see Fig. 3.4a, b. By computing the ranks on a per-mission basis, we put the results observed for the different missions on an equal footing, irrespectively of their possibly different ranges. The second one consists in normalizing

the performance of the generated control software on the basis of the performance of a baseline behavior: random walk—see Fig. 3.4c. As the random walk behavior we use has shown to perform equally well in simulation and on physical e-puck robots (Hasselmann et al., 2021), we normalize the performance for the different missions with the simulation performance of the baseline behavior to avoid further robot experiments. For both aggregation techniques, outcomes are similar: `Chocolate` and `EvoStick` perform similarly in simulation, whereas `Chocolate` performs significantly better than `EvoStick` in reality.

3.4 Discussion

To the best of our knowledge, mission generator MG 1 is the first generator of missions for swarm robotics. Because its purpose is merely to illustrate the concepts introduced, MG 1 is relatively limited in the nature of the missions it generates: it generates missions belonging to only three types of missions to be solved by specific robots with specific capabilities. However, MG 1 can be extended and generalized in many different ways, and therefore can be the starting point for future generators. In fact, the core idea of MG 1 is to generate missions of different types that are characterized by specific objective functions, and the definition of features of the environment together with relationships between these features. These elements can be easily reused to create mission generators dedicated to other robots, including robots of different nature (e.g., flying robots), under the condition that reasonable distributions can be devised for every variable of the missions. It should be also noted that in MG 1 we consider the number of robot as a parameter of the mission. As an alternative, the definition of the mission could impose a constraint on the maximum/minimum number of robots comprised in the swarm and the selection of the most appropriate number of robots could be left to the design process.[3]

To illustrate the protocol, we conducted an experiment in which we evaluated and compared two previously proposed optimization-based design methods on 30 missions generated by MG 1. In this illustrative study, we allocated the same design budget to the two methods for each mission they had to solve. A thorough assessment of the capabilities of fully-automatic design methods would require the evaluation of these methods under different levels of the design budget to understand which methods performs best under which conditions. Further, in the illustrative study, we estimated the performance of the methods under analysis on each mission, but the techniques we used to compute the aggregate performance across the missions are not ideal. The main difficulty when estimating the performance of a design method on different missions is that the range of performance might vary greatly across the missions at hand. Some sort of normalization prior to the aggregation of the performance is

[3]The idea of defining the size of swarm automatically within the design process has been already explored by Salman et al. (2019).

mandatory. Here, we aggregated the performance observed by reporting the expected rank, which is an implicit form of normalization. The drawback of using ranks is that they do not provide an estimation of the overall performance of each of the design methods under analysis. As an alternative to using ranks we also considered normalizing the performance obtained by the design methods on a given mission based on the performance of a baseline behavior—namely, random walk—when executed on the same mission. Although interesting, the normalization we performed is not optimal as we used the performance of the random walk behavior observed in simulation rather than the one observed on physical robots, as it should be. Yet, using the real-world performance of the random walk behavior might not be the perfect solution either: it would entail that an important part of the (bounded) number of executions of control software on the physical robots are dedicated to the evaluation of the baseline behavior rather than to the evaluation of the design methods themselves. Also, because the normalization involves a division by the performance of the baseline behavior, it cannot be computed if the performance of the baseline behavior is null. In our case, to remediate to occasional cases in which a null performance was observed, we used the average performance of the random walk behavior over five evaluations in simulation for each mission. Assessing multiple times the baseline behavior only emphasizes the aforementioned issue related to the executions on physical robots. Moreover, it does not exclude divisions by zero. As an alternative, one could normalize the performance on each mission based on the knowledge of the theoretical maximal and minimal performance, or based on a reasonable estimate of them, including, for example, the best and worse performance observed empirically (Hasselmann et al., 2021). However, in this illustrative experiment, these alternatives did not appear to be appropriate as no prior knowledge was available and only two methods were involved in the study, providing therefore too little data to perform a meaningful normalization.

Chapter 4

Challenging the complexity assumption

In this chapter, we introduce the device that is at the basis of the real-world performance predictors that we describe in the rest of the thesis: the pseudo-reality. Here, we use this device to shed further light on the reality gap, the most challenging problem in off-line design.

Off-line optimization-based methods are faced with the so-called reality gap: the difference between simulation and reality, which might be subtle but is unavoidable (Brooks, 1992). Due to the reality gap, it is likely that robot swarms do not display the same behavior in simulation and in reality. Several approaches have been proposed to mitigate the effects of the reality gap, but none of them has been studied in detail, no extensive comparison has been produced, and the reality gap remains an important issue in off-line design.

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

In the literature, the effects of the reality gap have only been observed when con-

trol software has been developed in a relatively simpler setting for then being tested in a more complex one. This holds true for the most common case in which control software is developed in simulation and then tested on real robots. It also holds true for two cases in which control software has been developed using a first simulator and then tested using a second one. Indeed, to the best of our knowledge, Dorigo et al. (2003) and Koos et al. (2013) have been the only ones to consider an artificial, simulation-only reality gap, which they used to investigate the properties of automatically generated control software. In both cases, their artificial reality gap relies on a simplified simulator to be used in the design, and an accurate one to be used for the evaluation.

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

The concept of pseudo-reality refers to a simulation model that differs from the one used in the design. It emerged from the contention that if an instance of control software shows similar performance in the simulation model on which it has been designed and in a different simulation model, it is somehow 'intrinsically' robust and it can be expected to cross the reality gap more satisfactorily than another instance that does not. By intrinsic robustness, we informally refer to the general ability of a design method to produce control software that transfers seamlessly from any (reasonable) model to reality, as opposite to the ability to produce control software that transfers from a specific model to reality.

With the notion of pseudo-reality, we create an artificial, simulation-only reality gap, and we investigate whether performance drop and rank inversion are to be necessarily ascribed to the fact that control software is evaluated in a context that is more complex than the one in which it is designed. We do so with a procedure that has the logical structure of a *reductio ad absurdum*: we show that, in the light of empirical results we produce, the complexity assumption leads to a contradiction—notably, that one model should be more and less complex than the other, at once. The procedure comprises two stages. In the first stage, we reproduce qualitatively, with simulation-only experiments, the results of Francesca et al. (2014b): they observed a rank inversion when comparing control software generated by two off-line design methods.

In their experiments, the authors generated control software on the basis of a model, which we shall call M_A, and assessed its performance in reality. In our simulation-only experiment, we also generate control software on the basis of the same model M_A, but we use a second model, which we shall call M_B, as a replacement of reality; a pseudo-reality. Here, we choose M_B by trial and error so that, when used as pseudo-reality to evaluate control software generated on M_A, a rank inversion occurs between the same off-line design methods studied by Francesca et al. (2014b). As designing on M_A and evaluating on M_B produces performance drop and rank inversion, if we were to accept the complexity assumption, we would conclude from this first stage that M_B is more complex than M_A.

In the second stage, we invert the roles of the two models: we automatically design control software on M_B and we evaluate it on the pseudo-reality M_A. Also in this second stage, we observe performance drop and rank inversion that are qualitatively similar to those reported by Francesca et al. (2014b). If we were to accept the complexity assumption, we would conclude that M_A is more complex than M_B. The clear contradiction between the conclusions of the first and second stage disproves the complexity assumption: it is not necessary to assume that the effects of the reality gap manifest because control software designed in simulation is evaluated in a more complex (pseudo-)reality.

4.1 Experimental setup

In the following, we provide details on the experimental protocol, we characterize M_A and M_B by giving the values of their parameters, and report the results. The instances of control software produced, the raw data obtained, p-values resulting from statistical tests, and videos illustrating the behaviors displayed by the control software are available as on-line supplementary material (Ligot and Birattari, 2022c).

4.1.1 Protocol

We consider two missions described below: AggregationXOR and Foraging. For each mission, we define an objective function to be maximized. The same objective function is used for both designing control software and assessing its performance. We run experiments in which the control software is designed by the same two design methods used in the previous chapter: `EvoStick` and `Chocolate`. Their description can be found in Section 3.3.

For each mission, we consider two stages: S_{AB} and S_{BA}—see Figure 4.1 for an illustration. In stage S_{AB}, each design method produces control software via simulations based on model M_A; the control software is then assessed with simulations based on model M_B. To study the generalization capability of the control software produced, the performance evaluated on model M_B is compared to the one evaluated on model M_A. In stage S_{BA}, the roles of the two models are inverted: control software

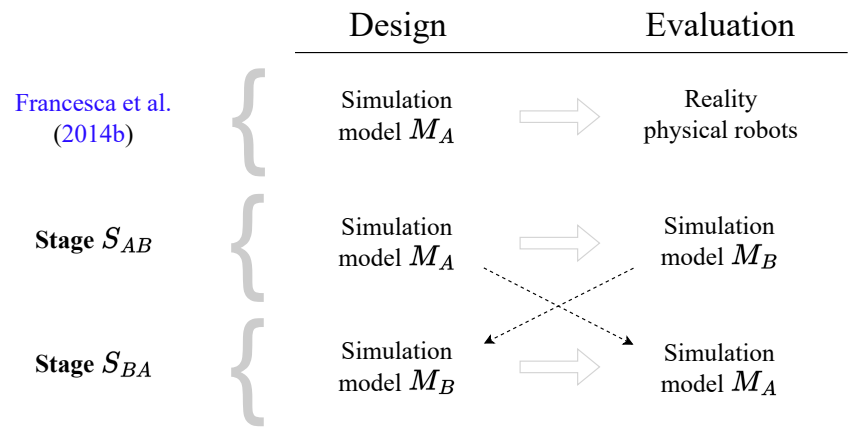


Figure 4.1: **Schematic overview of the study**. In stage S_{AB}, control software is generated on the basis of M_A and evaluated on the pseudo-reality model M_B. Model M_B was chosen via trial and error so that stage S_{AB} results in performance drop and rank inversion that are qualitatively similar to those observed by Francesca et al. (2014b) when they generated control software on the basis of the same simulation model M_A and evaluated it on physical robots. In stage S_{BA}, we invert the roles of the two models: control software is generated on the basis of M_B and evaluated on the pseudo-reality model M_A.

is produced on M_B and then assessed on M_A. Also in this case, the performance on M_A is compared to the one on M_B to study the generalization capability of the control software. In other terms, in stage S_{AB} the pseudo-reality is model M_B; whereas in stage S_{BA}, it is model M_A.

Each design method is run with a design budget of 200 000 simulations. For each mission and each stage S_{xy}—where by x and y we indicate A and B, or viceversa—each design method is run 20 times on model M_x and produces therefore a total of 20 instances of control software. For the assessment, each of these instances is evaluated 20 times on model M_x, and 20 times on model M_y to study their generalization capability.

We present the results by means of box-and-whiskers boxplots, and statements such as "method 1 is significantly better/worse than method 2" imply that significance has been assessed via a paired Wilcoxon signed rank test, with confidence of at least 95%. Moreover, to estimate the performance drop experienced in pseudo-reality, we present 95% confidence intervals, also computed via the paired Wilcoxon signed rank test. Finally, we use the Pearson correlation test to study the statistical relationship between performance drop experienced in pseudo-reality and performance on the design model.

Table 4.1: **The two e-puck models $\mathbf{M_A}$ and $\mathbf{M_B}$.** The values correspond to the parameters of ARGoS3 controlling the noise applied to the actuator values and sensor readings.

Actuator/Sensor	parameter	M_A	M_B
Wheels	p_g	0.05	0.15
Proximity	p_u	0.05	0.05
Light	p_u	0.05	0.90
Ground	p_u	0.05	0.05
Range-and-bearing	p_{fail}	0.85	0.90

4.1.2 Models

As discussed in Chapter 2, it is considered to be good practice to inject noise in the sensors and actuators of the simulated robots (Miglino et al., 1995; Jakobi et al., 1995). In the ARGoS3 simulator (Pinciroli et al., 2012), a uniform white noise is applied to the readings of the proximity, light, and ground sensors of the e-puck robot. A parameter p_u controls the level of noise: at every control cycle, for each sensor, a real value in the range $[-p_u, p_u]$ is uniformly sampled and added to the reading. A Gaussian white noise is applied to the velocities of each wheel and parameter p_g controls the level of noise: at every control cycle, for each wheel, a value is sampled according to a Gaussian distribution with mean 0 and standard deviation p_g, and added to the velocity. Finally, for the range-and-bearing module, a robot fails to estimate the relative position of a neighboring peer with probability p_{fail}.

We use the two e-puck models, namely M_A and M_B, described in Table 4.1. Model M_A is the same model used during the design process of the experiments ran by Francesca et al. (2014b). We generated model M_B by modifying actuator and sensor noise of model M_A. We did so via trial-and-error so that, when model M_B is used as a pseudo-reality to assess the performance of control software automatically generated on the basis of model M_A, we obtain a rank inversion that qualitatively resembles the one observed by Francesca et al. (2014b).

4.1.3 Missions

We consider two missions: AggregationXOR and Foraging. These missions must be performed by a swarm comprising 20 e-puck robots in a dodecagonal arena of 4.91 m^2 within a time of 250 s. At the beginning of an experimental run, we randomly position and orient the robots uniformly in the arena. Figure 4.2 depicts the simulated arenas for each mission.

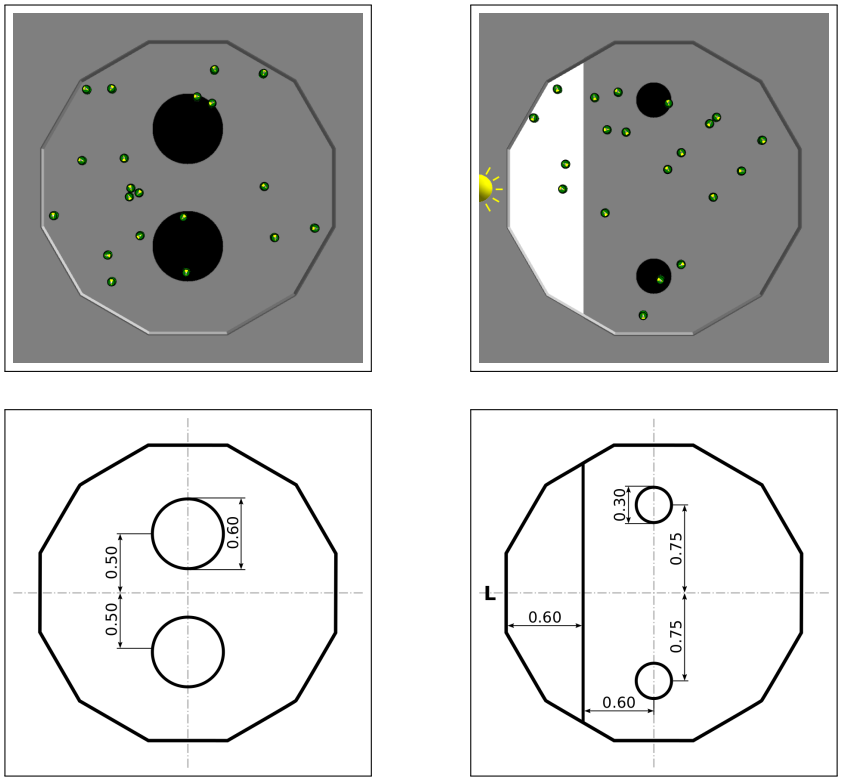


Figure 4.2: **ARGoS3 representations and technical diagrams of the arenas**. *Left*: AggregationXOR. *Right*: Foraging. Measures are expressed in meters. For Foraging, a light is placed behind the nest so that it is visible from everywhere in the arena.

AggregationXOR

The robots must aggregate on one of the two black zones, namely a or b. The size and position of the black zones are given in Figure 4.2. The objective function, to be maximized, is

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

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

Foraging

We consider an idealized version of foraging in which the robots must retrieve to the nest as many items as possible from either of two sources. The food sources are represented by black circular zones, and the nest by a white zone. A robot is considered to have picked up or dropped an item when it enters a black zone or the white zone,

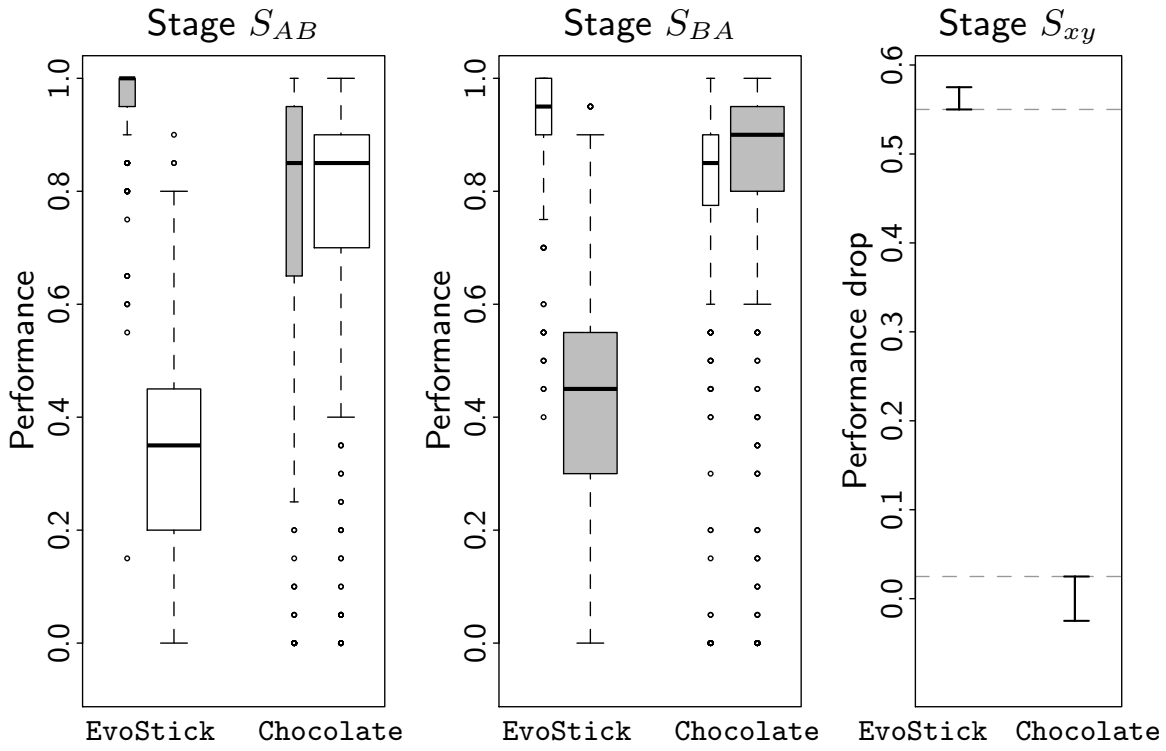


Figure 4.3: **Results for AggregationXOR mission**. *Left* and *center*: Performance in each stage—the higher, the better. Narrow boxes represent the performance assessed on the design model M_x; wide boxes represent the performance assessed on the pseudo-reality M_y. Gray boxes represent performance assessed on M_A; white boxes represent performance assessed on M_B. *Right*: Performance drop, aggregated across the two stages—the lower, the better. The segments represent the upper and lower bounds on the performance drop experienced in pseudo-reality—bounds are computed using Wilcoxon statistics, at 95% confidence.

respectively. A robot can only carry one object at a time. A light is placed behind the nest at a height of 0.75 m so that it is visible from everywhere in the arena. The size and position of sources and nest are given in Figure 4.2.

The objective function, to be maximized, is

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

where I is the number of items retrieved.

4.2 Results

In both stages S_{AB} and S_{BA}, we observe a noticeable performance drop for `EvoStick`, which determines a rank inversion—see Figure 4.3 and 4.4. Indeed, when comparing

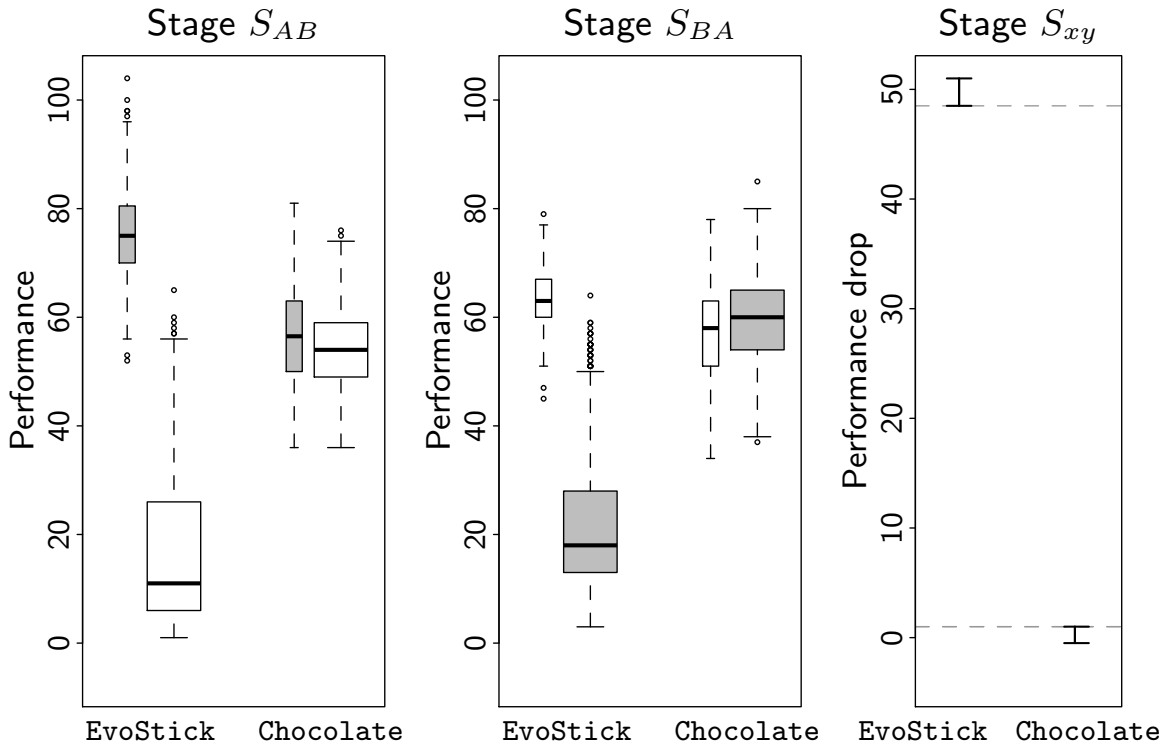


Figure 4.4: **Results for FORAGING mission**. *Left* and *center*: Performance in the two stages—the higher, the better. *Right*: Performance drop, aggregated across the two stages—the lower, the better. See caption of Figure 4.3 for a detailed description of the figures.

the two design methods on the basis of their performance assessed on model M_x—that is, the design model—`EvoStick` performs significantly better than `Chocolate`. However, when comparing the methods on the basis of their performance assessed on model M_y—that is, the pseudo-reality—`Chocolate` perform significantly better than `EvoStick`.

For AGGREGATIONXOR, a same instance of control software generated by `EvoStick` behaves in a qualitatively different way in M_x and in M_y—see Figure 4.5 (*top left*) for an illustration. In M_x, the robots tend first to navigate along the walls of the arena, and then to converge towards peers that are already located on one of the black zones. Once robots are on a black zone, they remain there, spinning in place. In M_y, the robots navigate in small circles without ever converging towards their peers. Eventually, the majority of the robots fail to find the black zones and to aggregate therein. Quantitatively, the performance drop that affects the control software generated by `EvoStick` is of at least 0.55—see Figure 4.3 (*right*). On the other hand, the control software produced by `Chocolate` behaves in a qualitatively similar way in M_x and M_y: the robots converge towards their peers to form clusters, and tend to remain on a black zone once they reach one—see Figure 4.5 (*bottom left*)

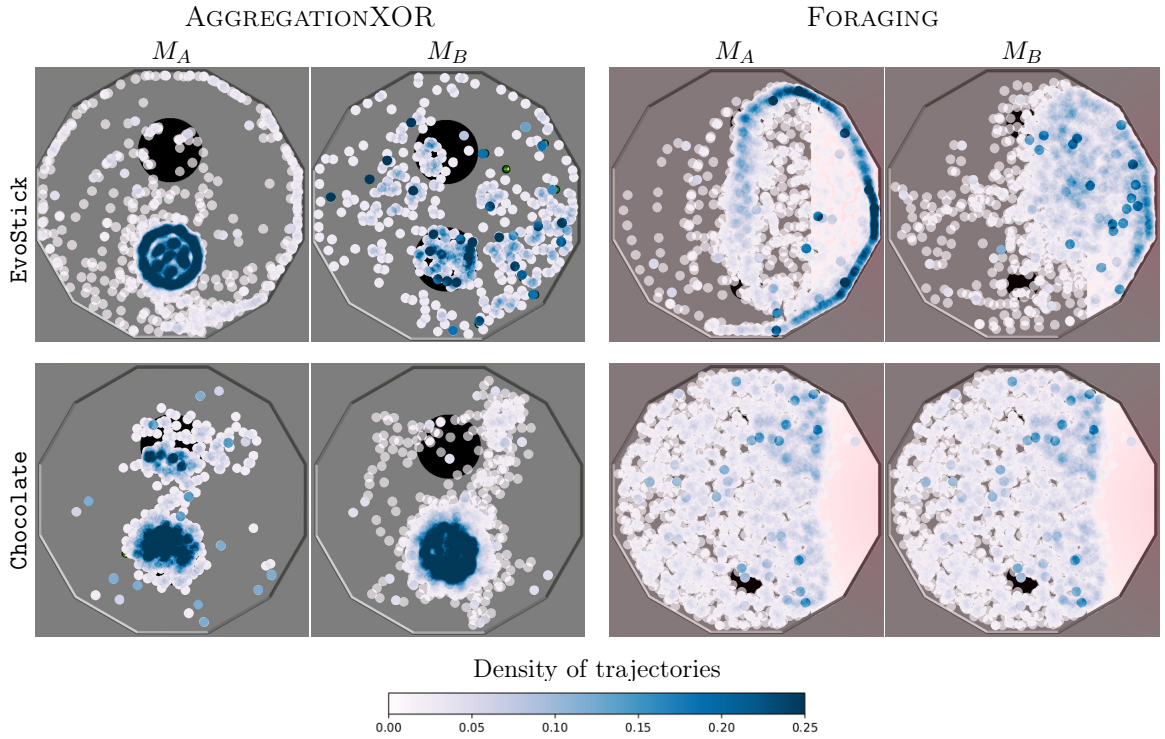


Figure 4.5: **Examples of trajectories of robots in $\mathbf{M_x}$ and $\mathbf{M_y}$.** For the two missions, the traces represent the execution of an instance of control software produced by `EvoStick` and `Chocolate` generated on the basis of M_A and evaluated in M_A (left-hand side) and in M_B (right-hand side). The darker the color of the spots, the longer a robot spend on that position.

for an illustration. Quantitatively, the performance drop is much smaller than the one experienced by `EvoStick`: at most 0.02—see Figure 4.3 (*right*).

The results for Foraging are similar to those of AggregationXOR. Indeed, the behavior of a same instance of control software generated by `EvoStick` is qualitatively different in M_x and M_y. In M_x, the robots navigate in circles of radius approximately equal to half the one of the whole arena: they follow the walls around the nest and cross the arena so that they navigate on at least one of the two food sources—see Figure 4.5 (*top right*) for an illustration. In M_y, the robots navigate in much smaller circles that often do not cross any of the food sources, which results in a drop of performance of at least 48 items—see Figure 4.4 (*right*).

Contrarily to `EvoStick`, `Chocolate` produces control software that behaves similarly in M_x and M_y: the robots randomly explore the gray area of the arena and, as soon as they encounter one of the food sources, they navigate towards the light to reach the nest—see Figure 4.5 (*bottom right*) for an illustration. The performance drop experienced by `Chocolate` is at most 1 item—see Figure 4.4 (*right*).

The results also show a positive correlation between the performance drop ex-

perienced in pseudo-reality and the performance assessed on the design model—see Figure 4.6. This holds true for the two design methods. Nonetheless, some difference should be noticed. For `EvoStick` and in the two missions, observations are concentrated in the top left quarter, which indicates a high performance on M_x, but a large performance drop. On the other hands, for `Chocolate` the observations are close to the center of the figure, which indicates a relatively lower performance on M_x with respect to the one of `EvoStick`, but a performance drop that is centered around zero.

4.3 Discussion

With this chapter, we shed further light on the reality gap. Specifically, we investigated how and under what conditions the effects of the reality gap manifest. We contend that, for the effects of the reality gap to manifest, it is unnecessary to assume that the control software is assessed under context/conditions that are more complex than those experienced in the design.

To substantiate our contention, we conceived a set of simulation-only experiments in which we created an artificial reality gap based on two robot models M_A and M_B. We used M_A for the design and M_B for the assessment; we then inverted the role of the two models. In both cases, we observed performance drop and rank inversion: a design method (`EvoStick`) performed significantly better than another (`Chocolate`) when the control software they produced was assessed on the same model used in the design, but significantly worse on the other one. Having observed performance drop and rank inversion both when (i) designing on M_A and assessing on M_B, and when (ii) designing on M_B and assessing on M_A, we can exclude that the effects of the reality gap emerge only due to the fact that the design is performed on a simplistic model that fails to reproduce the complexity of the environment in which the final assessment is performed.

The results of stage S_{AB}—where control software designed on the basis of M_A, which has been used as design model for generating control software for robot swarms in several studies (Francesca et al., 2014b; Birattari et al., 2016; Francesca et al., 2015; Hasselmann and Birattari, 2020; Kuckling et al., 2018; Ligot et al., 2020a; Hasselmann et al., 2021), were evaluated on the pseudo-reality model M_B—indicate that simulation-only experiments could be used to tell whether and to what extent automatic design methods are prone to performance drop and rank inversion, and eventually to predict real-world performance of control software. Here, we created M_B by hand via a trial-and-error approach so as that, when used as pseudo-reality, we obtained effects of the reality gap that resembles the one previously observed in reality. The trial-and-error approach we use is labor intensive and not easily reproducible, and thus inappropriate to be adopted in a methodology dedicated to validating control software or to predicting its real-world performance. In the Chapter 5, we discuss the generation of an appropriate pseudo-reality in an automatic way.

Finally, the results of stage S_{AB} might lead one to assume that the pseudo-reality

Table 4.2: **Performance drop** experienced by the control software produced by `EvoStick` and `Chocolate` when designed on the basis of M_A and M_B and assessed on the physical robots. We report the performance drop across the two missions considered with 95% confidence interval. To aggregate across the missions, we normalized the performance drop with the performance obtained in simulation: they are computed as $\frac{P_s(i)-P_r(i)}{P_s(i)}$ for each instance of control software i, where P_s and P_r are the performance in simulation and reality, respectively. A normalized performance drop of 0.25 implies that the performance in reality is 25% lower than the one obtained in simulation.

Method	Design model	Mean normalized performance drop
`EvoStick`	M_A	0.81 [0.74,0.88]
`EvoStick`	M_B	0.86 [0.80,0.92]
`Chocolate`	M_A	0.29 [0.20,0.38]
`Chocolate`	M_B	0.25 [0.13,0.37]

model M_B is a more truthful representation of reality than M_A, and that automatically generating control software on the basis of M_A is a bad design choice. As mentioned in Chapter 2, a number of approaches to handle the reality gap are motivated by the working hypothesis that the more accurate the simulations, the smoother the transition to reality. Following this hypothesis, one could assume that designing control software on the basis of model M_B rather than on M_A would result in better performance in reality. We conduct an experiment to put this inference to the test: we evaluate control software generated by `EvoStick` and `Chocolate` on the basis of both M_A and M_B on a swarm of 20-epuck robots.

Results reveal that designing control software on the basis of M_B does not yield better performance on the physical robots than designing them on the basis of M_A—see Figure 4.7. In fact, effects of the reality gap occur to the same degree regardless of the model used during the design: the control software designed by `Chocolate` suffers from mild performance drops in reality, whereas the one designed by `EvoStick` suffers from important ones—see Table 4.2. These results indicate that M_A is not to be blamed for performance drops eventually experienced by some design methods. Rather, these results—together with those discussed in the previous section—substantiate the contention that the effects of the reality gap are due to the fact that design methods might overfit the model on the basis of which they operate, hence producing control software that is not robust to the differences of conditions experienced once ported on physical robots.

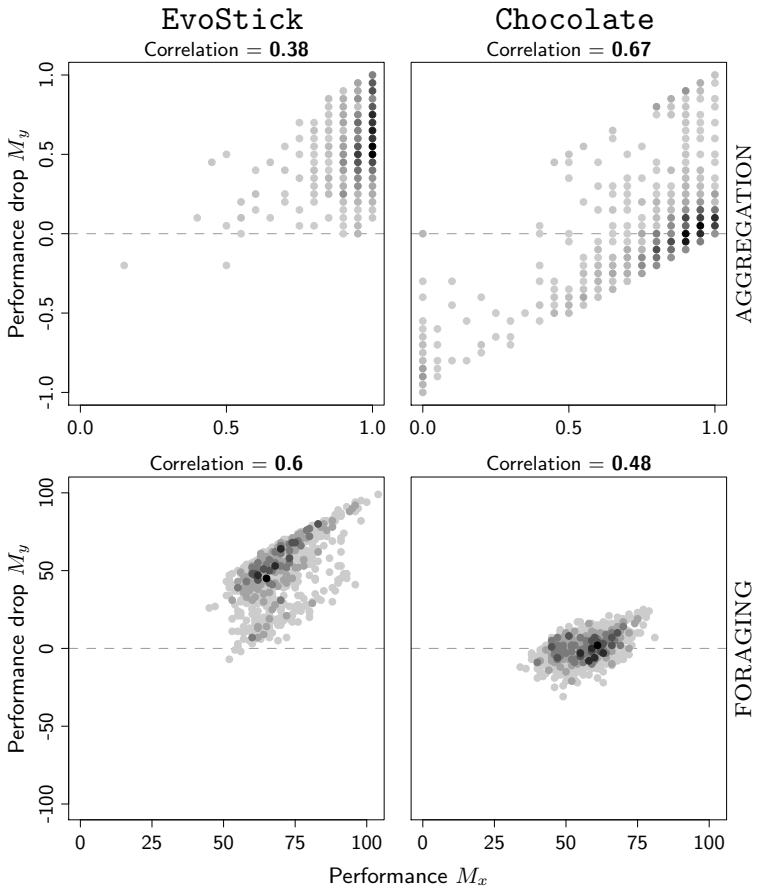
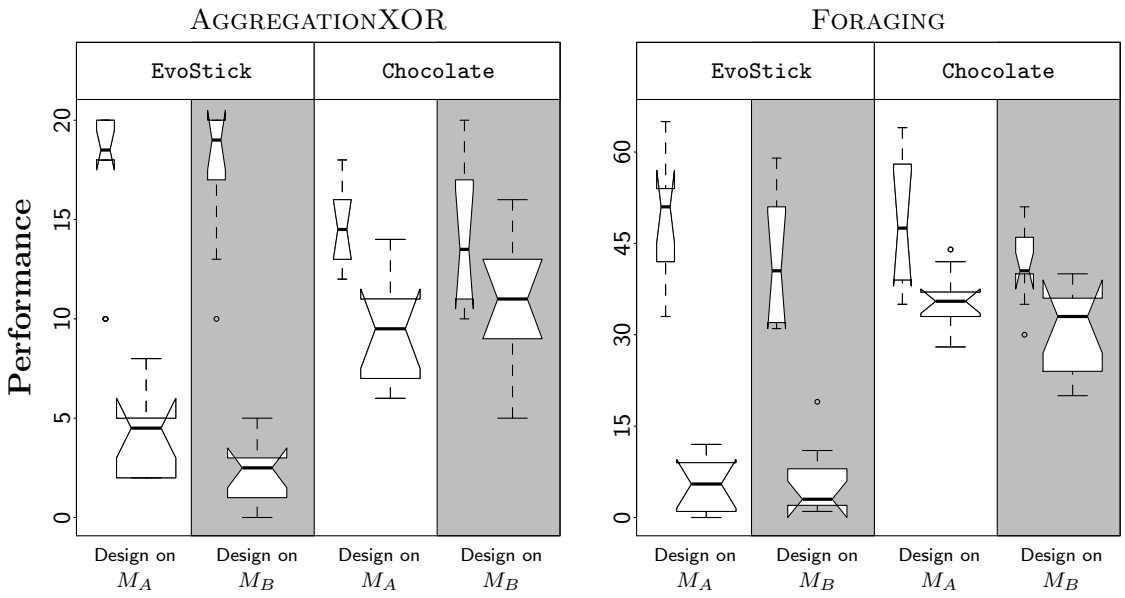


Figure 4.6: **Correlation between performance drop experienced in pseudo-reality and performance assessed on the design model.** The gray level of a point indicates its frequency of observation: the darker, the higher the frequency. All correlations are significantly different from 0 with confidence of at least 95%. For AggregationXOR, the organization of the points in columns is due to the quantum of F_{A} equal to $0.05 = 1/20$, which corresponds to a difference of one robot, out of the twenty comprised in the swarm, on the most populated black zone.



Source: Ligot A, Birattari M (2022b)

Figure 4.7: **Performance of control software automatically designed on the basis of M_A and M_B assessed on physical e-pucks.** The performance of `EvoStick` and `Chocolate` on AggregationXOR and Foraging, to be maximized, is represented by notched box-and-whiskers plots. Notches on the boxes represent the 95% confidence interval on the median, and allow for a convenient visual analysis of the results: if the notches of two boxes do not overlap, the difference between the two boxes is significant with a confidence of at least 95% (Chambers et al., 1983). White areas represent the performance of control software designed on the basis of model M_A; gray areas represent the one of control software designed on model M_B. Narrow boxes represent the performance of the control software obtained in simulation, that is, evaluated on the model M_x used during the design; wide boxes represent the performance of the same control software in reality.

Chapter 5

Sampling pseudo-reality models

In Chapter 4, we were able to find, by trial and error, a pseudo-reality that produced a performance drop and rank inversion that are similar to those previously observed when evaluating automatically generated control software on physical robots (Francesca et al., 2014b). These results suggest that one could use an artificial, simulation-only reality gap to conceive a methodology to predict the robustness of design methods to the reality gap. This methodology would be able to predict which of the design methods under analysis is more likely to produce control software that will suffer from a performance drop; whether the observation of a rank inversion is to be expected; and, eventually, which of the design methods under analysis is more likely to generate control software that successfully performs a given real-world mission. The trial-and-error approach we used in Chapter 4 to create a pseudo-reality is clearly inappropriate to be adopted in the framework of the methodology we have in mind. The approach is indeed labor intensive and not easily reproducible. To support the methodology, we need a way to generate an appropriate pseudo-reality in an automatic and reliable way.

In this chapter, we move a step in the direction of creating a pseudo-reality automatically. We investigate whether the results of Chapter 4 can be observed with other pseudo-realities, or whether they are an artifact of the specific pseudo-reality we employed there. We do so by reproducing the two-stage simulation-only experiment of the previous chapter, but this time using multiple evaluation models—and therefore, multiple pseudo-realities—to assess control software generated on the basis of model M_A. The different pseudo-realities are uniformly sampled from a range or predefined set of models build around M_A. We call this range of models R. In other words, we create multiple artificial reality gaps between pairs of models: M_A and a randomly sampled one. Because some of the sampled models might be too similar to M_A to yield a noticeable performance drop, we do not expect that every single model sampled can be used by itself as a pseudo-reality on which we can observe results analogous to those observed in Chapter 4. Nonetheless, we expect that, by evaluating control software on a sufficiently large number of such models, and by aggregating the results across all of them, we could obtain a correct overall picture of which methods are more likely to

suffer from a performance drop and of whether a rank inversion should be expected.

Finally, we propose multiple measures to quantify the *width* of the artificial reality gap between a pair of models. By width of a reality gap, we mean some measure of the difference between the model on which control software is designed and the (pseudo-)reality in which it is evaluated.

5.1 Experimental setup

In the following, we detail the protocol followed, we describe the set of models from which the pseudo-realities are sampled, and report the results. We also define the measures of difference between models, and report and discuss the correlation between width of pseudo-reality gap and performance drop.

5.1.1 Protocol

We consider two stages: S_{AR_1} and S_{R_1A}. In stage S_{AR_1}, an instance of control software is automatically generated on the basis of model M_A, and it is evaluated on M_A itself and on a pseudo-reality: one model uniformly sampled from the range R defined in the following section—we refer to that one model sampled from R as R_1. This process is repeated 20 times, which therefore results in the generation of 20 instances of control software on the basis of M_A and in the sampling of 20 models R_1. In stage S_{R_1A}, the same 20 models R_1 sampled in stage S_{AR_1} are used to automatically generate control software. For each design method and mission, a single instance of control software is generated on the basis of each R_1 model. This instance of control software is then evaluated once on the same R_1 used for the design, and once on a pseudo-reality, whose role here is played by M_A. We consider once again the missions AggregationXOR and Foraging, and the design methods `EvoStick` and `Chocolate` with a design budget of 200 000 simulation runs. The 20 uniformly sampled models R_1 are the same in each stage, for each mission, and for each automatic design method.

5.1.2 Models

Model M_A is the same model used in Chapter 4. Models R_1 are sampled from a predefined set of models that we conceived such that (i) it comprises both model M_A and model M_B used in the previous chapter, and (ii) it contains models that are more noisy than M_A and models that are less noisy. The set of models from which R_1 are sampled is given in Table 5.1, together with the parameters of M_A and M_B, which are repeated here for the convenience of the reader.

Table 5.1: **The models $\mathbf{M_A}$, $\mathbf{M_B}$, and the ranges of possible values for models within the range R**. The values correspond to the parameters of ARGoS3 controlling the noise applied to the actuator values and sensor readings—see Section 4.1.2 for a description of the different noise parameters.

Actuator/Sensor	parameter	M_A	M_B	Range R
Wheels	p_g	0.05	0.15	$[0.00, 0.20]$
Proximity	p_u	0.05	0.05	$[0.00, 0.10]$
Light	p_u	0.05	0.90	$[0.00, 1.50]$
Ground	p_u	0.05	0.05	$[0.00, 0.10]$
Range-and-bearing	p_{fail}	0.85	0.90	$[0.70, 1.00]$

5.2 Measuring the width of an artificial reality gap

Because all models considered in this chapter differ only by the values of five parameters, each model is fully identified by a vector in a five-dimensional space. Under this condition, we conjecture that the width of an artificial reality gap can be quantified by an appropriate distance measure between the vectors that identify the models involved in the artificial reality gap itself. We consider a number of distance measures between two vectors and the difference of a number of vector norms. We study their Pearson correlation with the performance drop experienced when designing control software on the model identified by one of the two vectors and evaluating it on the model identified by the other one.

In this five dimensional space of the models, for a generic vector $\mathbf{v}$, we consider its norms ℓ^1, ℓ^2, and ℓ^∞, where

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

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

As possible measures of the distance between two models $\mathbf{x}$ and $\mathbf{y}$, we consider the differences $\|\mathbf{y}\|_n - \|\mathbf{x}\|_n$ between the ℓ^n norm of their corresponding vectors $\mathbf{x}$ and $\mathbf{y}$, and the ℓ^n norm of their difference $\|\mathbf{y} - \mathbf{x}\|_n$, both with $n \in \{1, 2, \infty\}$. Differences of norms can be negative, with a negative value indicating that the evaluation model is less noisy than the design one. Moreover, differences of norms are anticommutative: for example, the width of the gap from M_A to M_B and the one from M_B to M_A have the same absolute value but opposite sign. It should be noted that a null value of

a difference of norms does not ensure that the two models are identical. This is a weakness of differences of norms as a measure of the width of the reality gap because one would expect that a zero measure indicates that the gap is null and therefore design and evaluation models are the same. On the other hand, norms of difference are non-negative and commutative: the larger the norm, the wider the gap; and, for example, the width of the gap from M_A to M_B and the one from M_B to M_A are equal.

As an alternative to measuring the width of an artificial reality gap, one could measure how similar the design and evaluation models are. For example, this could be done using the *cosine similarity* of models $\mathbf{x}$ and $\mathbf{y}$:

$$\frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\ \sqrt{\sum_i y_i^2}}.$$

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

For all the aforementioned measures, we consider both the unnormalized and normalized versions. We normalize each term v_i of a vector $\mathbf{v}$ with respect to the lower and upper bounds of its range—L_i and U_i, respectively. Ranges are given in Table 5.1. In particular, when a vector $\mathbf{m} = \{m_1, ..., m_5\}$ represents a model, it is normalized into a vector $\overline{\mathbf{m}} = \{\overline{m}_1, ..., \overline{m}_5\}$ where:

$$\overline{m}_i = \frac{m_i - L_i}{U_i - L_i} \qquad \text{for } i \in \{1, ..., 5\}. \tag{5.1}$$

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

$$\overline{d}_i = \begin{cases} \dfrac{b_i - a_i}{U_i - a_i}, & \text{if } b_i >= a_i; \\ \dfrac{b_i - a_i}{a_i - L_i}, & \text{if } b_i < a_i; \end{cases} \qquad \text{for } i \in \{1, ..., 5\}. \tag{5.2}$$

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

5.3 Results

The results of this experiment are qualitatively similar to those observed in Chapter 4: in both stages and for both missions considered, a rank inversion occurred between `EvoStick` and `Chocolate`—see Figures 5.1 and 5.2. Indeed, the control software generated by `EvoStick` performs significantly better than the one of `Chocolate` when the evaluation is performed on the design model, but the one of `Chocolate` performs

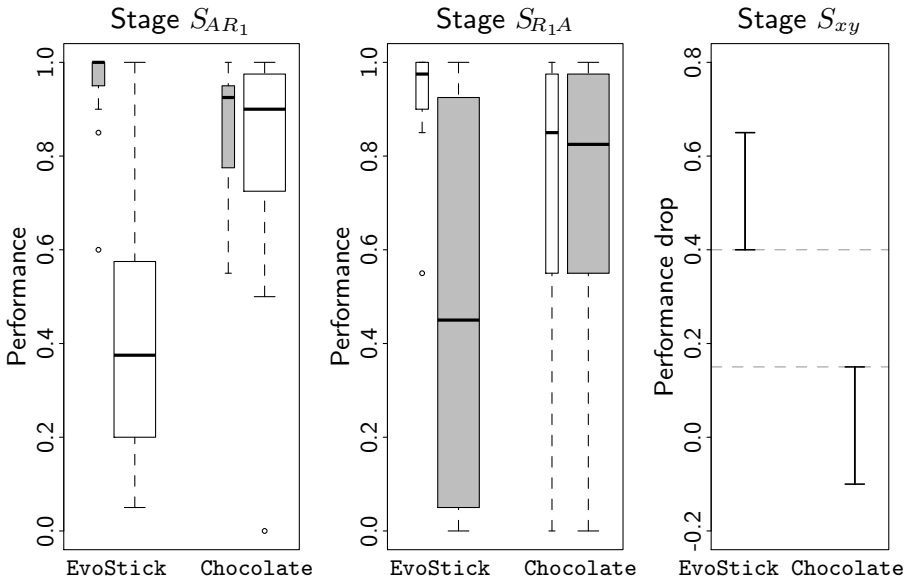


Figure 5.1: **Results for AggregationXOR mission**. Narrow boxes represent the performance assessed on the model used as design context; wide boxes represent the performance assessed in pseudo-reality. Gray boxes represent performance assessed on model M_A; white boxes represent performance assessed on models R_1 sampled from the range *R Right*: Performance drop, aggregated across the two stages—the lower, the better. The segments represent the upper and lower bounds on the performance drop experienced in pseudo-reality—bounds are computed using Wilcoxon statistics, at 95% confidence.

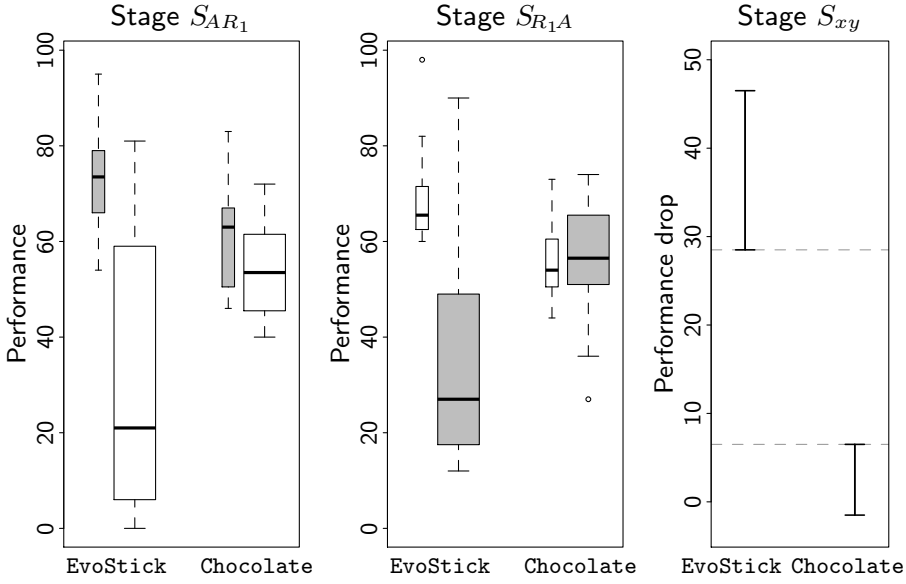


Figure 5.2: **Results for Foraging mission**. See caption of Figure 5.1 for a detailed description.

significantly better than that of `EvoStick` when the evaluation is performed in pseudo-reality. The control software generated, the models sampled, and the raw data obtained are available as on-line supplementary material (Ligot and Birattari, 2022c).

Concerning the correlation between performance drop and the width of the artificial reality gap, all measures we considered, both in the normalized and unnormalized versions, provided similar results. In the following, we only report the results concerning the normalized and unnormalized version of the ℓ^1 norm of difference, the difference of ℓ^1 norms, and the cosine similarity because trends are more apparent there. Figure 5.3 shows the correlation between the performance drop—aggregated across stages S_{AR_1} and S_{R_1A}—and the unnormalized version of the difference of ℓ^1 norms, the ℓ^1 norm of differences, and the cosine similarity. Figure 5.4 shows the correlation between the performance drop and the normalized version of these three measures. Additional figures, displaying the correlation between the performance drop and both the unnormalized and normalized versions of the other measures of the width of the pseudo-reality gaps, are available as on-line supplementary material (Ligot and Birattari, 2022c).

In all figures, a disparity can be noticed between the performance drop experienced by `EvoStick` and `Chocolate`: for `EvoStick`, most observations are above the zero-drop line; for `Chocolate` they are more equally spread above and below. Note that a positive performance drop indicates that the performance in pseudo-reality is worse than the one obtained on the design model, whereas a negative drop indicates that the performance in pseudo-reality is better than the one obtained on the design model. Moreover, the results for AggregationXOR fail to show a clear trend. For Foraging, differences of norms display a V-shape pattern. This is particularly evident for `EvoStick`—see Figure 5.3a (*bottom left*). This is possibly due to the anticommutative nature of the measure. Because of the V-shape pattern, the Pearson correlation fails to be informative on the actual correlation between width of the reality gap and performance drop. Nonetheless, the difference of norms has some merits as it highlights an interesting fact: for `EvoStick`, the performance drop grows with the absolute value of the width. In particular, we can observe that for negative width—that is, when the evaluation model is less noisy than the design one—we register a noticeable performance drop. This further corroborates our conjecture that performance drop is to be explained as a sort of overfitting of the conditions experienced during the design, rather than as the result of evaluating in a complex (pseudo-)reality control software that has been designed on the basis of a simplistic simulation model.

Norms of differences, which are commutative and effectively fold negative widths onto positive ones, highlight a correlation between width of the reality gap and performance drop—see Figure 5.3b (*bottom*). Pearson correlation is significant both for `EvoStick` and `Chocolate`, but is larger for `EvoStick`. Also the cosine similarity highlights a correlation between width of the reality gap and performance drop—more precisely, a negative correlation between similarity of the design and evaluation models and performance drop—see Figure 5.3c (*bottom*). In this case, the Pearson correlation is significant only for `EvoStick`.

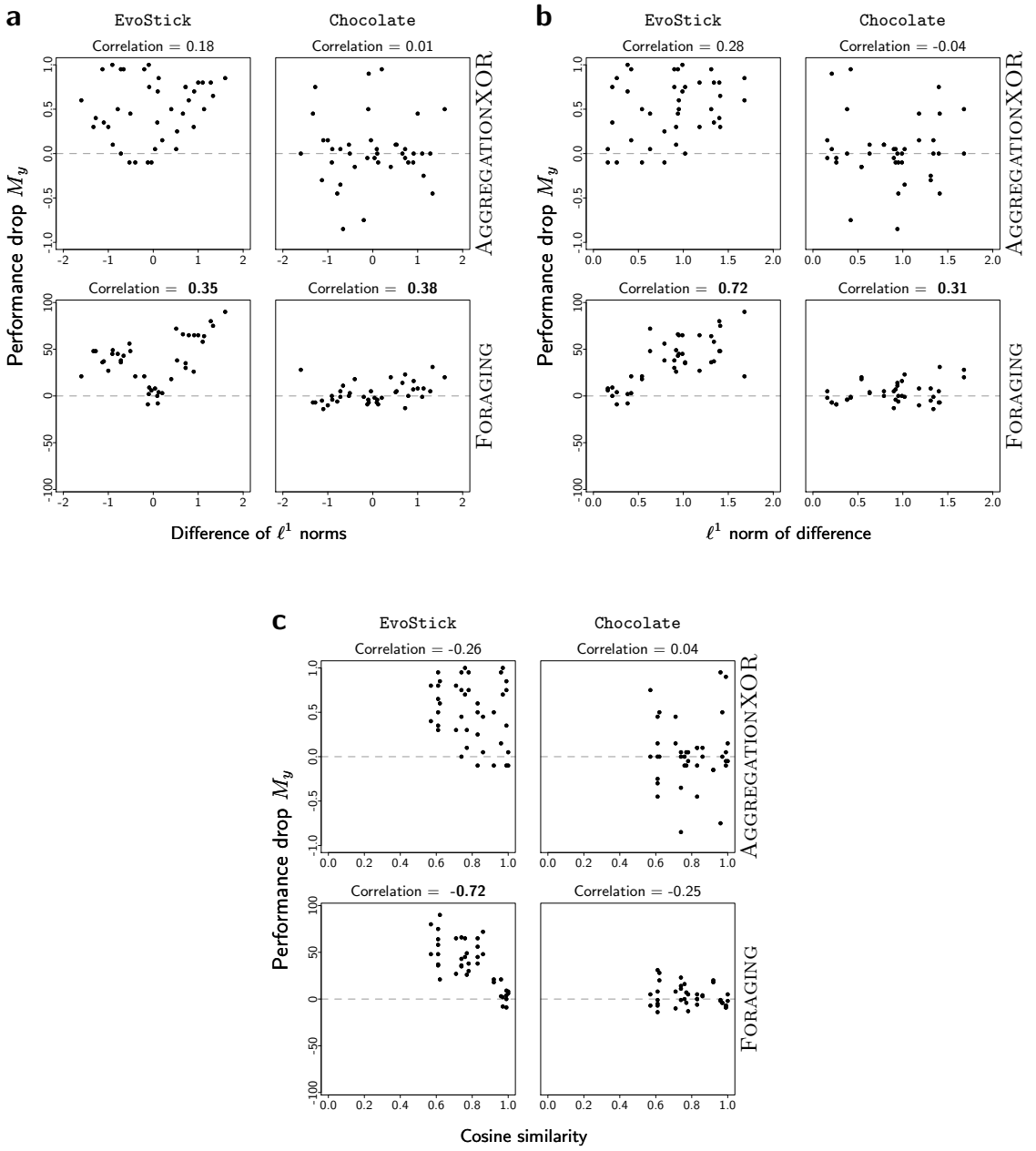


Figure 5.3: **Pearson correlation between performance drop and unnormalized measures of the width of the gaps created between the models used for design and evaluation**. (**a**) difference of ℓ^1 norms, (**b**) ℓ^1 norm of difference, (**c**) cosine similarity. Performance drop is aggregated across stages S_{AR_1} and S_{R_1A}. Boldface values indicate that the correlation is significantly different from 0 with confidence of at least 95%.

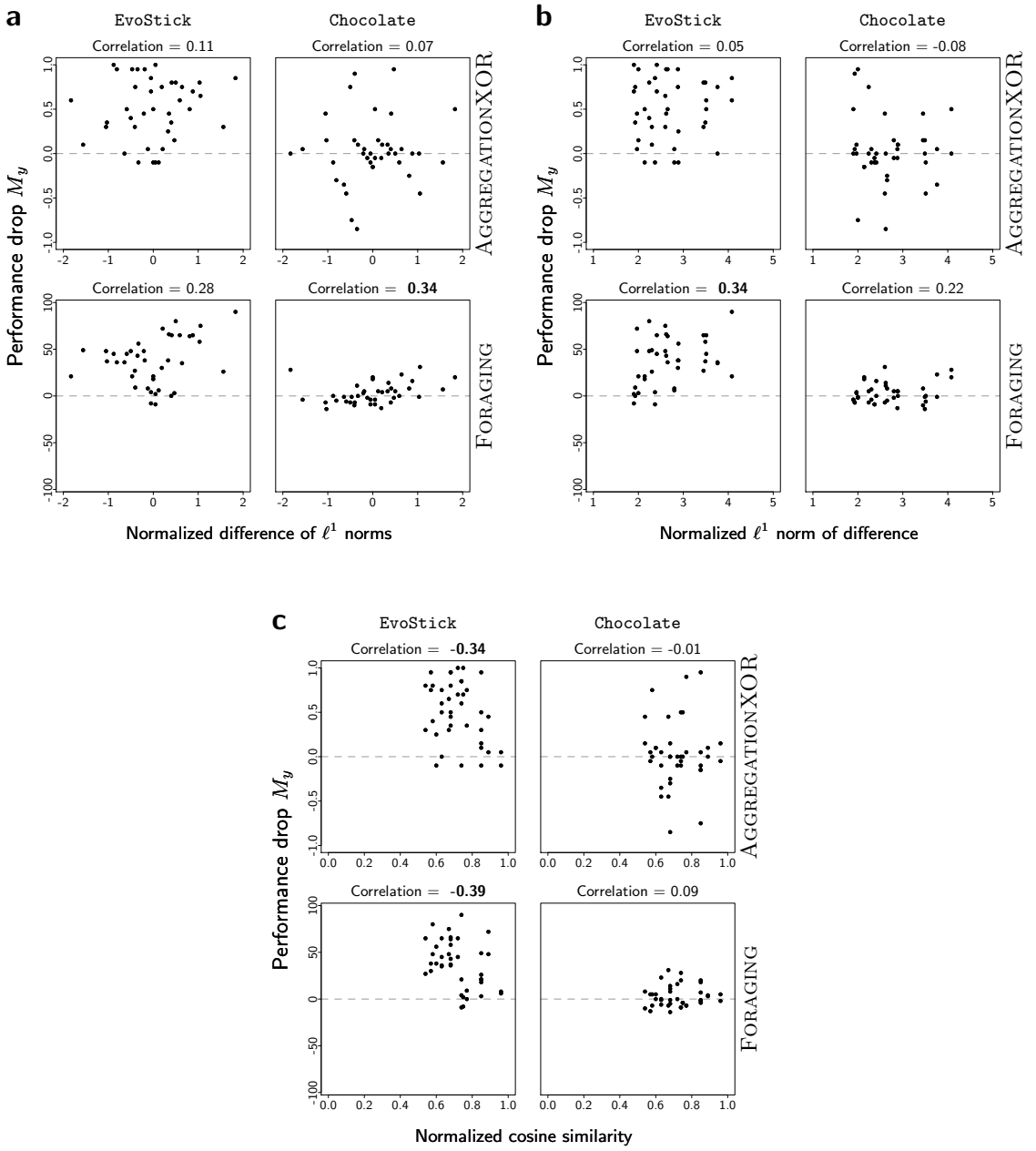


Figure 5.4: **Pearson correlation between performance drop and normalized measures of the width of the gaps created between the models used for design and evaluation**. (**a**) difference of ℓ^1 norms, (**b**) ℓ^1 norm of difference, (**c**) cosine similarity. Performance drop is aggregated across stages S_{AR_1} and S_{R_1A}. Boldface values indicate that the correlation is significantly different from 0 with confidence of at least 95%.

In R, the range of possible values for the noise applied to the light sensor is much larger than the range of possible values for the other sensors and actuators. As a result, it is likely that the difference of noise values applied to the light sensor between two given models wipes out the differences between the noise values applied to other sensors and actuators when considering the unnormalized vectors corresponding to these models. Using the normalized vectors instead levels the differences of range sizes. In this case, the same trends observed for the unnormalized version of the measures of the width can be observed, yet these trends are less pronounced—see Figure 5.4. For example, the V-shape pattern displayed by the difference of norms for `EvoStick` on AggregationXOR is barely visible—see Figure 5.4a (*bottom left*). Moreover, the Pearson correlation is lower for most cases, with the exception of the one between the performance drop observed by `EvoStick` on AggregationXOR and the normalized cosine similarity——see Figure 5.4a (*top left*).

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

5.4 Discussion

In this chapter, we reproduced the experiment of Chapter 4, this time with multiple artificial reality gaps. Although the results of the experiment conducted in this chapter are similar to those observed in the previous chapter, they cannot be used to disprove the complexity assumption. In fact, because we created the range R around model M_A, it is likely that, in a same stage, some instances of control software were evaluated on models that were more noisy than the design model, and that other instances were evaluated on models that were less noisy. One could therefore argue that the performance drop and rank inversion observed in stage S_{AR_1} are due to the fact that some of the R_1 models used to evaluate the control software are more complex than the design model M_A, and that the performance drop and rank inversion observed in S_{R_1A} are due to the fact that some of the R_1 models used as design model are simpler than the evaluation model M_A. However, our analysis of the width of the artificial reality gaps showed that some of the noticeable performance drop we registered resulted from the evaluation of control software on models that were less noisy than the design model. This does further corroborate our conjecture that performance drop is to be explained as a sort of overfitting of the conditions experienced during the design, rather than as the result of fact that control software is designed on the basis of a simulation model that is less complex than the (pseudo-)reality in which it is eventually evaluated.

The results of the first stage, in which we automatically designed control software on the basis of M_A and evaluated it on models uniformly sampled from the range R, are particularly interesting. In fact, they show that it is not only possible to mimic effects of the reality gap using a single, handpicked evaluation model; it is also possible to do so using multiple ones sampled around the design model. Being able to

reproduce realistic performance drop that leads to observed rank inversions between design methods suggests that the concept of pseudo-reality could be used for assessing the robustness of design methods. A simulation-only procedure to reliably predict real-world performance would be highly valuable, if only to diminish the amount of expensive and time-consuming experiments with physical robots needed to assess control software. In the following chapter, we define predictors on the basis of the concept of pseudo-reality and we study their accuracy.

Chapter 6

Predictors of real-world performance

In Chapter 4, we showed that it is possible to create a virtual, simulation-only reality gap between the design model and a pseudo-reality model—which we named M_B—that yields a performance drop that is qualitatively similar to the one observed in reality. In Chapter 5, we also obtained results that were qualitatively similar to the ones observed on physical robots by using multiple virtual reality gaps between the design model and pseudo-reality models randomly sampled from a range R. Although promising, the evidence produced so far is insufficient to elaborate any claim about the accuracy of these pseudo-reality predictors, nor about their superiority with respect to the classical evaluations on the design model[1]. Indeed, in addition to the fact that we only reported qualitative results, these results were obtained on the same data used for the definition of the pseudo-reality predictors, which fails to communicate on their generalization capability. Here, we address these flaws: we propose quantitative metrics, and we perform a thorough investigation of whether the reliability of these pseudo-reality predictors generalizes to control software produced by a wider range of methods and on a wider range of missions.

To do so, we created DS 1, a dataset of performance of robot swarms assessed on physical robots (Ligot and Birattari, 2022a). We gathered publicly available data—that is, instances of control software and associated real-world performance—from several studies in optimization-based design of robots swarms (Francesca et al., 2015; Kuckling et al., 2018; Hasselmann et al., 2018b; Spaey et al., 2019; Ligot et al., 2020a; Hasselmann et al., 2021; Ligot et al., 2022). In total, we collected 1021 instances

[1]Evaluations of control software on the design model is typically considered as a natural way to estimate its real-world performance. In fact, roboticists commonly report the performance obtained on the design model, alongside the one observed on physical robots (when available), to show whether the control software crosses the reality gap satisfactorily or not. However, as discussed previously, reports of severe performance drop and rank inversion suggest that the performance observed on the design model does not yield an accurate and reliable prediction of the actual performance that one will eventually obtain in the real world.

of control software generated by 18 different off-line design methods for 45 missions. By reusing data collected previously for other purposes—which, to the best of our knowledge, is a premiere in the optimization-based design of control software for robot swarms—we were able to perform an analysis that would have been quite costly, should we had to generate the required data from scratch. Although they originate from several distinct studies, all these instances were designed automatically on the basis of the same ARGoS3 (Pinciroli et al., 2012) simulator model M_A and evaluated on swarms of e-puck robots (Mondada et al., 2009). In addition to the real-world performance of the collected control software, DS 1 also contains the predicted performance obtained by evaluating the collected control software in simulation on the design model M_A, the pseudo-reality model M_B, and 1380 pseudo-reality models uniformly sampled from the range R.

In the following, we consider evaluations on the models M_A and M_B to correspond to the process of obtaining performance forecasts by the predictors we refer to as P_{M_A} and P_{M_B}, respectively. Evaluations on the randomly sampled models allow us to repeatedly execute the process of the predictor P_{R_1} which consists, for each instance of control software, in sampling a model from the range of possible models R and evaluating the instance of control software on it, similarly to what we did in Chapter 5. It also allows us to define the predictor P_{R_k}, a generalized version of P_{R_1} that consists, for each instance of control software, in sampling k models from R and evaluating on each of them the instance of control software once. Details about DS 1, the predictors, the optimization-based design methods, and the missions for which the considered control software has been generated are given in Appendix B. Figure 6.1 gives a schematic overview of the study conducted in this chapter.

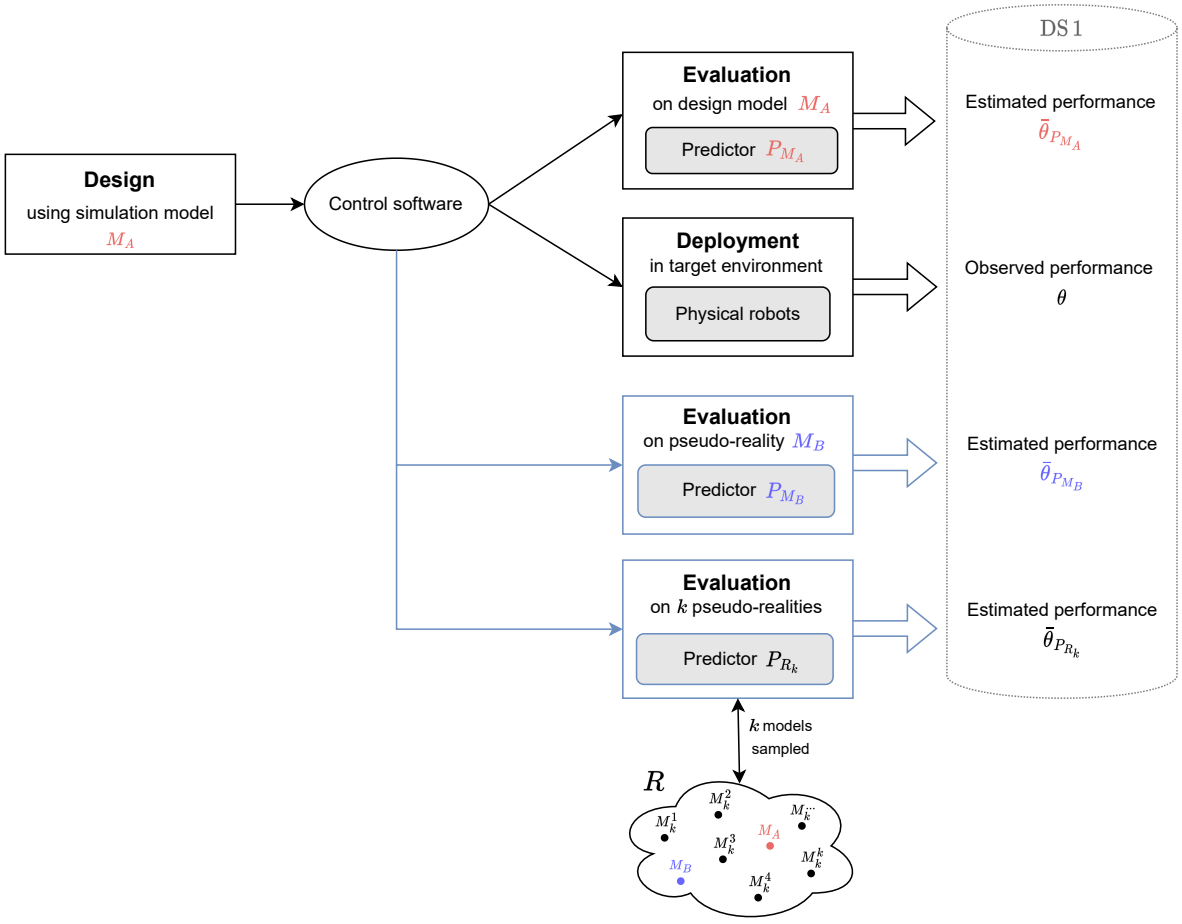
In the remainder of this chapter, we compare the predicted performance with the one observed on the physical robots, and we assess the accuracy of the predictors P_{M_A}, P_{M_B}, and P_{R_1} according to three evaluation criteria that we will present hereafter. We also do so for P_{R_k} with $k \in \{1, 3, 5, 10, 30, 50, 100, 500\}$.

6.1 Prediction of estimated performance: the error

The quantity *error* measures the accuracy of the predictions of the expected real-world performance of control software. We compute the normalized differences between the predicted performance $\bar{\theta}$ and the performance θ observed in reality, as reported in DS 1, as

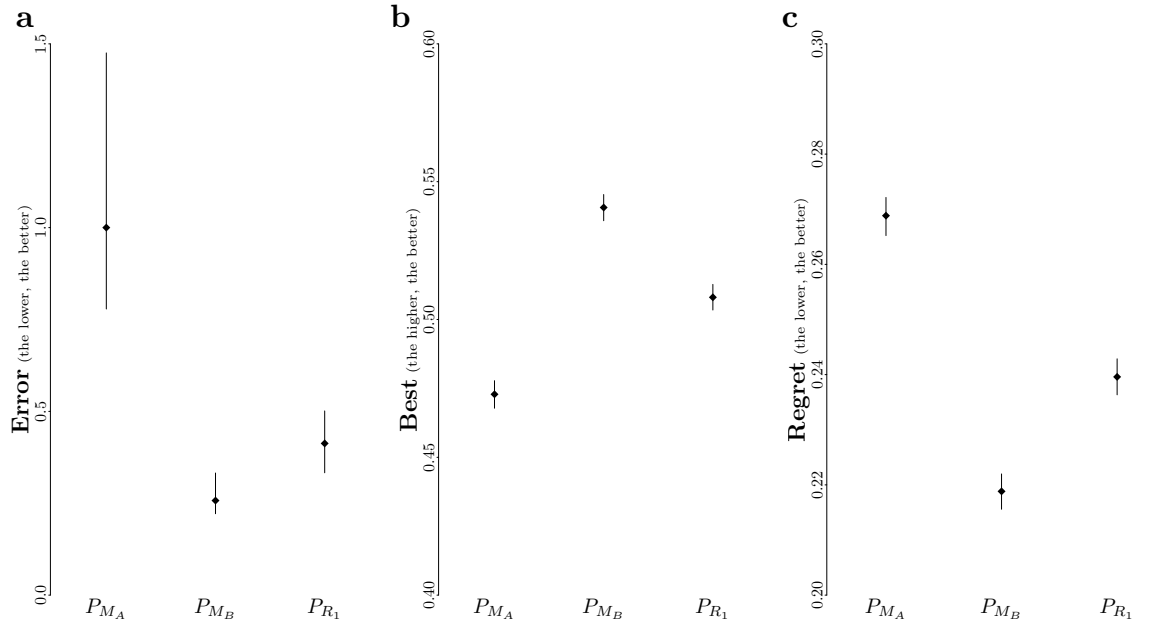
$$error = \left(\frac{\bar{\theta} - \theta}{\theta} \right)^2 . \tag{6.1}$$

Figure 6.2a reports the median *error* of the predictors P_{M_A}, P_{M_B}, and P_{R_1}. Results show that estimating the expected performance of control software on the basis of evaluations on the same simulation model used in the design process yields less accurate predictions than any of the two other pseudo-reality predictors. In fact, the median



Source: Ligot A, Birattari M (2022b)

Figure 6.1: **Schematic overview of the study**. In black, the typical process for generating control software for robot swarms consisting of a design phase, an evaluation phase, and eventually a deployment phase. The design phase is performed on the basis of a simulation model here named M_A. The evaluation phase is performed in simulation, on the same simulation model used during the design, and is a common way of predicting the real-world performance of control software. We name this popular predictor P_{M_A}. During the deployment phase, in which control software is executed on physical robots in the target environment, the actual performance θ of the control software is observed. In blue, our evaluation of the control software with pseudo-reality predictors. The predictor P_{M_B} evaluates each instance of control software on a single pseudo-reality model named M_B. The previously defined predictor P_{R_1} evaluates each of them once on a randomly sampled model from the set R, which contains both M_A and M_B. We introduce P_{R_k}, a generalized version of P_{R_1}, which evaluates each instance once on k randomly sampled models from R, and considered $k = \{1, 3, 5, 10, 30, 50, 100, 500\}$. The observed performance that we collected from previous studies, as well as the predicted performance given by the predictors we considered, are part of the dataset DS 1. See Appendix B for details about DS 1 and the predictors.



Source: Ligot A, Birattari M (2022b)

Figure 6.2: ***Error*, *best*, and *regret* of the predictors**. In each plot, points represent the (**a**) median *error* (to be minimized), (**b**) mean *best* (to be maximized), and (**c**) mean *regret* (to be minimized); lines represent the respective 95% confidence interval.

error of the predictor P_{M_A} is at least 2.4 times greater than the one of the other predictors: P_{M_A} obtains a median error of 1.0, P_{M_B} one of 0.26, and P_{R_1} one of 0.41. Although the two lines representing the 95% confidence interval of the median *error* of P_{M_B} and P_{R_1} are close, they do not overlap. P_{M_B} is therefore significantly more accurate than both P_{M_A} and P_{R_1}.

6.2 Prediction of best instance of control software: the best

The quantity *best* measures the ability of predictors to accurately predict the ranking of two given instances of control software—that is, the ability to predict which instance of control software performs better than the other in reality. To compute the *best*, we consider all 43 520 possible pairwise comparisons of instances of control software within the individual studies from which we collected them so as to ensure fair comparisons. For each possible pair of instances $\{\mathtt{X}, \mathtt{Y}\}$, we compute the *best* as

$$best = \begin{cases} 1, & \text{if } \operatorname{argmax}_{\mathtt{X}|\mathtt{Y}} \left(\bar{\theta}_{\mathtt{X}}, \bar{\theta}_{\mathtt{Y}}\right) = \operatorname{argmax}_{\mathtt{X}|\mathtt{Y}} \left(\theta_{\mathtt{X}}, \theta_{\mathtt{Y}}\right), \\ 0, & \text{otherwise.} \end{cases} \tag{6.2}$$

where $\theta_{\{\mathtt{X}|\mathtt{Y}\}}$ is the expected performance observed in reality, and $\bar{\theta}_{\{\mathtt{X}|\mathtt{Y}\}}$ is the one estimated by a predictor. For a pair of instances of control software, the *best* is 1 if a predictor correctly infers the better performing one; 0 otherwise.

We report in Figure 6.2b the mean *best* of each predictor over all possible pairs $\{\mathtt{X}, \mathtt{Y}\}$ of instances of control software, which is to be maximized. Results show that evaluations on M_A lead to the correct best performing method for less than 50% of the possible pairs as it obtains a *best* of 0.47. The accuracy of the other predictors is slightly above 50%, with a value of 0.54 for P_{M_B} and 0.51 for P_{R_1}. The improvement of P_{M_B} over P_{M_A} is of 14.4%, the one of P_{R_1} is of 7.4%. As the lines in Figure 6.2b representing the 95% confidence interval on the mean *best* do not overlap, the improvement of P_{M_B} and P_{R_1} over P_{M_A} is significant, and P_{M_B} is the most accurate predictor.

6.3 Impact of wrong predictions: the regret

The quantity *regret* measures the loss incurred due to wrong predictions of the best performing instance of control software of a given pair $\{\mathtt{X}, \mathtt{Y}\}$. The *regret* is computed as the difference between the real performance of the best performing instance of control software and the real performance of the instance predicted to be the best performing one. To aggregate the results obtained across different missions, we normalize the difference with the maximal performance observed in reality. For each possible pair of instances of control software $\{\mathtt{X}, \mathtt{Y}\}$, we compute the relative *regret* as

$$regret = \left(\max(\theta_{\mathtt{X}}, \theta_{\mathtt{Y}}) - \theta_{\mathrm{argmax}_{\mathtt{X}|\mathtt{Y}}(\bar{\theta}_{\mathtt{X}}, \bar{\theta}_{\mathtt{Y}})}\right) / \max(\theta_{\mathtt{X}}, \theta_{\mathtt{Y}}), \tag{6.3}$$

where $\theta_{\{\mathtt{X}|\mathtt{Y}\}}$ is the performance observed in reality, and $\bar{\theta}_{\{\mathtt{X}|\mathtt{Y}\}}$ is the one estimated by a predictor. For a given pair of instances of control software, if a predictor correctly determines which instance yields the best performance in reality, the relative *regret* is 0. Otherwise, the relative *regret* takes a value between 0 and 1, and it is an indicator of the impact of making a mistake in predicting the best performing instance: the larger the difference of real-world performance of two instances of control software, the larger the *regret*, and vice versa. The relative *regret* is therefore to be minimized.

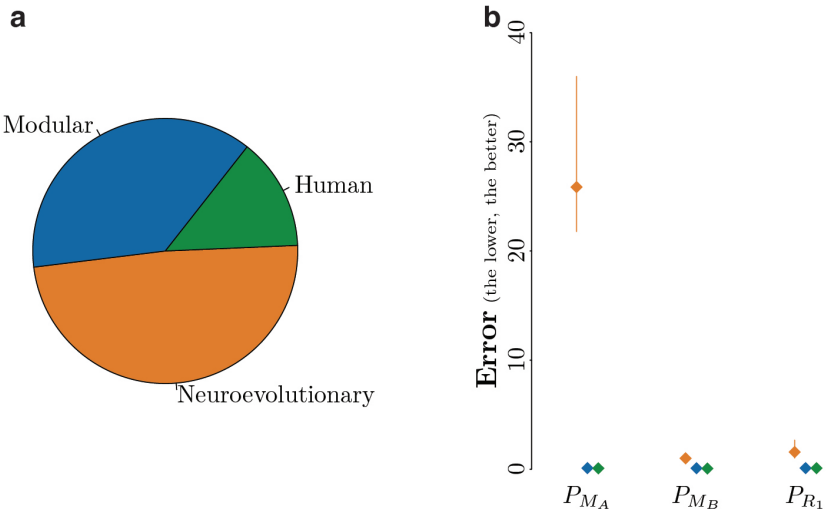
We report in Figure 6.2c the mean *regret* of all predictors. Results show that P_{M_A} obtains the largest *regret*, with a value of 0.26; P_{R_1} has the second largest one with a value of 0.24, and P_{M_B} obtains the smallest one with a value of 0.22. The improvement of P_{M_B} and P_{R_1} over P_{M_A} is significant. It should be noted that this improvement of the pseudo-reality predictors over P_{M_A} (and the one regarding the *best* reported in the previous subsection) is marginal with respect to the improvement we observed regarding the *error* (Figure 6.2a). This difference of magnitude throughout the evaluation criteria shows the importance of analyzing the estimations of relative performance of pairs of instances of control software together with the accuracy of the performance estimations of instances individually: a predictor might poorly estimate performance of two instances, yet it might estimate the relative performance correctly, leading to the correct prediction of the best performing control software.

6.4 Analysis based on the origin of the control software

Here we divide the data contained in DS 1 with respect to the approach used to generate the instances of control software. We consider three families: the neuroevolutionary one regroups instances in the form of neural networks that have been generated by neuroevolutionary design methods (Nolfi and Floreano, 2000), the modular one regroups instances in the form of probabilistic finite-state machines or behavior trees that have been produced by modular design methods (Francesca and Birattari, 2016; Kuckling et al., 2018), and the human one regroups instances produced by human designers. We analyze the accuracy of the predictors within and across these three families.

Of the 1021 instances of control software considered, 49% have been produced by 10 methods belonging to the neuroevolutionary family, and 38% by 6 methods belonging to the modular family; the remaining instances have been created by human designers (Fig 6.3a). Throughout the original studies from which we collected these instances of control software, methods belonging to the neuroevolutionary approach have shown to suffer a relatively large performance drop: they produced control software that performed well in simulation (that is, on the design model M_A), but poorly in reality. On the other hand, the modular methods and human designers produced control software that performed satisfactorily in both simulation and reality. The analysis of the *error* yield by the predictors on the three families of methods confirms these observations (Figure 6.3). In fact, the median *error* of P_{M_A} is considerably higher for the control software produced by neuroevolutionary methods (almost 26) with respect to those produced by the modular methods (0.1) and human designers (0.08). A similar trend can be observed for pseudo-reality predictors P_{M_B} and P_{R_1} (Figure 6.3b): the median *error* is substantially larger for performance predictions of neuroevolutionary instances (1 and 1.58, respectively) than for the one of modular instances (0.09 and 0.1) or for the one of the human instances (0.06 and 0.09). The pseudo-reality predictors are noticeably more accurate at predicting performance of the neuroevolutionary instances of control software than P_{M_A}, and P_{M_B} is more accurate than P_{R_1}. The improvement of P_{M_B} over P_{M_A} and P_{R_1} is significant with a confidence level of at least 95%. For what concerns the control software produced by modular methods and human designers, the difference in median *error* between P_{M_A} and the pseudo-reality predictors is negligible.

Among the 43 520 possible comparisons of instances of control software available in DS 1, 62% are homogeneous comparisons—that is, comparisons of pairs of instances produced by design methods belonging to the same family—and 48% are heterogeneous comparisons—that is, the two instances have been produced by design methods belonging to different families. Because two instances belonging to the same family display similar ranges of performance in reality, predicting which of the two will perform better is difficult. On the other hand, it is less challenging for two instances conceived by design methods belonging to different families as their real-world performance differ noticeably. Figure 6.4 reports the *best* and *regret* of the predictors for comparisons of

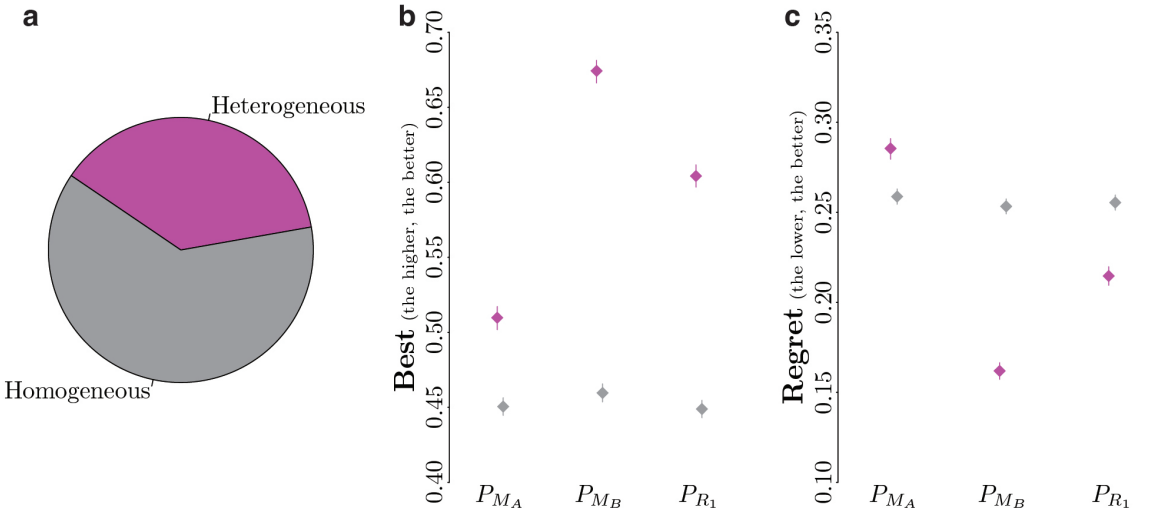


Source: Ligot A, Birattari M (2022b)

Figure 6.3: ***Error* of the predictors for different families of design methods**: neuroevolutionary methods, modular methods, and human designers. (**a**) The group 'neuroevolutionary' refers to instances of control software produced by neuroevolutionary methods, the group 'modular' refers to those produced by modular methods, and the group 'human' refers to those produced by human designers. (**b**) median *error*. For each predictor, the orange left-most points represent the median *error* when considering control software produced by neuroevolutionary robotics methods; the blue central points represent the one when considering control software produced by modular methods; the green right-most points represent the one when considering those produced by human designers. The vertical segments represent the 95% confidence interval.

homogeneous and heterogeneous pairs of design methods. Surprisingly, the mean *best* of the pseudo-reality predictors P_{M_B} and P_{R_1} is only slightly higher (better) than the one of P_{M_A} when considering homogeneous comparisons (Figure 6.4b). In fact, P_{M_A} obtains a mean *best* of 0.451, whereas P_{M_B} and P_{R_1} obtain one of 0.46 and 0.449, respectively. The three predictors thus fail to correctly predict which instance of control software performs better in reality for more than 50% of the comparisons. Also for the case of the mean *regret*, values are extremely close: P_{M_A} obtained a mean *regret* of 0.259, whereas P_{M_B} and P_{R_1} both obtained one of 0.253 and 0.256, respectively (Figure 6.4c).

Although all three predictors have higher mean *best* for heterogeneous pairs of instances of control software than for homogeneous ones, the difference is minor for what concerns P_{M_A} in comparison with the ones of P_{M_B} and P_{R_1} (Figure 6.4b). Whereas these last two predictors are able to correctly predict which instance is the best performing one in respectively 67.3% and 60.4% of the heterogeneous comparisons of DS 1,



Source: Ligot A, Birattari M (2022b)

Figure 6.4: ***Best* and *regret* of the predictors for different comparisons of design methods**. (**a**) The group 'homogeneous' refers to comparisons of control software generated by design methods belonging to the same family: both methods belong to either the neuroevolutionary approach, the modular one, or have been designed by a human. The group 'heterogeneous' refers to comparisons of control software generated by methods of different families. (**b**) Mean *best*; (**c**) mean *regret*. In both plots, and for each predictor, the fuchsia left-most points represent the mean best or mean regret when considering heterogeneous comparisons; the gray right-most points represent the ones when considering homogeneous comparisons. The vertical segments represent the 95% confidence interval on the respective metric.

P_{M_A} can only do so for 51%. As a result of the poor accuracy of P_{M_A} in predicting the best performing instance of control software for heterogeneous pairs, its mean *regret* is considerably larger with respect to the one for homogeneous pairs. In fact, P_{M_A} is the only predictor that has a larger mean *regret* for heterogeneous pairs than for homogeneous ones: the one of P_{M_B} drops to 0.16, whereas the one of P_{R_1} decreases to 0.21 (Figure 6.4c).

An analysis of the *best* and *regret* of the predictor P_{M_A} confirms the assumption that predicting the best performing instance of control software out of a heterogeneous pair is more straightforward as the two instances are more likely to display different ranges of performance in reality. In fact, although the *best* of P_{M_A} is higher (better) for heterogeneous comparisons than for homogeneous ones, its mean *regret* for heterogeneous pairs is also larger (worse) than for homogeneous ones. This is explained by the fact that the difference in real-world performance between instances of control software for a heterogeneous pair is important, leading to a larger relative *regret*—that is, a larger loss—when the wrong instance is predicted to be the best performing

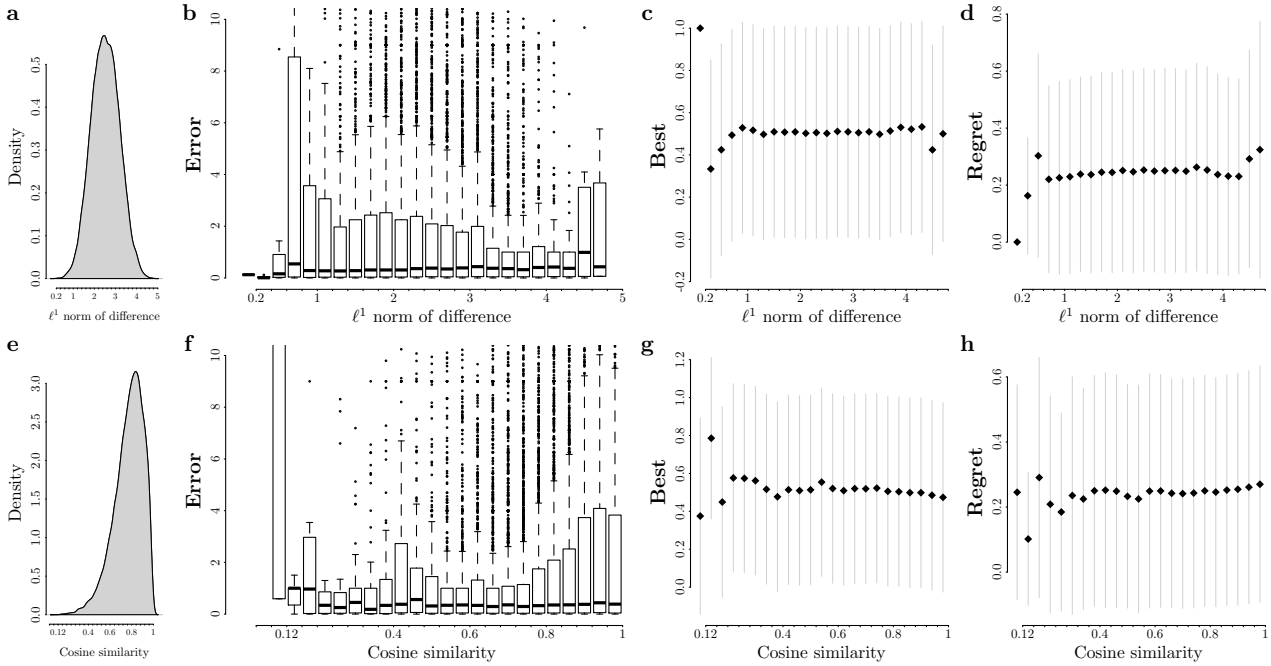
one in reality. On the contrary, mistakes in detecting the best performing instance in homogeneous pairs might be more frequent, but they result in a smaller loss as the performance in reality are more likely to be similar.

6.5 Correlation between predictions and width of the pseudo-reality gap

The notion of width of a pseudo-reality gap introduced in Chapter 5 refers to a measure of the difference between the design model in which control software is conceived and the pseudo-reality model in which it is evaluated. In this section, we evaluate each instance of control software on 30 models sampled from the range R, and we study the correlation between the three evaluation criteria and the width of the pseudo-reality gap these models create with the design model. We compute (a) the ℓ^1 norm of differences between two models and (b) their cosine similarity to quantify the extent to which the pseudo-reality models sampled from R differ from the design model M_A. These two measures are described in the Section 5.2. With this analysis, our goal is to learn whether the measures considered provide meaningful information, such as the possible existence of a subrange of R that leads to more accurate predictions. If it is the case, we envision that these measures (or similar ones) would be helpful for the future definition of new, better predictors.

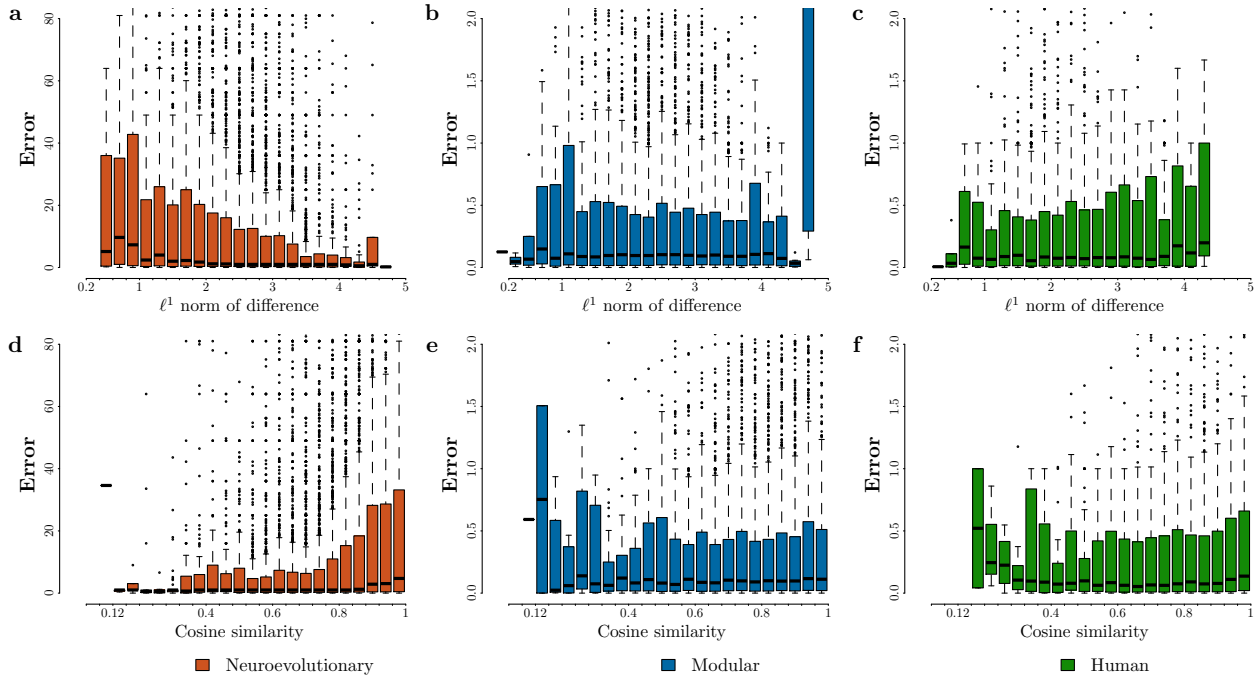
In Figure 6.5, we report the distribution of the widths computed with the two aforementioned measures. We also discretize the widths and report the *error*, *best*, and *regret* computed across all the available instances of control software. In particular, we represent the *error* with box-and-whiskers boxplots, and plot the mean *best* and mean *regret* together with the associated standard deviations. In Figures 6.6 and 6.7, we report the correlation between the widths and the *error*, *best*, and *regret* according the different groups considered in the previous section—that is, we consider the *error* of control software produced by neuroevolutionary, modular, and human design methods; and the *best* and *regret* resulting from homogeneous and heterogeneous comparisons of control software.

When computed as the ℓ^1 norms of the differences between the models R_1 and M_A, the width ranges from 0 to 5, with the larger the norm, the wider the gap between the two models (Figure 6.5a–d). The distribution of the widths of the pseudo-reality gap is symmetric around the mean 2.5 (Figure 6.5a). The *error* slightly decreases as the width increases, but increases for widths greater than 4.4 (Figure 6.5b). This suggests that a larger difference between evaluation and design model yields better accuracy, but that a too large difference might be counterproductive. In Figure 6.6, it can be observed that the *error* for control software produced by neuroevolutionary methods decreases when width increases, whereas the one for control software produced by modular methods remain stable. At visual inspection, the *error* for control software produced by human designers seems to increase as the width increases, but the Pearson



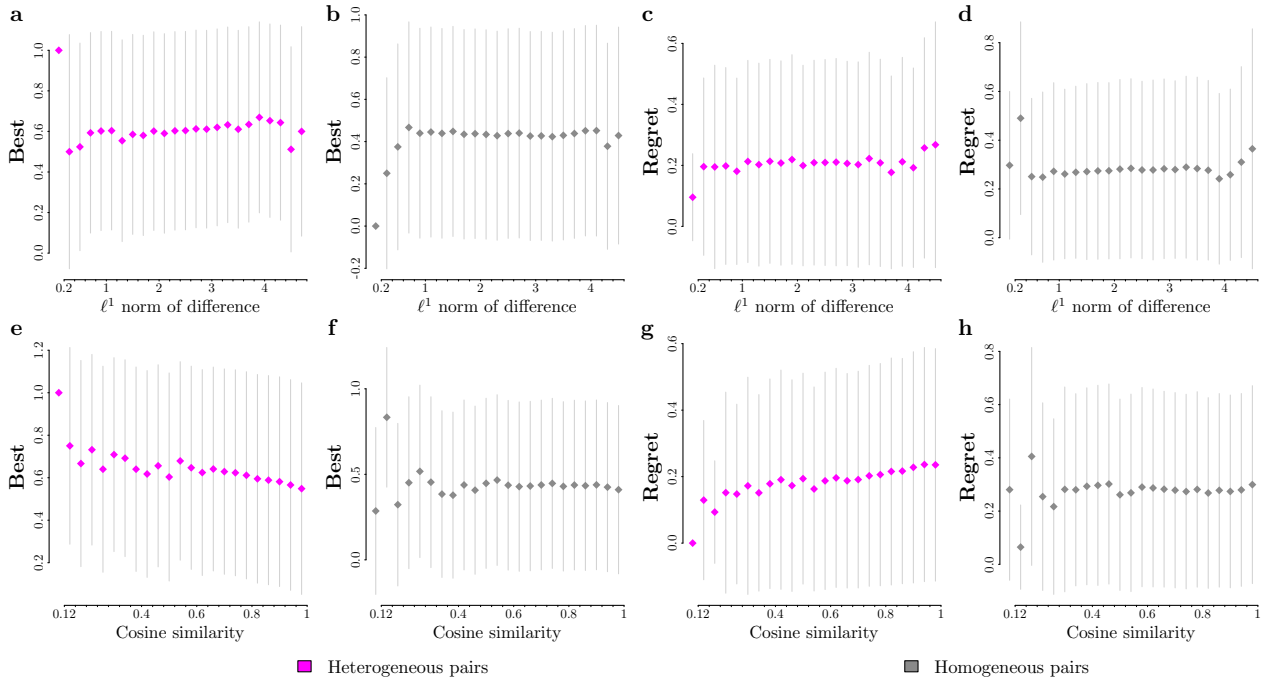
Source: Ligot A, Birattari M (2022b)

Figure 6.5: **Width of pseudo-reality gap: *error*, *best*, and *regret*.** For each instance of control software included in DS 1, and any possible pairwise combinations of these instances, 30 models were randomly sampled from the range R. We computed the width of the pseudo-reality gap created between the sampled models and the design model M_A using two measures: the ℓ^1 norm of differences, and the cosine similarity. (**a**) Distribution of the width measured as the ℓ^1 norm of the differences between the R_1 models and M_A, with mean equal to 2.5. (**b**), (**c**), and (**d**) *error*, *best*, and *regret* with respect to the ℓ^1 of the differences between the sampled models and the design one. Their Pearson correlation coefficients are equal to -0.01, 0.002, 0.01, respectively. (**e**) Distribution of the width computed as the cosine similarity between the R_1 models and M_A, with mean equal to 0.77. (**f**), (**g**), and (**h**) *error*, *best*, and *regret* with respect to the cosine similarity between sampled models and the design one. Their Pearson correlation coefficients are equal to 0.03, -0.02, 0.02, respectively. It is worth noting that the ℓ^1 norm of differences between M_A and M_B is equal to 1.59, whereas their cosine similarity is equal to 0.89.



Source: Ligot A, Birattari M (2022b)

Figure 6.6: **Width of the pseudo-reality gap**: *error* of control software produced by neuroevolutionary methods (**a**, **d**), modular methods (**b**, **e**), and human designers (**c**, **f**). Widths are computed with the ℓ^1 norm of differences (**a**, **b**, **c**) and the cosine similarity (**d**, **e**, **f**). Pearson correlation coefficients are equal to (**a**) -0.023 (**b**) -0.006 (**c**) -0.038 (**d**) 0.043 (**e**) 0.002 (**f**) 0.015.



Source: Ligot A, Birattari M (2022b)

Figure 6.7: **Width of the pseudo-reality gap**: *best* (**a**, **b**, **e**, **f**) and *regret* (**c**, **d**, **g**, **h**) when considering heterogeneous (**a**,**c**, **e**, **g**) and homogeneous (**b**, **f**, **d**, **h**) pairs of instances of control software. Widths are computed with the ℓ^1 norm of differences (**a**, **b**, **c**, **d**) and the cosine similarity (**e**, **f**, **g**, **h**). Pearson correlation coefficients are equal to (**a**) 0.03 (**b**) -0.007 (**c**) 0.001 (**d**) 0.008 (**e**) -0.054 (**f**) -0.003 (**g**) 0.052 (**h**) -0.004.

correlation coefficient of -0.04 indicates a negative correlation. When considering all instances of control software, the *best* first increases, then quickly plateaus for widths larger than 1 (Figure 6.5c). The same can be observed when only considering homogeneous comparisons of control software, whereas the *best* tends to keep increasing until widths larger that 4.4 (Figure 6.7a-b). For what concerns the *regret*, it remains relatively stable for all widths.

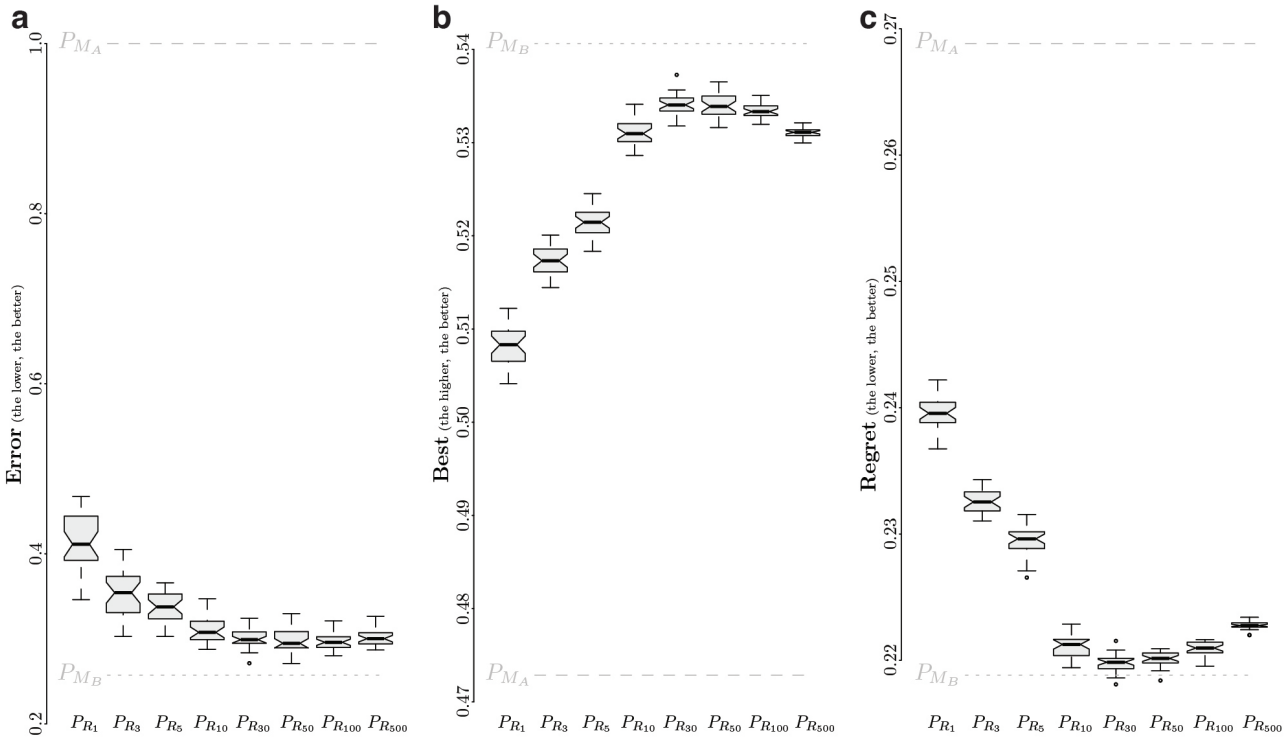
In a positive space such as the one considered here, the cosine similarity ranges from 0 to 1, with the lower the value, the wider the gap between the two models (Figure 6.5e–h). When models from R are sampled uniformly like we did in this study, the distribution of the width is skewed to the left (Figure 6.5e). Figure 6.5f shows that the *error* slightly increases as the design model and the evaluation ones get closer—in other words, when the gap reduces. Figure 6.5g-f show the *best* decreasing and the *regret* increasing as the gap gets smaller, respectively. Figures 6.6d-f show that, as design and evaluation models get closer, the *error* of the instances of control software produced by neuroevolutionary methods increases more decidedly than the one of those produced by modular methods or designed by humans. Figures 6.7e-h show that, for heterogeneous pairs of control software, the *best* decreases and the *regret* increases as the gaps narrow down, whereas it remains relatively constant when considering homogeneous pairs.

Overall, trends are more visible when one observes the cosine similarity than the ℓ^1 norms of the differences. The Pearson correlation coefficients reported in the captions of Figure 6.5 support this, with values slightly greater when using the cosine similarity, whereas those reported in Figures 6.6 and 6.7 exacerbate the differences between the different groups studied.

6.6 Varying the sample size of R

The predictor P_{R_k} consists, for a given instance of control software, in its evaluation on k models sampled from the range R of models. The resulting prediction of the performance of the instance is the median of the estimated performance resulting from the k evaluations. We consider $k \in \{1, 3, 5, 10, 30, 50, 100, 500\}$ to study the effect of the sample size on the accuracy of the predictor. We apply P_{R_k} on DS 1 30 times for each k considered, and present the resulting median *error*, mean *best*, and mean *regret* of the 30 executions in the form of box-and-whiskers plots (Figure 6.8).

In Figure 6.8a, each box represents 30 median *error*, each resulting from the execution of a predictor on DS 1. In addition to the decrease of the variance with the increase of the number of models sampled from R, the results show that the accuracy increases (i.e., the *error* decreases). The best score is obtained by $P_{R_{50}}$, with a median value of 0.295, which corresponds to an improvement of 28% over P_{R_1}. One can notice a slight increase of the *error* as the sample size exceeds 50 models (that is, for predictors $P_{R_{100}}$ and $P_{R_{500}}$). In Figure 6.8b, each box represents 30 mean *best*. The plot shows a clear improvement of the accuracy until the sample size reaches 30 models, the accuracy



Source: Ligot A, Birattari M (2022b)

Figure 6.8: **Effect of the sampling size on the predictor $\mathbf{P_{R_k}}$.** (**a**) median *error*, (**b**) mean *best*, and (**c**) mean *regret*. The results are presented using notched box-and-whiskers plots, where the notches represent the 95% confidence interval on the median. If notches on different boxes do not overlap, the medians of the corresponding predictors differ significantly with a confidence of at least 95%. Each box represents the metrics resulting from 30 executions of the predictors. Performance of P_{M_A} (dotted line) and P_{M_B} (dashed line) are added for comparison.

then plateaus. Surprisingly, the accuracy is significantly lower when 500 models are sampled (that is, $P_{R_{500}}$); the accuracy is then equivalent to estimating performance with 10 models, yet with lower variance. Both $P_{R_{30}}$ and $P_{R_{50}}$ obtain a score of 0.534, which is an improvement of about 5% with respect to the one of P_{R_1}. Figure 6.8b, in which each box represents 30 mean *regret*, shows similar trends, only inverted. In fact, one can see an improvement of the accuracy (i.e., a decrease of the *regret*) as the number of sampled models n increases. The *regret* is the lowest for $P_{R_{30}}$, with a value of 0.22 which corresponds to an improvement of 8% over P_{R_1}, then increases for larger sample size. In fact, $P_{R_{500}}$ is significantly worst than $P_{R_{10}}$.

Overall, considering multiple models sampled from R to estimate the performance of an instance of control software leads to better accuracy. However, our results shows that there is an optimal value in the number of models, and that larger sample size does not necessarily means higher accuracy. Yet, even if very large sample sizes are suboptimal, they are still significantly more accurate than very small ones or than P_{M_A}.

6.7 Discussion

The results reported in this chapter show that, on DS 1, the pseudo-reality predictors proposed are significantly more accurate than the design model to estimate the real-world performance of control software, and to forecast eventual rank inversions between pairs of instances of control software. The results confirm that the concept of pseudo-reality can be used as part of a methodology to predict the robustness of control software and of optimization-based design methods.

To the best of our knowledge, this study is the first that focuses on the problem of predicting real-world performance of robot control software that is generated off-line on the basis of simulation. The evaluations of predictors must be conducted on a substantial amount of instances of control software, ideally created by a variety of design methods and to address a variety of missions, in order to make meaningful and serious claims about their accuracy. Such study would therefore require instantiating a large number of design processes, and evaluating all the instances of control software produced with real robots, which would be extremely expensive and time consuming. To avoid this burden, we reused control software and its real-world performance that we collected from several previously published studies in optimization-based design of robot swarms—which is, to the best of our knowledge, a first as well.

A possible shortcoming of our study is the fact that the data we reused to evaluate the predictors were produced entirely by researchers of our laboratory. This because, unfortunately, due to the aforementioned issues in running real-robot experiments in swarm robotics, large experimental campaigns are particularly rare: few laboratories and research groups have the resources to afford them, and many studies still only rely on the simulation model used during the design to evaluate automatically generated control software. Moreover, publicly sharing results, control software, and source code

in a user-friendly and reusable manner is not a common practice in our community. We hope that our study will influence other researchers to share their results and convince them that experimental data might have a use that goes beyond the specific study in which they were produced.

Chapter 7

Conclusion

This thesis contributes to the progress of optimization-based design of robot swarms. Optimization-based design has the potential to become a general methodology for the conception of robot swarms as it bypasses the complex endeavor of deriving the rules that dictate the appropriate robot-robot and robot-environment interactions from the desired collective behavior (Dorigo et al., 2021). To meet expectations, many fundamental issues have to be overcome; this thesis addressed a few of them.

The literature on optimization-based design lacks a consistently applied empirical practice and, as a result, a well-established state of the art (Francesca and Birattari, 2016). Recent discussions have disclosed the fact that two approaches have so far been entangled in the literature: semi-automatic and fully-automatic design (Birattari et al., 2019, 2020). In semi-automatic design, a design method is a tool that a human operator uses to realize the swarm behavior that they have in mind. To produce the desired behavior—or one that is as close as possible—the operator fine-tunes aspects of the design process such as the control software architecture, the parameters of the optimization algorithm, or the performance measure to be optimized. This fine-tuning is generally a three-step process that is iterated at will: the execution of the design method, the evaluation (possibly on physical robots) of control software produced, and the modification of some elements of the design process. The semi-automatic approach is labor intensive, and it is thus best suited for complex, one-of-a-kind missions for which it is reasonable to expect that sufficient time and resources are available to properly calibrate the design method. In fully-automatic design, the design method cannot undergo any per-mission manual adaptation. It is thus best suited for solving missions consecutively sampled from a given stream or class of missions, in such a way that it would be unfeasible for a human to supervise and modify the design processes associated to each mission.

The classification between semi-automatic and fully-automatic design, alongside the traditional one that distinguishes between on-line and off-line design, is a first step towards establishing a clear state of the art. In fact, it contributes to highlighting the research questions relevant to each approach, to setting appropriate expectations on what each should achieve, and to defining the challenges to be faced by each of them.

Research questions, expectations, and challenges that are specific to semi-automatic design revolve around the nature and complexity of the mission to be solved, the level of expertise required by the human operator, and the amount of time needed to be devoted by the human operator or the number of iterations of the design process ought to be executed. Research questions, expectations, and challenges that are specific to fully-automatic design revolve around the nature and complexity of the class of missions to be addressed, the robustness of the solution produced to the differences between design and deployment conditions, and the computational time needed to solve each mission of the class considered. The fundamental differences between the two approaches naturally imply the necessity to adopt different experimental protocols for empirical studies in each domain. Our analysis of the literature showed a disparity in the assessment of design methods within the two approaches. More importantly, it revealed shortcomings.

Experimental protocols employed in semi-automatic studies typically consider one mission, instantiate multiple design processes for that mission, select a subset of the instances of control software produced by the design processes, and execute the instances they selected on physical robots for further evaluation and assessment. Often, the subset of control software produced that is then ported to physical robots is composed of only one instance: the one that performs best in simulation. Although discrepancies can be found within the experimental protocols employed in semi-automatic design (in terms of number of design processes executed, number of instances of control software ported on physical robots for evaluation, or number of evaluations of each of these instances), they generally reflect the tenets of semi-automatic design: they are not intended to estimate the expected performance of a design method for a given mission, but rather to demonstrate that the method can be used as an adjustable tool to produce the collective behavior needed to solve the mission at hand.

Experimental protocols employed in fully-automatic studies typically consider several missions of different types (that is, that are characterized by different performance measures), instantiate multiple design processes for each missions, and execute all the generated instances of control software once on physical robots. It has been shown that these experimental protocols are to be adopted when one wants to estimate the performance of a design method on a given mission (Birattari, 2020). However, these experimental protocols fail to encompass one of the cornerstone of fully-automatic design: the notion of class of missions. A class of missions is a set composed of missions of different types and of missions of the same type but that differ from one another by manor variations (two missions of the same type might be configured differently if, for example, the number and/or positions of obstacles and/or points of interest within these missions vary), together with a probability measure that determines the relative frequency of appearance of each of them. By evaluating and comparing the performance of design methods on a specific configuration of a mission rather than on a whole class like it is currently done in the literature, one might (unwillingly) introduce bias towards one of the methods due to specificities of the configuration chosen.

The first main contribution of this thesis is the definition of an experimental protocol that aims at evaluating the performance of design methods over a whole class of missions, and that is therefore to be adopted when one evaluates design methods in the fully-automatic context. This protocol is characterized by two complementary elements: the notion of mission generator to define benchmarks for the evaluation of design methods, and a sampling strategy that minimizes the variance when estimating their expected performance. A mission generator is a tool that samples missions belonging to a given class of mission—that is, a tool that selects missions of the class according to the associated probability measure, and independently of the design method to which it applies. The sampling strategy recommends to assess the expected performance of a design method on the maximal number of missions possible, to run one design process per mission, and to execute the resulting instance of control software once on the physical robots. We provided an intuitive explanation and a formal proof that this sampling strategy, under the assumption that a limited number of executions of control software on physical robots can be performed, is the one that ought to be adopted to minimize the variance of the expected performance of a design method. The assumption on the maximal number of executions of control software on the physical robots is realistic because robot experiments are expensive and time consuming, and as such represent the real bottleneck of the research in optimization-based design. The notion of mission generator and the sampling strategy are fundamentally complementary as the sampling strategy recommends to evaluate a design method on the maximal number of missions possible, and the notion of mission generator allows one to sample as many missions as desired.

To illustrate the experimental protocol, we presented an experiment in which we evaluated and compared the performance of two previously proposed design methods. For this purpose, we created the first generator of missions for swarm robotics: MG 1. We created MG 1 as an open-source library for the ARGoS3 simulator (Pinciroli et al., 2012). Although MG 1 is relatively limited in the sense that it can only generate missions of three different types to be solved by robots with specific capabilities, MG 1 can easily be extended and generalized as it defines missions in terms of environmental features and relationships between these features. In addition, the library we created does not only implement MG 1, but a whole class of generators. In fact, the modification of parameters of MG 1, such as the frequency of appearance of environmental features, entails the creation of a different mission generator that would sample a different class of missions.

MG 1 considers the number of robots in the swarm as a parameter of the mission itself. Rather than imposing the size of the swarm, we expect future generators to define missions that impose a constraint on the maximal/minimal number of robots and to allow the design process to determine the most appropriate number of robots. More generally, one could conceive generators whose purpose is to evaluate the performance of concurrent design methods—that is, methods that select and configure the sensors and actuators of the individual robots in addition to designing their control software.

This could be done by specifying a set of possible hardware modules that the design process must select and combine to conceive the robots of the swarm. We also expect such generators, in prospective practical applications of fully-automatic design, to integrate elements of real-world design problems such as economic constraints for more realistic evaluations of performance of design methods. Taking inspiration from the work of Salman et al. (2019), this could be done by specifying a cost to each hardware module available, and to include constraints on the total monetary budget available with each mission of the class.

The main difficulty when estimating the performance of a design method on different missions is that the range of performance might vary greatly across the missions at hand. Normalization prior to the aggregation of the performance is needed. In our illustrative experiment, we aggregated the performance observed by reporting the expected rank, which is an implicit form of normalization. However, using ranks does not provide an estimation of the overall performance of each of the design methods under analysis. We also normalized the performance obtained by the generated control software with the one of a baseline behavior: random walk. Although promising, this normalization comports drawbacks in the form of potential divisions by zero if the performance of the baseline behavior is null, and the necessity of executing the baseline behavior on physical robots. As an alternative, one could normalize the performance on each mission based on the knowledge of the theoretical maximal and minimal performance, or based on a reasonable estimate of them, including, for example, the best and worse performance observed empirically (Hasselmann et al., 2021). However, in our illustrative experiment, these alternatives were not appropriate as no prior knowledge was available and only two methods were involved in the study, providing therefore too little data to perform a meaningful normalization. The aggregation of performance thus remains an open issue.

The second main contribution of this thesis consists in shedding further light on one of the most important problem to be faced in off-line optimization-based design: the reality gap. It is commonly believed that effects of the reality gap (that is, performance drop and rank inversion) are due to the fact that reality is more complex than the simulations on the basis of which control software is designed—or equivalently, that simulations are too simplistic. We brought empirical evidence that following this complexity assumption leads to a contradiction. In fact, we showed that effects of the reality gap can occur even if we can exclude that the simulation model under which control software is designed is a simplistic version of the context under which the control software is assessed. Rather, we showed that the effects of the reality gap should be ascribed to a sort of *overfitting* of the conditions experienced in the design phase, regardless of the fact that these conditions are more or less complex than those faced in the evaluation phase.

The third main contribution of the thesis is the definition and empirical assessment of predictors of real-world performance. The predictors we considered are based on the concept of pseudo-reality: a simulation model, different from the one used during the

design, that is used for the evaluation of control software and therefore plays the role of reality. In particular, we considered the predictor P_{M_B} that consists in the execution of control software on the pseudo-reality model M_B used to disprove the aforementioned complexity assumption; the predictor P_{R_1} that consists in the execution of an instance of control software on a single pseudo-reality model sampled from a range R of possible models; and its generalized version P_{R_k} that consists in the execution of an instance of control software on k models sampled from R. We investigated the ability of these predictors to forecast real-world performance of automatically generated control software, and compared it to the classical approach adopted in the literature to estimate real-world performance, which relies on the evaluation of control software on the simulation model used in the design process.

To make meaningful and serious claims about the accuracy of predictors, their assessments need to be performed on a large amount of control software, ideally conceived by various design methods and to tackle various missions. This would require running an extremely large amount of experiments with real robots, which would be expensive and time consuming. We avoided the burden of running this large amount of experiments by reusing control software (and its real-world performance) that we collected from seven previously published studies in swarm robotics—which is, to the best of our knowledge, a first in swarm robotics. We compared the predicted performance with the one observed on the physical robots, and we assessed the accuracy of the different predictors with three evaluation criteria: one related to the accuracy of the predictions of the expected real-world performance, the other two related to the occurrence of rank inversions between pairs of instances of control software. The results we obtained are promising: they show that a more accurate alternative to the widely adopted practice for predicting real-world performance exists.

Future work

A number of issues remain to be addressed. Firstly, the experimental protocol we proposed does not apply to semi-automatic design even though the lack of empirical practice is arguably more severe than for the fully-automatic approach: almost no comparisons of design methods are performed. Secondly, the thesis does not provide solutions to the reality-gap problem.

Elaborating adequate protocols for the assessment and comparison of multiple semi-automatic methods is particularly challenging due to the role played by the human expert during the design process as well as the specific nature of the missions to be accomplished. We foresee that an adequate protocol could be inspired by the one used by Francesca et al. (2015) to compare optimization-based and manual design methods. In that study, five experts were each asked to i) define a mission and ii) solve two missions created by their peers using manual methods. Rather than conceiving the control software with manual methods, one could ask participants to manipulate design methods so as to produce solutions that they deem appropriate. One could,

for example, limit the amount of time given to the experimenters or the maximal amount of visual evaluations of the produced control software to assess the ease of use of the design methods considered. Involving several humans in the definition of the benchmark and in the steering of design processes seems to be necessary to avoid bias in the estimation of the expected performance of design methods used in the semi-automatic context, but we leave this for future work. Nonetheless, as semi-automatic and fully-automatic design are closely tied to one another, we foresee that progress in fully-automatic design will somehow contribute to the progress of semi-automatic design, and that the protocol we proposed will therefore also positively impact off-line and on-line semi-automatic design, even if indirectly.

The pseudo-reality predictors we proposed did not yield perfect estimations of real-world performance, and there is therefore room for improvement. In this thesis, we considered pseudo-reality models that differ from the design model by the amount of noise applied to the sensors and actuators of the robots. We foresee that one could define predictors with higher accuracy by following parallel avenues. One might consider a wider range of possible pseudo-realities by including offsets to the noise applied to the sensors and actuators, and multiple distributions. One might also go beyond noise and consider different parameters, or consider different structures of the simulation model. Our study on the correlation between accuracy and differences between the design model and the sampled ones suggests that an optimal subrange of models (or a single model) that leads to higher performance exists. Rather than searching for an optimal subrange or a unique model by hand, one could define an automatic procedure instead. We foresee such a procedure to use an optimization algorithm, one or several evaluation criteria to guide the search, and to be based on the decomposition of the available control software and corresponding real-world performance into training and evaluation sets as it is typically done in machine learning (Hastie et al., 2009; Goodfellow et al., 2016). In this work, we used the same pseudo-reality model for all the robots. One might consider an heterogeneous approach and simulate each robot with a distinct model. Finally, one might consider different physics engines, or even different simulators.

It is reasonable to assume that simulation predictors, however accurate they might become, will never replace experiments with physical robots. Yet, we foresee that they could considerably reduce the amount of tests with physical robots, and would therefore facilitate the research in off-line design of robot swarms. In particular, in the case of fully-automatic design, reliable predictors could contribute to handle the so-called *overdesign*: it has been shown that, past an optimal number of steps of the design process, the performance observed in reality diverges from the one in simulation (Birattari et al., 2016). As a consequence, protracting a design process indefinitely could be counterproductive. A reliable simulation-only predictor of real-world performance could be used to implement an early stopping mechanism that halts the design process when overdesign has occurred (Morgan and Bourlard, 1990; Caruana et al., 2001).

In addition, we foresee that the concept of pseudo-reality could be leveraged to

produce control software that is more robust to the reality gap—a problem that tends to be overlooked by our community (Hasselmann et al., 2021). The results of this thesis substantiate the contention that the problem of the reality gap is reminiscent of the generalization problem faced in machine learning; we hence believe that the literature of that domain, and in particular the techniques employed to enhance the generalization capabilities of machine learning models, should be a source of inspiration to optimization-based design of robot swarms. It is well known that the field of machine learning, and especially deep learning (Zhu et al., 2016), requires a lot of data to obtain satisfactory results (Goodfellow et al., 2016). A wide variety of data is indeed needed during the training to enable machine learning models to later generalize to unseen data. In this regard, the practice in optimization-based design of robot swarms is quite different. In fact, the candidate instances of control software are typically evaluated in a single context of execution during the design—that is, the simulation model—before being ported on the physical robots.

The most promising solution to mitigate effects of the reality gap is the *transferability approach* proposed by Koos et al. (2013). This approach uses periodic evaluations of instances of control software on physical robots to constrain the design process to consider only those that do not overfit simulation, and therefore to discard those that are not robust. In other words, similarly to what is done in machine learning, the transferability approach relies on the estimation of the robustness of control software on several execution contexts—two to be precise. Relying on robot experiments is time consuming and expensive, and more importantly, limits the adoption of this approach to the semi-automatic design of robot swarms. In machine learning, researchers often use *data augmentation* to counter the high cost related to the collection and the labeling of sufficient data, a technique that consists in adding synthetic data to the training set (see Mumuni and Mumuni (2022) for a recent review of the recent data augmentation techniques). In their paper, Koos et al. (2013) compared their transferability approach to a technique that is similar to data augmentation as it consists in applying random variation in the simulation—a technique sometimes called *domain randomization* (Peng et al., 2018; Andrychowicz et al., 2020). This technique, in the experiment conducted by Koos et al. (2013), did not yield satisfactory results. Yet, we believe that this approach deserves more attention. The authors only considered random variations related to the simulation environment—what we call arena in this thesis—not to the simulation models. We foresee that evaluating candidate instances of control software on multiple simulation models (i.e., multiple pseudo-realities) rather than on a single one could be beneficial as it would give a better estimation of their intrinsic robustness. Using task-agnostic predictors of real-world performance like the pseudo-reality ones we proposed instead of the robot experiments employed in the transferability approach would be particularly valuable as it would completely automatize the design process, and substantially reduce its cost and length. We would recommend this avenue of research to be explored in a systematic and incremental way: we expect that understanding which elements of simulations (and which combinations of these elements)

will be crucial to the design of instances of control software that are robust to the reality gap.

Appendix A

Minimizing variance: a mathematical proof

This appendix provides a complete proof of the theorem presented in Chapter 3.

A sampling strategy for estimating the expected performance of a design method on a class of missions, given that a maximum number N of executions can be performed, can be formally described by a triple $\langle n_m, n_d, n_x \rangle$, with $\widetilde{N} := n_m \cdot n_d \cdot n_x \leq N$. The expected performance is estimated on the basis of n_m missions, n_d design processes per mission (to generate n_d instances of control software per mission), and n_x executions of each of them. We can assume that: (1) n_m missions m_i (with $i = 1 \dots n_m$) can be sampled independently from a same (fixed) distribution $P_M(\cdot)$, where $P_M(m)$ is the probability of having to solve mission m; (2) given a mission $m_{i'}$, n_d instances of control software $d_{i'j}$ (with $j = 1 \dots n_d$) can be generated for that mission, which can be formally described as sampling independently n_d instances of control software from a same (fixed) condition distribution $P_D(\cdot|m_{i'})$, where $P_D(d|m)$ is the conditional probability of producing design d, having to solve mission m; (3) given mission $m_{i'}$ and design $d_{i'j'}$, n_x execution can be performed of design $d_{i'j'}$ on mission $m_{i'}$ so as to observe n_x scores $s_{i'j'k}$ (with $k = 1 \dots n_x$), which can be formally described as being sampled independently from a same (fixed) conditional distribution $P_S(\cdot|d_{i'j'}, m_{i'})$, where $P_S(s|d, m)$ is the conditional probability of observing score s when running design d on mission m. Further, we can assume that the cost (in abstract terms: time and resources) of running a design process is negligible compared to the one of running robot experiments. We can also assume that sampling a mission from a class of instances is inexpensive and that a sample of arbitrary size can be obtained. We also assume that, before running a design process on a given mission, we do not have any prior information on how well the control software we can generate automatically will perform and on what will be the variance of the performance. It has to be noticed that any triple $\langle n_m, n_d, n_x \rangle$ yields an unbiased estimate of the expected performance. Yet, different triples might differ for what concerns the variance of the estimate they yield.

Theorem 2. *Under the assumptions made above, given that a maximum number N of executions can be performed, the sampling strategy described by the triple $\mathcal{E} = \langle n_m, n_d, n_x \rangle$, with $n_m = N$, $n_d = 1$, and $n_x = 1$, is the one that minimizes the variance of the estimate.*

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

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

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

The remainder of this appendix is dedicated to deriving Equation A.1.

A.1 Definitions

A sampling strategy $\mathcal{E} = \langle n_m, n_d, n_x \rangle$ is a triplet of integers where n_m denotes the number of missions, n_d the number of designs per mission (resulting in n_d instances of control software), and n_x the number of executions on the robots of each instance of control software. The total number of executions on the robots is denoted by $N = n_m \cdot n_d \cdot n_x$.

The joint probability of having to solve mission m, producing the design d, and eventually observing the score s is:

$$P(s, d, m) = P_S(s|d, m)P_D(d|m)P_M(m), \tag{A.2}$$

where $P_M(m)$ is the probability of having to solve mission m; $P_D(d|m)$ is the conditional probability of producing design d, having to solve mission m; and $P_S(s|d, m)$ is the conditional probability of observing score s, while performing design d on mission m. The expected value of s with respect to this joint probability is:

$$\mu := \int s \, \mathrm{d}P_M(m) \, \mathrm{d}P_D(d|m) \, \mathrm{d}P_S(s|d, m). \tag{A.3}$$

The expected value of the score within mission m and within design d for mission m are respectively:

$$\mu_m := \int s \, \mathrm{d}P_D(d|m) \, \mathrm{d}P_S(s|d, m) \qquad \text{and} \qquad \mu_{md} := \int s \, \mathrm{d}P_S(s|d, m). \tag{A.4}$$

The variance within mission m is:

$$\sigma_m^2 := \int (s - \mu_m)^2 \, \mathrm{d}P_D(d|m) \, \mathrm{d}P_S(s|d, m). \tag{A.5}$$

The **expected within-mission variance** provides information on how scores differ one from the other within a same mission (averaged across all possible missions); it is defined as:

$$\bar{\sigma}_{WM}^2 := \int \sigma_m^2 \, \mathrm{d}P_M(m) = \int (s - \mu_m)^2 \, \mathrm{d}P_M(m) \, \mathrm{d}P_D(d|m) \, \mathrm{d}P_S(s|d, m). \tag{A.6}$$

The across-design variance within mission m is :

$$\bar{\sigma}_{AD,m}^2 := \int (\mu_{md} - \mu_m)^2 \, \mathrm{d}P_D(d|m). \tag{A.7}$$

The **expected across-design variance** provides information on how designs differ one from the other within a same mission (averaged across all possible missions); it is defined as:

$$\bar{\sigma}_{AD}^2 := \int \bar{\sigma}_{AD,m}^2 \, \mathrm{d}P_M(m) = \int (\mu_{m,d} - \mu_m)^2 \, \mathrm{d}P_M(m) \, \mathrm{d}P_D(d|m). \tag{A.8}$$

The **across-mission variance** provides information on how missions differ one from the other; it is defined as:

$$\sigma_{AM}^2 := \int (\mu_m - \mu)^2 \, \mathrm{d}P_M(m). \tag{A.9}$$

In the following, with the notation: $\int f(v_1, v_2, \ldots, v_L) \bigodot_{l=1}^{L} \mathrm{d}P(v_l)$, we denote the sequence of nested integrals $\int\int \cdots \int f(v_1, v_2, \ldots, v_L) \, \mathrm{d}P(v_1) \, \mathrm{d}P(v_2) \ldots \mathrm{d}P(v_L)$.

A.2 Proof

The goal of this proof is to show that given the following estimator

$$\hat{\mu}_{\mathcal{E}} = \frac{1}{\widetilde{N}} \sum_{i=1}^{n_m} \sum_{j=1}^{n_d} \sum_{k=1}^{n_x} s_{ijk},$$

where s_{ijk} is the score (or performance) observed in the execution x_{ijk} of the instance of control software issued from the design d_{ij} on the mission m_i, the variance of $\hat{\mu}_{\mathcal{E}}$ is:

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

$$\mathbb{E}[(\hat{\mu}_{\mathcal{E}}-\mu)^2]=\int(\hat{\mu}_{\mathcal{E}}-\mu)^2\,\mathrm{d}P(\hat{\mu}_{\mathcal{E}})=$$

$$\int\left(\frac{1}{\widetilde{N}}\sum_{i=1}^{n_m}\sum_{j=1}^{n_d}\sum_{k=1}^{n_x}s_{ijk}-\mu\right)^2\bigodot_{i=1}^{n_m}\mathrm{d}P_M(m_i)\bigodot_{j=1}^{n_d}\mathrm{d}P_D(d_{ij}|m_i)\bigodot_{k=1}^{n_x}\mathrm{d}P_S(s_{ijk}|d_{ij},m_i)=$$

$$\int\left(\frac{1}{\widetilde{N}}\sum_{i=1}^{n_m}\sum_{j=1}^{n_d}\sum_{k=1}^{n_x}(s_{ijk}-\mu)\right)^2\bigodot_{i=1}^{n_m}\mathrm{d}P_M(m_i)\bigodot_{j=1}^{n_d}\mathrm{d}P_D(d_{ij}|m_i)\bigodot_{k=1}^{n_x}\mathrm{d}P_S(s_{ijk}|d_{ij},m_i)=$$

$$\frac{1}{\widetilde{N}^2}\int\left(\sum_{i=1}^{n_m}\sum_{j=1}^{n_d}\sum_{k=1}^{n_x}\big(\underbrace{s_{ijk}-\mu_{m_i}}_{a}+\underbrace{\mu_{m_i}-\mu}_{b}\big)\right)^2\bigodot_{i=1}^{n_m}\mathrm{d}P_M(m_i)\bigodot_{j=1}^{n_d}\mathrm{d}P_D(d_{ij}|m_i)\bigodot_{k=1}^{n_x}\mathrm{d}P_S(s_{ijk}|d_{ij},m_i)=$$

$$\frac{1}{\widetilde{N}^2}\int\sum_{i=1}^{n_m}\sum_{j=1}^{n_d}\sum_{k=1}^{n_x}\sum_{i'=1}^{n_m}\sum_{j'=1}^{n_d}\sum_{k'=1}^{n_x}(s_{ijk}-\mu_{m_i}+\mu_{m_i}-\mu)\left(s_{i'j'k'}-\mu_{m_{i'}}+\mu_{m_{i'}}-\mu\right)\bigodot_{i=1}^{n_m}\mathrm{d}P_M(m_i)$$

$$\bigodot_{j=1}^{n_d}\mathrm{d}P_D(d_{ij}|m_i)\bigodot_{k=1}^{n_x}\mathrm{d}P_S(s_{ijk}|d_{ij},m_i)\bigodot_{i'=1}^{n_m}\mathrm{d}P_M(m_{i'})\bigodot_{j'=1}^{n_d}\mathrm{d}P_D(d_{i'j'}|m_{i'})\bigodot_{k'=1}^{n_x}\mathrm{d}P_S(s_{i'j'k'}|d_{i'j'},m_{i'})=$$

$$\frac{1}{\widetilde{N}^2}\sum_{i=1}^{n_m}\sum_{j=1}^{n_d}\sum_{k=1}^{n_x}\sum_{i'=1}^{n_m}\sum_{j'=1}^{n_d}\sum_{k'=1}^{n_x}\int(s_{ijk}-\mu_{m_i}+\mu_{m_i}-\mu)\left(s_{i'j'k'}-\mu_{m_{i'}}+\mu_{m_{i'}}-\mu\right)\bigodot_{i=1}^{n_m}\mathrm{d}P_M(m_i)$$

$$\bigodot_{j=1}^{n_d}\mathrm{d}P_D(d_{ij}|m_i)\bigodot_{k=1}^{n_x}\mathrm{d}P_S(s_{ijk}|d_{ij},m_i)\bigodot_{i'=1}^{n_m}\mathrm{d}P_M(m_{i'})\bigodot_{j'=1}^{n_d}\mathrm{d}P_D(d_{i'j'}|m_{i'})\bigodot_{k'=1}^{n_x}\mathrm{d}P_S(s_{i'j'k'}|d_{i'j'},m_{i'})$$

Using $(a+b)^2 = a^2 + b^2 + 2ab$ and the linearity of the integral we can break the last integral into three terms which will be analyzed separately:

$$\frac{1}{\widetilde{N}^2}\sum_{i=1}^{n_m}\sum_{j=1}^{n_d}\sum_{k=1}^{n_x}\sum_{i'=1}^{n_m}\sum_{j'=1}^{n_d}\sum_{k'=1}^{n_x}\int\Bigg(\underbrace{(s_{ijk}-\mu_{m_i})(s_{i'j'k'}-\mu_{m_{i'}})}_{a^2}+\underbrace{(\mu_{m_i}-\mu)(\mu_{m_{i'}}-\mu)}_{b^2}+$$
$$+\underbrace{(s_{ijk}-\mu_{m_i})(\mu_{m_{i'}}-\mu)+(s_{i'j'k'}-\mu_{m_{i'}})(\mu_{m_i}-\mu)}_{ab+ab}\Bigg)$$
$$\bigodot_{i=1}^{n_m}\mathrm{d}P_M(m_i)\bigodot_{j=1}^{n_d}\mathrm{d}P_D(d_{ij}|m_i)\bigodot_{k=1}^{n_x}\mathrm{d}P_S(s_{ijk}|d_{ij},m_i)$$
$$\bigodot_{i'=1}^{n_m}\mathrm{d}P_M(m_{i'})\bigodot_{j'=1}^{n_d}\mathrm{d}P_D(d_{i'j'}|m_{i'})\bigodot_{k'=1}^{n_x}\mathrm{d}P_S(s_{i'j'k'}|d_{i'j'},m_{i'}) =$$
$$\frac{1}{\widetilde{N}^2}\sum_{i=1}^{n_m}\sum_{j=1}^{n_d}\sum_{k=1}^{n_x}\sum_{i'=1}^{n_m}\sum_{j'=1}^{n_d}\sum_{k'=1}^{n_x}\int\Bigg(\underbrace{(s_{ijk}-\mu_{m_i})(s_{i'j'k'}-\mu_{m_{i'}})}_{a^2}+\underbrace{(\mu_{m_i}-\mu)(\mu_{m_{i'}}-\mu)}_{b^2}+$$
$$+\underbrace{2(s_{ijk}-\mu_{m_i})(\mu_{m_{i'}}-\mu)}_{2ab}\Bigg)$$
$$\bigodot_{i=1}^{n_m}\mathrm{d}P_M(m_i)\bigodot_{j=1}^{n_d}\mathrm{d}P_D(d_{ij}|m_i)\bigodot_{k=1}^{n_x}\mathrm{d}P_S(s_{ijk}|d_{ij},m_i)$$
$$\bigodot_{i'=1}^{n_m}\mathrm{d}P_M(m_{i'})\bigodot_{j'=1}^{n_d}\mathrm{d}P_D(d_{i'j'}|m_{i'})\bigodot_{k'=1}^{n_x}\mathrm{d}P_S(s_{i'j'k'}|d_{i'j'},m_{i'})$$

Summand I: a^2

$$\frac{1}{\widetilde{N}^2}\sum_{i=1}^{n_m}\sum_{j=1}^{n_d}\sum_{k=1}^{n_x}\sum_{i'=1}^{n_m}\sum_{j'=1}^{n_d}\sum_{k'=1}^{n_x}\int(s_{ijk}-\mu_{m_i})(s_{i'j'k'}-\mu_{m_{i'}})\bigodot_{i=1}^{n_m}\mathrm{d}P_M(m_i)\bigodot_{j=1}^{n_d}\mathrm{d}P_D(d_{ij}|m_i)$$
$$\bigodot_{k=1}^{n_x}\mathrm{d}P_S(s_{ijk}|d_{ij},m_i)\bigodot_{i'=1}^{n_m}\mathrm{d}P_M(m_{i'})\bigodot_{j'=1}^{n_d}\mathrm{d}P_D(d_{i'j'}|m_{i'})\bigodot_{k'=1}^{n_x}\mathrm{d}P_S(s_{i'j'k'}|d_{i'j'},m_{i'}).$$

At this point we will analyze Summand I for different indices.

$i \neq i'$

Since $(s_{ijk} - \mu_{m_i})$ and $(s_{i'j'k'} - \mu_{m_{i'}})$ depend on different variables, any addend of Summand I with $i \neq i'$ can be rewritten as

$$\frac{1}{\widetilde{N}^2} \int (s_{ijk} - \mu_{m_i})(s_{i'j'k'} - \mu_{m_{i'}}) \bigodot_{i=1}^{n_m} \mathrm{d}P_M(m_i) \bigodot_{j=1}^{n_d} \mathrm{d}P_D(d_{ij}|m_i) \bigodot_{k=1}^{n_x} \mathrm{d}P_S(s_{ijk}|d_{ij}, m_i)$$
$$\bigodot_{i'=1}^{n_m} \mathrm{d}P_M(m_{i'}) \bigodot_{j'=1}^{n_d} \mathrm{d}P_D(d_{ij'}|m_i) \bigodot_{k'=1}^{n_x} \mathrm{d}P_S(s_{ijk'}|d_{ij}, m_i) =$$
$$\frac{1}{\widetilde{N}^2} \int (s_{ijk} - \mu_{m_i}) \bigodot_{i=1}^{n_m} \mathrm{d}P_M(m_i) \bigodot_{j=1}^{n_d} \mathrm{d}P_D(d_{ij}|m_i) \bigodot_{k=1}^{n_x} \mathrm{d}P_S(s_{ijk}|d_{ij}, m_i)$$
$$\cdot \int (s_{i'j'k'} - \mu_{m_{i'}}) \bigodot_{i'=1}^{n_m} \mathrm{d}P_M(m_{i'}) \bigodot_{j'=1}^{n_d} \mathrm{d}P_D(d_{i'j'}|m_{i'}) \bigodot_{k'=1}^{n_x} \mathrm{d}P_S(s_{i'j'k'}|d_{i'j'}, m_{i'}) = 0 \text{ (by definition)}$$

$i = i'$, $j \neq j'$

$$\frac{1}{\widetilde{N}^2} \int (s_{ijk} - \mu_{m_i})(s_{ij'k'} - \mu_{m_{i'}}) \bigodot_{i=1}^{n_m} \mathrm{d}P_M(m_i) \bigodot_{j=1}^{n_d} \mathrm{d}P_D(d_{ij}|m_i) \bigodot_{k=1}^{n_x} \mathrm{d}P_S(s_{ijk}|d_{ij}, m_i)$$
$$\bigodot_{i'=1}^{n_m} \mathrm{d}P_M(m_{i'}) \bigodot_{j'=1}^{n_d} \mathrm{d}P_D(d_{ij'}|m_i) \bigodot_{k'=1}^{n_x} \mathrm{d}P_S(s_{ijk'}|d_{ij}, m_i) =$$
$$= \frac{1}{\widetilde{N}^2} \int \left[\int (s_{ijk} - \mu_{m_i}) \bigodot_{j=1}^{n_d} \mathrm{d}P_D(d_{ij}|m_i) \bigodot_{k=1}^{n_x} \mathrm{d}P_S(s_{ijk}|d_{ij}, m_i) \cdot \right.$$
$$\left. \int (s_{ij'k'} - \mu_{m_{i'}}) \bigodot_{j'=1}^{n_d} \mathrm{d}P_D(d_{ij'}|m_i) \bigodot_{k'=1}^{n_x} \mathrm{d}P_S(s_{ij'k'}|d_{ij'}, m_i) \right] \bigodot_{i=1}^{n_m} \mathrm{d}P_M(m_i) = 0 \text{ (by definition)}$$

$i = i'$, $j = j'$, $k \neq k'$

$$\frac{1}{\widetilde{N}^2} \int (s_{ijk} - \mu_{m_i})(s_{ijk'} - \mu_{m_{i'}}) \bigodot_{i=1}^{n_m} \mathrm{d}P_M(m_i) \bigodot_{j=1}^{n_d} \mathrm{d}P_D(d_{ij}|m_i) \bigodot_{k=1}^{n_x} \mathrm{d}P_S(s_{ijk}|d_{ij}, m_i)$$
$$\bigodot_{i'=1}^{n_m} \mathrm{d}P_M(m_{i'}) \bigodot_{j'=1}^{n_d} \mathrm{d}P_D(d_{ij'}|m_i) \bigodot_{k'=1}^{n_x} \mathrm{d}P_S(s_{ijk'}|d_{ij}, m_i) =$$

$$= \frac{1}{\widetilde{N}^2} \int \left[\int (s_{ijk} - \mu_{m_i}) \bigodot_{k=1}^{n_x} \mathrm{d}P_S(s_{ijk}|d_{ij}, m_i) \cdot \int (s_{ijk'} - \mu_{m_i}) \bigodot_{k'=1}^{n_x} \mathrm{d}P_S(s_{ijk'}|d_{ij}, m_i) \right]$$

$$\bigodot_{j=1}^{n_d} \mathrm{d}P_D(d_{ij}|m_i) \bigodot_{i=1}^{n_m} \mathrm{d}P_M(m_i) = \frac{1}{\widetilde{N}^2} \int (\mu_{m_i,d_j} - \mu_{m_i})^2 \bigodot_{i=1}^{n_m} \mathrm{d}P_M(m_i) \bigodot_{j=1}^{n_d} \mathrm{d}P_D(d_{ij}|m_i) = \frac{\bar{\sigma}^2_{AD}}{\widetilde{N}^2},$$

and therefore

$$\sum_{i=1}^{n_m} \sum_{j=1}^{n_d} \sum_{k=1}^{n_x} \sum_{k'=1}^{n_x} \frac{\bar{\sigma}^2_{AD}}{\widetilde{N}^2} = \frac{n_m\, n_d\, n_x^2}{\widetilde{N}^2} \bar{\sigma}^2_{AD} = \frac{\bar{\sigma}^2_{AD}}{n_m\, n_d}.$$

$i = i'$, $j = j'$, $k = k'$

$$\frac{1}{\widetilde{N}^2} \int (s_{ijk} - \mu_{m_i})(s_{ijk} - \mu_{m_i}) \bigodot_{i=1}^{n_m} \mathrm{d}P_M(m_i) \bigodot_{j=1}^{n_d} \mathrm{d}P_D(d_{ij}|m_i) \bigodot_{k=1}^{n_x} \mathrm{d}P_S(s_{ijk}|d_{ij}, m_i) =$$

$$= \frac{1}{\widetilde{N}^2} \int (s_{ijk} - \mu_{m_i})^2 \bigodot_{i=1}^{n_m} \mathrm{d}P_M(m_i) \bigodot_{j=1}^{n_d} \mathrm{d}P_D(d_{ij}|m_i) \bigodot_{k=1}^{n_x} \mathrm{d}P_S(s_{ijk}|d_{ij}, m_i) = \frac{\bar{\sigma}^2_{WM}}{\widetilde{N}^2},$$

and thus

$$\sum_{i=1}^{n_m} \sum_{j=1}^{n_d} \sum_{k=1}^{n_x} \frac{\bar{\sigma}^2_{WM}}{\widetilde{N}^2} = \frac{n_m\, n_d\, n_x}{\widetilde{N}^2} \bar{\sigma}^2_{WM} = \frac{\bar{\sigma}^2_{WM}}{n_m\, n_d\, n_x}.$$

Gathering everything, Summand I amounts to

$$a^2 = \frac{\bar{\sigma}^2_{AD}}{n_m\, n_d} + \frac{\bar{\sigma}^2_{WM}}{n_m\, n_d\, n_x}.$$

Summand II: b^2

$$\frac{1}{\widetilde{N}^2} \sum_{i=1}^{n_m} \sum_{j=1}^{n_d} \sum_{k=1}^{n_x} \sum_{i'=1}^{n_m} \sum_{j'=1}^{n_d} \sum_{k'=1}^{n_x} \int (\mu_{m_i} - \mu)(\mu_{m_{i'}} - \mu) \bigodot_{i=1}^{n_m} \mathrm{d}P_M(m_i) \bigodot_{j=1}^{n_d} \mathrm{d}P_D(d_{ij}|m_i) \bigodot_{k=1}^{n_x} \mathrm{d}P_S(s_{ijk}|d_{ij}, m_i)$$
$$\bigodot_{i'=1}^{n_m} \mathrm{d}P_M(m_{i'}) \bigodot_{j'=1}^{n_d} \mathrm{d}P_D(d_{i'j'}|m_{i'}) \bigodot_{k'=1}^{n_x} \mathrm{d}P_S(s_{i'j'k'}|d_{i'j'}, m_{i'}) =$$
$$\frac{n_d^2\, n_x^2}{\widetilde{N}^2} \sum_{i=1}^{n_m} \sum_{i'=1}^{n_m} \int (\mu_{m_i} - \mu)(\mu_{m_{i'}} - \mu) \bigodot_{i=1}^{n_m} \mathrm{d}P_M(m_i) \bigodot_{i'=1}^{n_m} \mathrm{d}P_M(m_{i'})$$

$i \neq i'$

$$\int (\mu_{m_i} - \mu) \bigodot_{i=1}^{n_m} \mathrm{d}P_M(m_i) \cdot \int (\mu_{m_{i'}} - \mu) \bigodot_{i'=1}^{n_m} \mathrm{d}P_M(m_{i'}) = 0 \text{ (by definition)}$$

$i = i'$

$$\int (\mu_{m_i} - \mu)^2 \bigodot_{i=1}^{n_m} \mathrm{d}P_M(m_i) = \sigma^2_{AM},$$

and therefore

$$\frac{n_d^2\, n_x^2}{\widetilde{N}^2} \sum_{i=1}^{n_m} \sigma^2_{AM} = \frac{\sigma^2_{AM}}{n_m}$$

Gathering everything Summand II adds to

$$b^2 = \frac{\sigma^2_{AM}}{n_m}.$$

Summand III: $2ab$

$$\frac{2}{\widetilde{N}^2} \sum_{i=1}^{n_m} \sum_{j=1}^{n_d} \sum_{k=1}^{n_x} \sum_{i'=1}^{n_m} \sum_{j'=1}^{n_d} \sum_{k'=1}^{n_x} \int (s_{ijk} - \mu_{m_i})(\mu_{m_{i'}} - \mu) \bigodot_{i=1}^{n_m} \mathrm{d}P_M(m_i) \bigodot_{j=1}^{n_d} \mathrm{d}P_D(d_{ij}|m_i) \bigodot_{k=1}^{n_x} \mathrm{d}P_S(s_{ijk}|d_{ij}, m_i)$$
$$\bigodot_{i'=1}^{n_m} \mathrm{d}P_M(m_{i'}) =$$

$$= \int \left[(\mu_{m_{i'}} - \mu) \int (s_{ijk} - \mu_{m_i}) \bigodot_{j=1}^{n_d} \mathrm{d}P_D(d_{ij}|m_i) \bigodot_{k=1}^{n_x} \mathrm{d}P_S(s_{ijk}|d_{ij}, m_i) \right] \bigodot_{i=1}^{n_m} \mathrm{d}P_M(m_i) \bigodot_{i'=1}^{n_m} \mathrm{d}P_M(m_{i'}) = 0$$

(by definition the inmost integral is null)

Gathering all the nonzero terms yields:

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

Appendix B

Dataset DS 1

This appendix provides a description of the content of the dataset DS 1 presented in Chapter 6.

DS 1 contains the real-world performance of 1021 instances of control software generated by 18 different off-line design methods for 45 missions. The majority of these instances have been evaluated once on physical robots, and a few have been evaluated multiple times under different initial configurations of the swarm—that is, positions and orientations of the robots. In total, DS 1 contains 1385 observations of real-world performance. For each real-world evaluation, DS 1 also contains the predictions yield by P_{M_A}, P_{M_B}, and of 1380 models uniformly sampled from the range R of possible pseudo-reality models—see subsection Predictors for more details. The predictions were obtained by executing the 1021 available instances of control software on the different simulation models and with the same initial configurations of the swarm that were used during the evaluations on the physical robots. In the ARGoS3 simulator (Pinciroli et al., 2012), an initial configuration is configured via a seed fed to the random number generator—the seeds used in the executions of the control software are also part of DS 1.

B.1 Robotic platform

The e-puck is a small two wheeled robot commonly used in swarm robotics (Mondada et al., 2009). All the control software collected in this study has been generated to be executed on the same version of the e-puck enhanced with additional hardware (Garattoni et al., 2015): the Overo Gumstix, the ground sensor module, and the range-and-bearing module (Gutiérrez et al., 2009). This version can detect obstacles and measure the ambient light, perceive the gray-level color of the floor situated under its body, and detect the number of neighboring peers situated in an approximate range of 0.70 m as well as estimate their relative position.

The capabilities of the robot are formally described in a reference model which serves as an interface for the control software: it describes what variables associated to

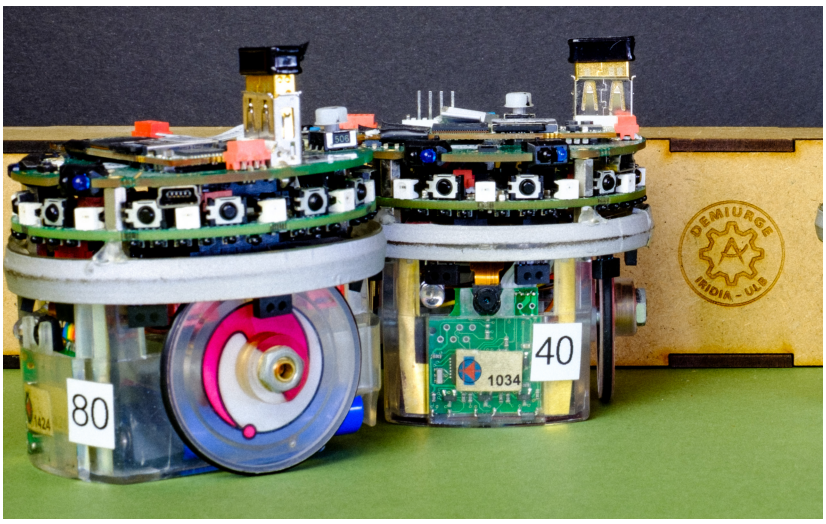


Figure B.1: Picture of two e-puck robots in the configuration used in the experiments presented in this thesis. The e-puck robot is cylindrical in shape, with a height of about 50 mm and a diameter of about 70 mm.

the capabilities of the robots are accessible to the control software. The vast majorities of the design methods used to generate the control software collected have access to the reference model RM1.1. Two design methods, namely `Gianduja` and `EvoCom`, have access to reference model RM2, which enables the robots to send and react to a one bit message broadcasted via the range-and-bearing module. The two reference models are depicted in Table B.1.

B.2 Design methods

We briefly describe the main characteristics of the design methods that produced the control software whose real-performance and estimated ones compose DS 1. We divide these methods according to three families of approaches: the neuroevolutionary one, the modular one, and the human one. A summary is given in Table B.2. We refer the reader to the original papers for further details on these design methods.

B.2.1 Neuroevolutionary methods

Neuroevolutionary robotics is the most popular optimization-based approach for designing control software for robot swarms. In this approach, a neural network controls the individual robots: sensor readings are fed to the neural network as inputs, and the network's output dictates the robot actuator values. The configuration of the neural network is optimized by an evolutionary algorithm.

With the exception of `EvoCom`, all implementations of the neuroevolutionary approach described here generate neural networks defined on the basis of reference model

Table B.1: Reference model RM1.1 and RM2. Rows in gray are specific to RM2. The range-and-bearing vector $V = \sum_{m=1}^{n}(\frac{1}{1+r_m}, \angle b_m)$, where r_m and $\angle b_m$ are range and bearing of neighbor m, respectively, points to the aggregate position of the neighboring peers. If $n = 0$, then $V = (1, \angle 0)$. The vector V_b is computed as V by restricting to the b broadcasting neighboring robots. The variables are updated every 100 ms.

sensor	variables
proximity	$prox_i \in [0, 1]$, with $i \in \{0, 1, ..., 7\}$
light	$light_i \in [0, 1]$, with $i \in \{0, 1, ..., 7\}$
ground	$ground_i \in \{white, gray, black\}$, with $i \in \{0, 1, 2\}$
range-and-bearing	$n \in \{0, 1, ..., 19\}$
	$V \in ([0.5, 20], [0, 2\pi]\,\text{rad})$
	$b \in \{0, 1, ..., 19\}$
	$V_b \in ([0.5, 20], [0, 2\pi]\,\text{rad})$

actuator	variables
wheels	$v_l, v_r \in [-0.12, 0.12]\text{ms}^{-1}$
broadcast	$s \in \{on, off\}$

RM1.1. The neural networks produced have 25 input nodes, 2 output nodes, and the synaptic weights range in $[-5, 5]$. The 25 input nodes are organized as follows: 8 are dedicated to the readings of the proximity sensors, 8 to those of the light sensors, 3 to those of the ground sensors, 4 to the projections of the range-and-bearing vector V on four unit vectors that point to 45°, 135°, 225°, and 315°, 1 to the number of peers perceived, and 1 is a bias. The 2 output nodes define the velocity of the wheels.

The instances of control software considered in the dataset DS 1 have been generated by the following neuroevolutionary methods: **`EvoStick`** is a simple implementation of the neuroevolutionary robotics approached introduced by Francesca et al. (2012). It generates fully-connected, feed-forward neural networks that do not comprise hidden layers. `EvoStick` uses an evolutionary algorithm that has a population size of 100 individuals, evaluates each individual 10 times per generation, and produces novel populations based on elitism and mutation: the 20 best individuals are passed unchanged to the following generation, and the remaining 80 individuals are obtained via mutations applied to the same 20 best individuals. **`CMA-ES-slp`** is based on the evolutionary algorithm `CMA-ES` (Hansen and Ostermeier, 2001). In `CMA-ES`, the population is described in statistical terms via the covariance matrix of its distribution. `CMA-ES-slp` adopts the same network topology as `EvoStick`. **`CMA-ES-mlp`** differs from `CMA-ES-slp` in the topology of the neural networks produced: the ones

of CMA-ES-mlp contain one hidden layer composed of 14 nodes, including 1 bias node. **xNES-slp** is based on the evolutionary algorithm xNES (Glasmachers et al., 2010). xNES is identical to CMA-ES with the exception that its update rule is defined in a principled way. xNES-slp adopts the same network topology as EvoStick. **xNES-mlp** differs from xNES-slp only in the network topology adopted: it generates neural networks that contain a hidden layer of 14 nodes. **NEAT-A-slp** is based on the evolutionary algorithm NEAT (Stanley and Miikkulainen, 2002). With NEAT, both the synaptic weights and the topology of the neural networks are optimized. The initial population is composed of fully-connected, feed-forward neural networks that do not comport hidden layers. **NEAT-A-nl** differs from NEAT-A-slp only in the topology of the neural networks that compose the initial population. Here, the input nodes are initially not connected to the output nodes. **NEAT-B-slp** differs from NEAT-A-slp only in the value of some hyper-parameters of NEAT. Here, NEAT is configured so that it has a higher compatibility coefficient, does not penalize old species, and can generate recurrent neural networks. **NEAT-B-nl** differs from NEAT-B-slp only in the topology of the neural networks that compose the initial population: the input nodes are initially disconnected from the output nodes. **EvoCom** is derived from EvoStick and differs from all the previous methods in the input and output nodes that comport the neural networks it produces. Indeed, EvoCom is defined on the basis of RM2 which adds communication capabilities with respect to RM1.1. EvoCom generates neural networks that have 5 additional input nodes and 1 additional output node with respect to those of EvoStick. The 5 additional input nodes are dedicated to the detection of peers that are broadcasting a message: 1 is dedicated to the number of broadcasting peers perceived, and 4 to the projections of the range-and-bearing vector V_b on four unit vectors that point to 45°, 135°, 225°, and 315°.

B.2.2 Modular methods

The modular methods that produced control software belonging to the dataset DS 1 all belong to the AutoMoDe approach (Francesca et al., 2014b). All these design methods generate control software by selecting and combining pre-defined modules: low-level behaviors that are executed by the robots, and conditions that are used to transition from one low-level behavior to another. With the exception of those of Gianduja, these modules are defined on the basis of reference model RM1.1.

The instances of control software considered in the dataset DS 1 have been generated by the following AutoMoDe methods: **Vanilla** is the first implementation of AutoMoDe (Francesca et al., 2014b). Vanilla generates control software in the form of probabilistic finite-state machines that can comprise up to four states and up to four outgoing edges per state. Vanilla has its disposal a set of 12 software modules to conceive the probabilistic finite-state machines: 6 are low-level behaviors that are used as states, and 6 are conditions that are used as edges to transition from one behavior to another. All these software modules have been conceived by hand once-and-for-all

in a mission-agnostic way by a human designer; some of them have parameters that are tuned during the design by the optimization algorithm to adjust their functioning. `Vanilla` uses the F-race optimization algorithm to select, tune, and combine the modules (Birattari, 2009; Birattari et al., 2002). **`Chocolate`** is the second implementation of AutoMoDe, and only differs from `Vanilla` in the optimization algorithm adopted: `Chocolate` uses Iterated F-race (Iterated F-race) (Balaprakash et al., 2007; Birattari et al., 2010; López-Ibáñez et al., 2016), an improved version of the F-race algorithm adopted by `Vanilla`. `Chocolate` has been introduced in Francesca et al. (2015) and later used many studies (Ligot et al., 2020b; Hasselmann and Birattari, 2020; Ligot et al., 2020a; Spaey et al., 2020; Hasselmann et al., 2021; Ligot et al., 2022). **`Maple`** differs from `Chocolate` in the architecture of the control software produced: `Maple` selects, tunes, and combines the modules into behaviors trees (Kuckling et al., 2018; Champandard, 2007; Champandard et al., 2010). **`Arlequin`** differs from `Chocolate` in the nature of the 6 pre-defined low-level behaviors that are at its disposal for conceiving control software: rather than combining manufactured behaviors, `Arlequin` combines behaviors that are automatically generated via the neuroevolutionary method `EvoStick` (Ligot et al., 2020a). **`Coconut`** differs from `Chocolate` only in the number of pre-defined low-level behaviors that it can select, tune, and combine: `Coconut` embeds 2 additional exploration schemes within its modules (Spaey et al., 2020). **`Gianduja`** is derived from `Chocolate` differs from all the previous methods in the robot capabilities it exploits: `Gianduja`'s modules are defined on the basis of reference model RM2, which extends RM1.1 with communication capabilities (Hasselmann et al., 2018b; Hasselmann and Birattari, 2020). `Gianduja` operates on 8 low-level behaviors: 6 are the same as `Chocolate` extended with a binary parameter deciding whether a one bit message is broadcast while the behavior is performed, the 2 others make the robot go towards broadcasting peers, or in the opposite direction. `Gianduja` also operates on 8 conditions: 6 are the same as `Chocolate`, the 2 others are related to the number of broadcasting peers perceived.

B.2.3 Manual methods

In Francesca et al. (2015), the authors compared the performance of the control software generated by `Vanilla`, `EvoStick`, and `Chocolate` with the one conceived by 5 human experts in swarm robotics. Each expert had to solve two different missions, once following two different guidelines (that is, using two different manual design method). In both cases, the experts had to conceive control software on the basis of RM1.1. The two manual methods that have produced instances of control software present in the DS 1 are the following: **`U-Human`**, short for unconstrained-human, consists in letting the human designer implement the control software in the way they deem most appropriate, without any kind of restriction. **`C-Human`** short for constrained-human, consists in constraining the human designer to use the same software modules used by `Vanilla` and `Chocolate`. They are indeed constrained to combine the software

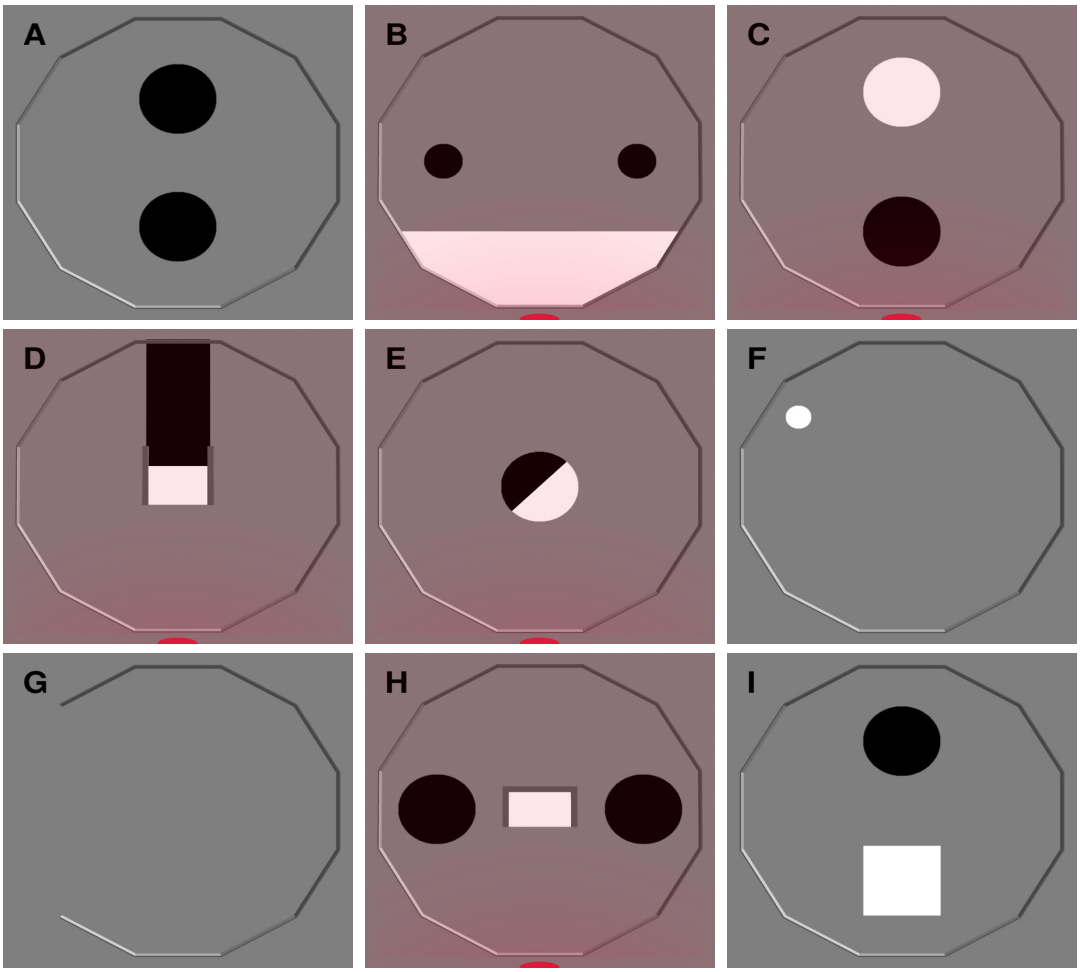
modules into probabilistic finite-states machines that comport the same restrictions as those produced by those methods: up to four states with up to four outgoing edges.

In Hasselmann et al. (2021), the authors compared the performance of control software generated by 9 neuroevolutionary methods to the performance of a simple strategy consisting in the robots randomly roaming in the environment. This strategy was called **`RandomWalk`** in their paper, and we consider it as an instance of control software produced by manual method.

B.3 Missions

We briefly describe here the missions to be solved by the control software collected. Some missions have been used in multiple studies, and might differ slightly in some aspects. We refer the reader to the original papers for further information. Some missions have been studied with different types of objective functions which can be classified as *endtime*, if the performance is computed once at the very end of the experiment, and as *anytime*, if the performance is computed continuously throughout the experiment. Fig. B.2 illustrates some of the arenas corresponding to the missions described bellow.

In **AggregationXOR**, the swarm must select one of the two black areas and aggregate there (Fig. B.2a). The endtime objective function, to be maximized, is $E_{\text{XOR}} = \max(N_l, N_r)/N$, where N_l and N_r are the number of robots located on the left and right area, respectively; and N is the total number of robots. The objective function is maximized when all robots are either on the left or the right area. The anytime objective function to be maximized is $A_{\text{XOR}} = \sum_{t=1}^{T} \max(N_l(t), N_r(t))/N$, where T is the duration of the mission. AggregationXOR has been studied in 3 of the works from which we collected data (Ligot et al., 2020b; Hasselmann et al., 2021; Ligot et al., 2022). In **Foraging**, the swarm must retrieve virtual items from food sources and bring them the nest. The food sources are represented by small black disks, the nest is represented by a white area. A robot is deemed to pick up an item when it enters a food source, and drop the item as soon as it then enters the nest (Fig. B.2b). A light source is placed behind the nest and can by used by the robots to orient themselves in the arena. The objective function to be maximized is $F_{\text{F}} = I$, where I is the number of items retrieved. Foraging has been studied in 4 of the works from which we collected data (Ligot et al., 2020b; Spaey et al., 2020; Hasselmann et al., 2021; Ligot et al., 2022). In one of these works (Spaey et al., 2020), a variant of the mission has been studied, which is characterized by an unbounded arena (Fig. B.2g). In **Homing**, the swarm must aggregate on an area designated as their home. The endtime objective function to be maximized is $F_{\text{H}} = N_{home}$, where N_{home} is the number of robots located on the aggregation area. The anytime objective function to be maximized is $F_{\text{H}} = \sum_{t=1}^{T} N_{home}(t)$, where T is the duration of the mission. This mission has been studied in different forms across the studies considered. In one of them (Francesca et al., 2015), the mission is called AAC—short for aggregation



Source: Ligot A, Birattari M (2022b)

Figure B.2: Illustrations of the arenas of some of the missions considered. (**A**) AggregationXOR. (**B**) Foraging. (**C**) Homing as studied by Francesca et al. (2015) under the name AAC. (**D**) DirectionalGate. (**E**) Decision. The circular area in the middle of the arena is either completely white or completely black. (**F**) Stop. (**G**) Unbounded arena. (**H**) Shelter. (**I**) SPC. The red glow in (**B**), (**C**), (**D**), (**E**), and (**H**) indicate the presence of a light source placed outside the south side of the arena that can be used by the robots to orient themselves.

with ambient cues—and is characterized by the presence of two aggregation areas of different colors and a source of light indicating to the robots on which area to aggregate (Fig. B.2c). In a second one (Hasselmann et al., 2018b), the mission is called AGGREGATION: also in this case, the arena contains two aggregation areas of different colors, but not light source. In a third one (Hasselmann et al., 2021), the mission is called HOMING: in this case, the arena contains only one aggregation area. In a fourth one (Spaey et al., 2020), the mission is called AGGREGATION: the arena contains only one aggregation area. The mission is studied twice: once in a closed arena and once with in an unbounded one (see Fig. B.2g). In **DIRECTIONALGATE**, the swarm must traverse a gate from North to South (Hasselmann et al., 2021) (see Fig. B.2d). The gate is positioned in the center of the arena and is identified by white floor. A light source is placed outside of the arena and in the axis of the gate to help the robots orientate themselves. A black corridor leads to the North entrance of the gate. The objective function to be maximized is $F_{\text{DG}} = K - \bar{K}$, where K is the number of times the robots traversed the gate in the right direction (North to South), and $\bar{K}$ the number of times they traversed it in the wrong direction (South to North). In **DECISION**, the swarm must aggregate on the right-hand side or the left-hand side of the arena depending on the color of a circular area positioned in the middle of the arena (Hasselmann and Birattari, 2020) (Fig. B.2e). In each experimental run, the circular area can be either black or white, with equal probability. A light source is placed outside the arena, at its right. The objective function, to be maximized, is $F_{\text{D}} = 24000 - \sum_{t=1}^{T} \sum_{i=1}^{N} I_i(t)$, where T is the duration of the experiment, N is the number of robots, and $I_i(t) = 0$ if robot i is in the correct half of the arena and $I_i(t) = 1$ otherwise. In **STOP**, the swarm must find a small circular white spot as soon as possible, and stop right after (Fig. B.2f). The objective function, to be maximized, is $F_{\text{S}} = 48000 - \left(\bar{t}N + \sum_{t=1}^{\bar{t}} \sum_{i=1}^{N} \bar{I}_i(t) + \sum_{t=\bar{t}+1}^{T} \sum_{i=1}^{N} I_i(t)\right)$, where $\bar{t}$ is the time at which the first robot finds the white spot, T is the duration of the experiment, N is the number of robots, $I_i(t) = 1$ if robot i is moving and $I_i(t) = 0$ otherwise, and $\bar{I}_i(t) = 1 - I_i(t)$. In **GRIDEXPLORATION**, the swarm must explore the arena and cover as much space as possible (Spaey et al., 2020). To measure the performance of the swarm, the arena is divided in a grid of 10 tiles by 10 tiles. For each tile, a counter c retains the time t elapsed since the last time the tile was visited by a robot. The counter is reset to 0 when a robot visits a tile. The objective function to be maximized is $F_{\text{GE}} = \sum_{t=1}^{T} \left(\frac{1}{N_{tiles}} \sum_{j=1}^{N_{tiles}} -c_j(t)\right)$, where T is the duration of the experiment, N_{tiles} is the number of tiles in the arena, and $c_j(t)$ is the value of the counter associated to tile j. The mission is studied twice (Spaey et al., 2020): once in a bounded arena, once in an unbounded one (Fig. B.2g). In **CFA**—short for coverage with forbidden areas—the swarm must cover the arena, avoiding the forbidden areas denoted by the three circular black areas (Francesca et al., 2015). The objective function to be minimized is $F_{\text{CFA}} = E\left[d(T)\right]$, where $E\left[d(T)\right]$ is the expected distance, at the end T of the experiment, between a generic point of the arena and the closest robot that is not in the

forbidden areas. In **LCN**—short for largest covering network—the swarm must create a connected network that covers the largest area possible in an empty arena (Francesca et al., 2015). Each robot covers a circular area of 0.35 m radius. Two robots are considered to be connected if they are separated by less than 0.35 m. The objective function to be maximized is $F_{\text{LCN}} = A_C$, where C is the largest network of connected robots, and A_C is the area covered by C. In **Shelter**, the swarm must aggregate in a rectangular white area surrounded by three walls and positioned in the center of the arena (Fig. B.2h). A light source is positioned outside the arena, in front of the open side of the shelter. The arena also features two black circular areas that do not have any predefined purpose/role in the definition of the mission: they are noise-features of the environment. The objective function to be maximized is $F_{\text{s}} \sum_{t=1}^{T} N(t)$, where T is the duration of the experiment and $N(t)$ is the number of robots in the shelter at time t. This mission has been studied in two of the works considered (Francesca et al., 2015; Hasselmann et al., 2021); in one of them (Francesca et al., 2015) it is studied under the name SCA—short for shelter with constrained access. In **SPC**—short for surface and perimeter coverage—the arena contains a square white area and a circular black area (Fig. B.2i). The swarm must cover the area of the white square and aggregate on the perimeter of the black circle (Francesca et al., 2015). The objective function to be minimized is $F_{\text{SPC}} = E\left[d_a(T)\right]/c_a + E\left[d_p(T)\right]/c_p$, where $E\left[d_a(T)\right]$ is the expected distance, at the end T of experiment, between a generic point in the square area and the closest robot in the square, and $E\left[d_p(T)\right]$ is the expected distance between a generic point on the circumference of the circular area and the closest robot that intersects the circumference. c_a and c_p are scaling factors fixed to 0.08 and 0.06, respectively. If no robot is on the surface of the square area and/or on the perimeter of the circular area, $E\left[d_a(T)\right]$ and/or $E\left[d_p(T)\right]$ are undefined and we thus assign an arbitrarily large value to F_{SPC}.

B.4 Data availability

All 1385 observations of real-world performance and predictions yield by P_{M_A}, P_{M_B}, and by the 1380 models uniformly sampled from the range R of pseudo-reality models are available in the DS 1 repository: https://doi.org/10.5281/zenodo.6501500.

The instances of control software that produced DS 1, as well as the source code necessary to execute them—that is, the design methods, the simulator and the dependencies, the configuration files of the missions, together with scripts to compile the sources, generate necessary files, and execute the instances of control software—are available in the following repository: https://doi.org/10.5281/zenodo.6501616.

Table B.2: Summary of the design methods. PFSM stands for probabilistic finite state machines, BT for behavior trees, MLP for multi layer perceptron, SLP for single layer perceptron, EA for evolutionary algorithm. NN stands for neural network for which the topology is evolved and thus varies for each design.

Method	Family	Control software architecture	Optimization algorithm	Reference model
`Arlequin`	Modular	PFSM	Iterated F-race Balaprakash et al. (2007) Birattari et al. (2010) López-Ibáñez et al. (2016)	RM1.1
`Chocolate`	Modular	PFSM	Iterated F-race	RM1.1
`CMA-ES-mlp`	Evolutionary	MLP	`CMA-ES` Hansen and Ostermeier (2001)	RM1.1
`CMA-ES-slp`	Evolutionary	SLP	`CMA-ES`	RM1.1
`Coconut`	Modular	PFSM	Iterated F-race	RM1.1
`C-Human`	Human	PFSM	∅	RM1.1
`EvoCom`	Evolutionary	SLP	EA	RM2
`EvoStick`	Evolutionary	SLP	EA	RM1.1
`Gianduja`	Modular	PFSM	Iterated F-race	RM2
`Maple`	Modular	BT	Iterated F-race	RM1.1
`NEAT-A-nl`	Evolutionary	NN	`NEAT` Stanley and Miikkulainen (2002)	RM1.1
`NEAT-A-slp`	Evolutionary	NN	`NEAT`	RM1.1
`NEAT-B-nl`	Evolutionary	NN	`NEAT`	RM1.1
`NEAT-B-slp`	Evolutionary	NN	`NEAT`	RM1.1
`RandomWalk`	Human	∅	∅	RM1.1
`U-Human`	Human	unconstrained	∅	RM1.1
`Vanilla`	Modular	PFSM	F-race Birattari (2009) Birattari et al. (2002)	RM1.1
`xNES-mlp`	Evolutionary	MLP	`xNES` Glasmachers et al. (2010)	RM1.1
`xNES-slp`	Evolutionary	MLP	`xNES`	RM1.1

Bibliography

Ampatzis C, Tuci E, Trianni V, Dorigo M (2006) Evolving communicating agents that integrate information over time: a real robot experiment. In: Talbi EG, Liardet P, Collet P, Lutton E, Schoenauer M (eds) Artificial Evolution. Seventh International Conference, Evolution Artificielle (EA 2005), Springer Verlag, Berlin, Germany, Lecture Notes in Computer Science, vol 3871, pp 248–254

Ampatzis C, Tuci E, Trianni V, Christensen AL, Dorigo M (2009) Evolving self-assembly in autonomous homogeneous robots: experiments with two physical robots. Artificial Life 15(4):465–484, DOI 10.1162/artl.2009.Ampatzis.013

Andrychowicz M, Baker B, Jozefowicz R, McGrew B, Pachocki J, Petron A, Plappert M, Powell G, Ray A, Schneider J, Sidor S, Tobin J, Welinder P, Weng L, Zaremba W (2020) Learning dexterous in-hand manipulation. The International Journal of Robotics Research 39(1):3–20, DOI 10.1177/0278364919887447

Balaprakash P, Birattari M, Stützle T (2007) Improvement strategies for the F-Race algorithm: sampling design and iterative refinement. In: Bartz-Beielstein T, Blesa MJ, Blum C, Naujoks B, Roli A, Rudolph G, Sampels M (eds) Hybrid Metaheuristics, 4th International Workshop, HM 2007, Springer, Berlin, Germany, LNCS, vol 4771, pp 108–122, DOI 10.1007/978-3-540-75514-2_9

Baldassarre G, Trianni V, Bonani M, Mondada F, Dorigo M, Nolfi S (2007) Self-organized coordinated motion in groups of physically connected robots. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 37(1):224–239, DOI 10.1109/TSMCB.2006.881299

Beal J, Dulman S, Usbeck K, Viroli M, Correll N (2012) Organizing the aggregate: languages for spatial computing. In: Marjan M (ed) Formal and Practical Aspects of Domain-Specific Languages: Recent Developments, IGI Global, Hershey, PA, USA, pp 436–501, DOI 10.4018/978-1-4666-2092-6.ch016

Beni G (2004) From swarm intelligence to swarm robotics. In: Şahin E, Spears WM (eds) Swarm Robotics, SAB, Springer, Berlin, Germany, LNCS, vol 3342, pp 1–9, DOI 10.1007/978-3-540-30552-1_1

Berman S, Kumar V, Nagpal R (2011) Design of control policies for spatially inhomogeneous robot swarms with application to commercial pollination. In: IEEE International Conference on Robotics and Automation, ICRA, IEEE, Piscataway, NJ, USA, pp 378–385, DOI 10.1109/ICRA.2011.5980440

Birattari M (2004) On the estimation of the expected performance of a metaheuristic on a class of instances. how many instances, how many runs? Tech. Rep. TR/IRIDIA/2004-01, IRIDIA, Université libre de Bruxelles, Belgium

Birattari M (2009) Tuning Metaheuristics: A Machine Learning Perspective. Springer, Berlin, Germany, DOI 10.1007/978-3-642-00483-4

Birattari M (2020) Notes on the estimation of the expected performance of automatic methods for the design of control software for robot swarms. Tech. Rep. TR/IRIDIA/2020-10, IRIDIA, Université libre de Bruxelles, Belgium

Birattari M, Stützle T, Paquete L, Varrentrapp K (2002) A racing algorithm for configuring metaheuristics. In: Langdon WB, Cantú-Paz E, Mathias K, Roy R, Davis D, Poli R, Balakrishnan K, Honavar V, Rudolph G, Wegener J, Bull L, Potter MA, Schultz AC, Miller JF, Burke EK, Jonoska N (eds) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO, Morgan Kaufmann Publishers, San Francisco, CA, USA, pp 11–18

Birattari M, Yuan Z, Balaprakash P, Stützle T (2010) F-Race and Iterated F-Race: an overview. In: Bartz-Beielstein T, Chiarandini M, Paquete L, Preuss M (eds) Experimental Methods for the Analysis of Optimization Algorithms, Springer, Berlin, Germany, pp 311–336, DOI 10.1007/978-3-642-02538-9_13

Birattari M, Delhaisse B, Francesca G, Kerdoncuff Y (2016) Observing the effects of overdesign in the automatic design of control software for robot swarms. In: Dorigo M, Birattari M, Li X, López-Ibáñez M, Ohkura K, Pinciroli C, Stützle T (eds) Swarm Intelligence – ANTS, Springer, Cham, Switzerland, Lecture Notes in Computer Science, vol 9882, pp 45–57, DOI 10.1007/978-3-319-44427-7_13

Birattari M, Ligot A, Bozhinoski D, Brambilla M, Francesca G, Garattoni L, Garzón Ramos D, Hasselmann K, Kegeleirs M, Kuckling J, Pagnozzi F, Roli A, Salman M, Stützle T (2019) Automatic off-line design of robot swarms: a manifesto. Frontiers in Robotics and AI 6:59, DOI 10.3389/frobt.2019.00059

Birattari M, Ligot A, Hasselmann K (2020) Disentangling automatic and semi-automatic approaches to the optimization-based design of control software for robot swarms. Nature Machine Intelligence 2(9):494–499, DOI 10.1038/s42256-020-0215-0

Birattari M, Ligot A, Francesca G (2021) Automode: a modular approach to the automatic off-line design and fine-tuning of control software for robot swarms. In:

Pillay N, Qu R (eds) Automated Design of Machine Learning and Search Algorithms, Natural Computing Series, Springer, Cham, Switzerland, DOI 10.1007/978-3-030-72069-8_5

Boeing A, Bräunl T (2012) Leveraging multiple simulators for crossing the reality gap. In: Proceedings of the International Conference on Control, Automation, Robotics and Vision – ICARCV, IEEE, Piscataway, NJ, USA, pp 1113–1119, DOI 10.1109/ICARCV.2012.6485313

Bongard JC, Lipson H (2004) Once more unto the breach: co-evolving a robot and its simulator. In: Pollack JB, Bedau MA, Husbands P, Watson RA, Ikegami T (eds) Artificial Life IX: Proceedings of the Conference on the Simulation and Synthesis of Living Systems, MIT Press, Cambridge, MA, USA, pp 57–62, a Bradford Book

Bozhinoski D, Di Ruscio D, Malavolta I, Pelliccione P, Tivoli M (2015) Flyaq: enabling non-expert users to specify and generate missions of autonomous multicopters. In: Cohen M, Grunske L, Whalen M (eds) Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, Piscataway, NJ, USA, pp 801–806, DOI 10.1109/ASE.2015.104

Brambilla M, Ferrante E, Birattari M, Dorigo M (2013) Swarm robotics: a review from the swarm engineering perspective. Swarm Intelligence 7(1):1–41, DOI 10.1007/s11721-012-0075-2

Brambilla M, Brutschy A, Dorigo M, Birattari M (2014) Property-driven design for swarm robotics: a design method based on prescriptive modeling and model checking. ACM Transactions on Autonomous Adaptive Systems 9(4):17:1–17:28, DOI 10.1145/2700318

Bredeche N, Montanier JM, Liu W, Winfield A (2012) Environment-driven distributed evolutionary adaptation in a population of autonomous robotic agents. Mathematical and Computer Modelling of Dynamical Systems 18(1):101–129, DOI 10.1080/13873954.2011.601425

Bredeche N, Haasdijk E, Prieto A (2018) Embodied evolution in collective robotics: a review. Frontiers in Robotics and AI 5:12, DOI 10.3389/frobt.2018.00012

Brooks RA (1991) Intelligence without representation. Artificial Intelligence 47:139–159, DOI 10.1007/s00422-006-0080-x

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

Brugali D (ed) (2007) Software Engineering for Experimental Robotics. Springer Tracts in Advanced Robotics, Springer, Berlin, Germany, DOI 10.1007/978-3-540-68951-5

Bühlmann H (1967) Experience rating and credibility. ASTIN Bulletin: The Journal of the IAA 4(3):199–207

Cambier N, Ferrante E (2022) Automode-pomodoro: an evolutionary class of modular designs. In: Fieldsend JE (ed) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO, ACM, New York, NY, USA, 8, pp 100–103, DOI 10.1145/3520304.3529031

Caruana R, Lawrence S, Giles CL (2001) Overfitting in neural nets: backpropagation, conjugate gradient, and early stopping. In: Leen TK, Dietterich TG, Tresp V (eds) Advances in Neural Information Processing Systems 13, MIT Press, Cambridge, MA, USA, pp 402–408

Chambers JM, Cleveland WS, Kleiner B, Tukey PA (1983) Graphical Methods For Data Analysis. CRC Press, Belmont, CA, USA

Champandard AJ (2007) Understanding behavior trees. http://aigamedev.com/open/articles/bt-overview/

Champandard AJ, Dawe M, Hernandez-Cerpa D (2010) Behavior trees: three ways of cultivating game AI. https://www.gdcvault.com/play/1012744/Behavior-Trees-Three-Ways-of, game Developers Conference, AI Summit

Christensen AL, Dorigo M (2006) Evolving an integrated phototaxis and hole-avoidance behavior for a swarm-bot. In: Rocha LM, Yaeger LS, Bedau MA, Floreano D, Goldstone RL, Vespignani A (eds) Artificial Life X: Proceedings of the Tenth International Conference on the Simulation and Synthesis of Living Systems, MIT Press, Cambridge, MA, USA, pp 248–254, a Bradford Book

Colledanchise M, Ögren P (2018) Behavior Trees in Robotics and AI: An Introduction, 1st edn. Chapman & Hall/CRC Artificial Intelligence and Robotics Series, CRC Press, Boca Raton, FL, DOI 10.1201/9780429489105

Cotorruelo A, Ligot A, Garone E, Birattari M (2021) Minimizing the variance in the estimation of the performance of a method for the fully-automatic design of robot swarms: a mathematical proof. Tech. Rep. TR/IRIDIA/2021-007, IRIDIA, Université libre de Bruxelles, Belgium

Şahin E (2004) Swarm robotics: from sources of inspiration to domains of application. In: Şahin E, Spears WM (eds) Swarm Robotics, SAB, Springer, Berlin, Germany, LNCS, vol 3342, pp 10–20, DOI 10.1007/978-3-540-30552-1_2

Di Mario E, Martinoli A (2014) Distributed particle swarm optimization for limited-time adaptation with real robots. Robotica 32(2):193–208, DOI 10.1017/S026357471300101X

Di Mario E, Navarro I, Martinoli A (2015) A distributed noise-resistant particle swarm optimization algorithm for high-dimensional multi-robot learning. In: IEEE International Conference on Robotics and Automation, ICRA, IEEE, Piscataway, NJ, USA, pp 5970–5976, DOI 10.1109/ICRA.2015.7140036

Di Ruscio D, Malavolta I, Pelliccione P (2014) A family of domain-specific languages for specifying civilian missions of multi-robot systems. In: Aßmann U, Wagner G (eds) Proceedings of the 1st International Workshop on Model-Driven Robot Software Engineering – MORSE, CEUR-WS, Aachen, Germany, vol 1319, pp 13–26

Doncieux S, Mouret JB (2014) Beyond black-box optimization: a review of selective pressures for evolutionary robotics. Evolutionary Intelligence 7(2):71–93, DOI 10.1007/s12065-014-0110-x

Doncieux S, Bredeche N, Mouret JB, Eiben A (2015) Evolutionary robotics: what, why, and where to. Frontiers in Robotics and AI 2:4, DOI 10.3389/frobt.2015.00004

Dorigo M, Birattari M (2007) Swarm intelligence. Scholarpedia 2(9):1462, DOI 10.4249/scholarpedia.1462

Dorigo M, Trianni V, Şahin E, Groß R, Labella H Thomas, Baldassarre G, Nolfi S, Deneubourg JL, Mondada F, Floreano D, Gambardella LM (2003) Evolving self-organizing behaviors for a Swarm-bot. Autonomous Robots 17:223–245, DOI 10.1023/B:AURO.0000033973.24945.f3

Dorigo M, Floreano D, Gambardella LM, Mondada F, Nolfi S, Baaboura T, Birattari M, Bonani M, Brambilla M, Brutschy A, Burnier D, Campo A, Christensen AL, Decugnière A, Di Caro GA, Ducatelle F, Ferrante E, Förster A, Martinez Gonzales J, Guzzi J, Longchamp V, Magnenat S, Mathews N, Montes de Oca M, O'Grady R, Pinciroli C, Pini G, Retornaz P, Roberts J, Sperati V, Stirling T, Stranieri A, Stützle T, Trianni V, Tuci E, Turgut AE, Vaussard F (2013) Swarmanoid: a novel concept for the study of heterogeneous robotic swarms. IEEE Robotics & Automation Magazine 20(4):60–71, DOI 10.1109/MRA.2013.2252996

Dorigo M, Birattari M, Brambilla M (2014) Swarm robotics. Scholarpedia 9(1):1463, DOI 10.4249/scholarpedia.1463

Dorigo M, Theraulaz G, Trianni V (2020) Reflections on the future of swarm robotics. Science Robotics 5:eabe4385, DOI 10.1126/scirobotics.abe4385

Dorigo M, Theraulaz G, Trianni V (2021) Swarm robotics: past, present, and future [point of view]. Proceedings of the IEEE 109(7):1152–1165, DOI 10.1109/JPROC.2021.3072740

Duarte M, Oliveira SM, Christensen AL (2014) Evolution of hierarchical controllers for multirobot systems. In: Sayama H, Rieffel J, Risi S, Doursat R, Lipson H (eds)

Artificial Life 14. Proceedings of the Fourteenth International Conference on the Synthesis and Simulation of Living Systems, MIT Press, Cambridge, MA, USA, pp 657–664, DOI 10.7551/978-0-262-32621-6-ch105

Duarte M, Oliveira SM, Christensen AL (2015) Evolution of hybrid robotic controllers for complex tasks. Journal of Intelligent & Robotic Systems 78(3):463–484

Duarte M, Costa V, Gomes J, Rodrigues T, Silva F, Oliveira SM, Christensen AL (2016) Evolution of collective behaviors for a real swarm of aquatic surface robots. PLOS ONE 11(3):e0151834, DOI 10.1371/journal.pone.0151834

Fay III RE, Herriot RA (1979) Estimates of income for small places: an application of james-stein procedures to census data. Journal of the American Statistical Association 74(366a):269–277

Floreano D, Mondada F (1996) Evolution of plastic neurocontrollers for situated agents. In: Maes P, Matarić MJ, Meyer JA, Pollack JB, Wilson SW (eds) From Animals to Animats 4. Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior, SAB, MIT Press, Cambridge, MA, USA, pp 402–410, DOI 10.7551/mitpress/3118.003.0049

Floreano D, Urzelai J (2000) Evolutionary robots with on-line self-organization and behavioral fitness. Neural Networks 13:431–443, DOI 10.1016/S0893-6080(00)00032-0

Floreano D, Urzelai J (2001) Evolution of plastic control networks. Autonomous Robots 11(3):311–317, DOI 10.1023/A:1012459627968

Floreano D, Husbands P, Nolfi S (2008) Evolutionary robotics. In: Siciliano B, Khatib O (eds) Springer Handbook of Robotics, Springer Handbooks, Springer, Berlin, Heidelberg, Germany, pp 1423–1451, DOI 10.1007/978-3-540-30301-5_62, first edition

Francesca G, Birattari M (2016) Automatic design of robot swarms: achievements and challenges. Frontiers in Robotics and AI 3(29):1–9, DOI 10.3389/frobt.2016.00029

Francesca G, Brambilla M, Trianni V, Dorigo M, Birattari M (2012) Analysing an evolved robotic behaviour using a biological model of collegial decision making. In: Ziemke T, Balkenius C, Hallam J (eds) From Animals to Animats 12. Proceedings of the twelveth International Conference on Simulation of Adaptive Behavior, SAB, Springer, Berlin, Germany, Lecture Notes in Computer Science, vol 7426, pp 381–390, DOI 10.1007/978-3-642-33093-3_38

Francesca G, Brambilla M, Brutschy A, Garattoni L, Miletitch R, Podevijn G, Reina A, Soleymani T, Salvaro M, Pinciroli C, Trianni V, Birattari M (2014a) An experiment in automatic design of robot swarms: AutoMoDe-Vanilla, EvoStick, and human experts. In: Dorigo M, Birattari M, Garnier S, Hamann H, Montes de Oca M, Solnon

C, Stützle T (eds) Swarm Intelligence – ANTS, Springer International Publishing, Cham, Switzerland, LNCS, vol 8667, pp 25–37, DOI 10.1007/978-3-319-09952-1_3

Francesca G, Brambilla M, Brutschy A, Trianni V, Birattari M (2014b) AutoMoDe: a novel approach to the automatic design of control software for robot swarms. Swarm Intelligence 8(2):89–112, DOI 10.1007/s11721-014-0092-4

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

Ganguli M (1941) A note on nested sampling. Sankhyā: The Indian Journal of Statistics pp 449–452

Garattoni L, Birattari M (2016) Swarm robotics. In: Webster JG (ed) Wiley Encyclopedia of Electrical and Electronics Engineering, John Wiley & Sons, Hoboken, NJ, USA, pp 1–19, DOI 10.1002/047134608X.W8312

Garattoni L, Birattari M (2018) Autonomous task sequencing in a robot swarm. Science Robotics 3(20):eaat0430, DOI 10.1126/scirobotics.aat0430

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

Garzón Ramos D, Birattari M (2020) Automatic design of collective behaviors for robots that can display and perceive colors. Applied Sciences 10(13):4654, DOI 10.3390/app10134654

Gauci M, Chen J, Li W, Dodd TJ, Groß R (2014a) Clustering objects with robots that do not compute. In: Proceedings of the 2014 International Conference on Autonomous Agents and Multi-Agent Systems – AAMAS2014, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, USA, AAMAS '14, pp 421–428, DOI 10.5555/2615731.2615800

Gauci M, Chen J, Li W, Dodd TJ, Groß R (2014b) Self-organized aggregation without computation. The International Journal of Robotics Research 33(8):1145–1161, DOI 10.1177/0278364914525244

Geman S, Bienenstock E, Doursat R (1992) Neural networks and the bias/variance dilemma. Neural Computation 4(1):1–58, DOI 10.1162/neco.1992.4.1.1

Glasmachers T, Schaul T, Yi S, Wierstra D, Schmidhuber J (2010) Exponential natural evolution strategies. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, GECCO, ACM, pp 393–400, DOI 10.1145/1830483.1830557

Gomes J, Christensen AL (2018) Task-agnostic evolution of diverse repertoires of swarm behaviours. In: Dorigo M, Birattari M, Blum C, Christensen AL, Reina A, Trianni V (eds) Swarm Intelligence – ANTS, Springer, Cham, Switzerland, LNCS, vol 11172, pp 225–238, DOI 10.1007/978-3-030-00533-7_18

Gongora MA, Passow BN, Hopgood AA (2009) Robustness analysis of evolutionary controller tuning using real systems. In: IEEE Congress on Evolutionary Computation, CEC, IEEE Press, Piscataway, NJ, USA, pp 606–613, DOI 10.1109/CEC.2009.4983001

Goodfellow I, Bengio Y, Courville A (2016) Deep Learning, 1st edn. MIT Press, Cambridge, MA, USA

Gutiérrez Á, Campo A, Dorigo M, Donate J, Monasterio-Huelin F, Magdalena L (2009) Open e-puck range & bearing miniaturized board for local communication in swarm robotics. In: Kosuge K (ed) IEEE International Conference on Robotics and Automation, ICRA, IEEE, Piscataway, NJ, USA, pp 3111–3116, DOI 10.1109/ROBOT.2009.5152456

Haasdijk E, Bredeche N, Eiben A (2014) Combining environment-driven adaptation and task-driven optimisation in evolutionary robotics. PLOS ONE 9(6):e98466, DOI 10.1371/journal.pone.0098466

Hamann H (2018) Swarm robotics: a formal approach. Springer, Cham, Switzerland, DOI 10.1007/978-3-319-74528-2

Hamann H, Wörn H (2008) A framework of space–time continuous models for algorithm design in swarm robotics. Swarm Intelligence 2(2–4):209–239, DOI 10.1007/s11721-008-0015-3

Hansen N, Ostermeier A (2001) Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation 9(2):159–195, DOI 10.1162/106365601750190398

Hardeo S, Ojeda MM (2005a) Three-way nested classification. In: Analysis of Variance for Random Models: Unbalanced Data, Birkhäuser, Boston, MA, USA, pp 329–369

Hardeo S, Ojeda MM (2005b) Two-way nested classification. In: Analysis of Variance for Random Models, vol 2, Birkhäuser, Boston, MA, USA, pp 287–328

Hasselmann K, Birattari M (2020) Modular automatic design of collective behaviors for robots endowed with local communication capabilities. PeerJ Computer Science 6:e291, DOI 10.7717/peerj-cs.291

Hasselmann K, Ligot A, Francesca G, Garzón Ramos D, Salman M, Kuckling J, Mendiburu FJ, Birattari M (2018a) Reference models for AutoMoDe. Tech. Rep. TR/IRIDIA/2018-002, IRIDIA, Université libre de Bruxelles, Belgium

Hasselmann K, Robert F, Birattari M (2018b) Automatic design of communication-based behaviors for robot swarms. In: Dorigo M, Birattari M, Garnier S, Hamann H, Montes de Oca M, Solnon C, Stützle T (eds) Swarm Intelligence – ANTS, Springer, Cham, Switzerland, LNCS, vol 11172, pp 16–29, DOI 10.1007/978-3-030-00533-7_2

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

Hastie T, Tibshirani R, Friedman J (2009) The Elements of Statistical Learning: Data mining, Inference and Prediction, 2nd edn. Springer, Berlin, Germany

Hauert S, Zufferey JC, Floreano D (2009) Evolved swarming without positioning information: an application in aerial communication relay. Autonomous Robots 26(1):21–32, DOI 10.1007/s10514-008-9104-9

Jakobi N (1997) Evolutionary robotics and the radical envelope-of-noise hypothesis. Adaptive Behavior 6(2):325–368, DOI 10.1177/105971239700600205

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

Jakobi N, Husbands P, Harvey I (1995) Noise and the reality gap: the use of simulation in evolutionary robotics. In: Morán F, Moreno A, Merelo JJ, Chacón P (eds) Advances in Artificial Life: Third european conference on artificial life, Springer, Berlin, Germany, Lecture Notes in Artificial Intelligence, vol 929, pp 704–720, DOI 10.1007/3-540-59496-5_337

Jones S, Studley M, Hauert S, Winfield A (2018) Evolving behaviour trees for swarm robotics. In: Groß R, Kolling A, Berman S, Frazzoli E, Martinoli A, Matsuno F, Gauci M (eds) Distributed Autonomous Robotic Systems (DARS), Springer, Cham, Switzerland, SPAR, vol 6, pp 487–501, DOI 10.1007/978-3-319-73008-0_34

Jones S, Winfield A, Hauert S, Studley M (2019) Onboard evolution of understandable swarm behaviors. Advanced Intelligent Systems 1(6):1900031, DOI 10.1002/aisy.201900031

Kaiser TK, Hamann H (2022) Innate motivation for robot swarms by minimizing surprise: From simple simulations to real-world experiments. IEEE Transactions on Robotics 38(6):3582–3601, DOI 10.1109/TRO.2022.3181004

Kazadi S (2009) Model independence in swarm robotics. International Journal of Intelligent Computing and Cybernetics 2(4):672–694, DOI 10.1108/17563780911005836

Kerry R, Oliver M, Frogbrook Z (2010) Sampling in precision agriculture. In: Geostatistical applications for precision agriculture, Springer, pp 35–63

Koos S, Mouret JB, Doncieux S (2013) The transferability approach: crossing the reality gap in evolutionary robotics. IEEE Transactions on Evolutionary Computation 17(1):122–145, DOI 10.1109/TEVC.2012.2185849

Kuckling J, Ligot A, Bozhinoski D, Birattari M (2018) Behavior trees as a control architecture in the automatic modular design of robot swarms. In: Dorigo M, Birattari M, Blum C, Christensen AL, Reina A, Trianni V (eds) Swarm Intelligence – ANTS, Springer, Cham, Switzerland, LNCS, vol 11172, pp 30–43, DOI 10.1007/978-3-030-00533-7_3

Kuckling J, Ubeda Arriaza K, Birattari M (2020) AutoMoDe-IcePop: automatic modular design of control software for robot swarms using simulated annealing. In: Bogaerts B, Bontempi G, Geurts P, Harley N, Lebichot B, Lenaerts T, Louppe G (eds) Artificial Intelligence and Machine Learning: BNAIC 2019, BENELEARN 2019, CCIS, vol 1196, Springer, Cham, Switzerland, pp 3–17

König L, Mostaghim S, Schmeck H (2009) Decentralized evolution of robotic behavior using finite state machines. International Journal of Intelligent Computing and Cybernetics 2(4):695–723, DOI 10.1108/17563780911005845

Li S, Batra R, Brown D, Chang HD, Ranganathan N, Hoberman C, Rus D, Lipson H (2019) Particle robotics based on statistical mechanics of loosely coupled components. Nature 567(7748):361–365, DOI 10.1038/s41586-019-1022-9

Ligot A, Birattari M (2018) On mimicking the effects of the reality gap with simulation-only experiments. In: Dorigo M, Birattari M, Garnier S, Hamann H, Montes de Oca M, Solnon C, Stützle T (eds) Swarm Intelligence – ANTS, Springer, Cham, Switzerland, LNCS, vol 11172, pp 109–122, DOI 10.1007/978-3-030-00533-7_9

Ligot A, Birattari M (2019) Simulation-only experiments to mimic the effects of the reality gap in the automatic design of robot swarms. Swarm Intelligence pp 1–24, DOI 10.1007/s11721-019-00175-w

Ligot A, Birattari M (2022a) DS1. *Zenodo*, DOI https://doi.org/10.5281/zenodo.6501500

Ligot A, Birattari M (2022b) On using simulation to predict the performance of robot swarms. Scientific Data 9(788), DOI 10.1038/s41597-022-01895-1

Ligot A, Birattari M (2022c) Supplementary material for the thesis: Assessing and forecasting the performance of automatic methods for the design of robot swarms. Available at http://iridia.ulb.ac.be/supp/IridiaSupp2022-004/

Ligot A, Hasselmann K, Delhaisse B, Garattoni L, Francesca G, Birattari M (2017) AutoMoDe, NEAT, and EvoStick: implementations for the e-puck robot in ARGoS3. Tech. Rep. TR/IRIDIA/2017-002, IRIDIA, Université libre de Bruxelles, Belgium

Ligot A, Hasselmann K, Birattari M (2020a) AutoMoDe-Arlequin: neural networks as behavioral modules for the automatic design of probabilistic finite state machines. In: Dorigo M, Stützle T, Blesa MJ, Blum C, Hamann H, Heinrich MK, Strobel V (eds) Swarm Intelligence – ANTS, Springer, Cham, Switzerland, LNCS, vol 12421, pp 271–281, DOI 10.1007/978-3-030-60376-2_21

Ligot A, Kuckling J, Bozhinoski D, Birattari M (2020b) Automatic modular design of robot swarms using behavior trees as a control architecture. PeerJ Computer Science 6:e314, DOI 10.7717/peerj-cs.314

Ligot A, Cotorruelo A, Garone E, Birattari M (2022) Towards an empirical practice in off-line fully-automatic design of robot swarms. IEEE Transactions on Evolutionary Computation 26(6):1236–1245, DOI 10.1109/TEVC.2022.3144848

Lipson H (2005) Evolutionary robotics and open-ended design automation. In: Biomimetics: Biologically Inspired Technologies, vol 17, CRC Press, Boca Raton, FL, pp 129–155

López-Ibáñez M, Dubois-Lacoste J, Pérez Cáceres L, Birattari M, Stützle T (2016) The irace package: iterated racing for automatic algorithm configuration. Operations Research Perspectives 3:43–58, DOI 10.1016/j.orp.2016.09.002

Marcuse S (1949) Optimum allocation and variance components in nested sampling with an application to chemical analysis. Biometrics 5(3):189–206

Miglino O, Lund HH, Nolfi S (1995) Evolving mobile robots in simulated and real environments. Artificial Life 2(4):417–434, DOI 10.1162/artl.1995.2.4.417

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

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

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

Mumuni A, Mumuni F (2022) Data augmentation: A comprehensive survey of modern approaches. Array 16:100258, DOI https://doi.org/10.1016/j.array.2022.100258

Nolfi S, Floreano D (2000) Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines, 1st edn. MIT Press, Cambridge, MA, USA, a Bradford Book

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

Nouyan S, Campo A, Dorigo M (2008) Path formation in a robot swarm: self-organized strategies to find your way home. Swarm Intelligence 2(1):1–23, DOI 10.1007/s11721-007-0009-6

Pagliuca P, Nolfi S (2019) Robust optimization through neuroevolution. PLOS ONE 14(3):e0213193, DOI 10.1371/journal.pone.0213193

Parker LE (1997) L-ALLIANCE: ask-oriented multi-robot learning in behavior-based systems. Advanced Robotics 11:305–322

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

Pinciroli C, Beltrame G (2016) Buzz: a programming language for robot swarms. IEEE Software 33(4):97–100, DOI 10.1109/MS.2016.95

Pinciroli C, Trianni V, O'Grady R, Pini G, Brutschy A, Brambilla M, Mathews N, Ferrante E, Di Caro GA, Ducatelle F, Birattari M, Gambardella LM, Dorigo M (2012) ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. Swarm Intelligence 6(4):271–295, DOI 10.1007/s11721-012-0072-5

Quinn M, Smith L, Mayley G, Husbands P (2003) Evolving controllers for a homogeneous system of physical robots: structured cooperation with minimal sensors. Philosophical Transactions of the Royal Society of London Series A: Mathematical, Physical and Engineering Sciences 361(1811):2321–2343, DOI 10.1098/rsta.2003.1258

Regan W, van Beugel F, Lipson H (2006) Towards evolvable hovering flight on a physical ornithopter. In: Rocha LM, Yaeger LS, Bedau MA, Floreano D, Goldstone RL, Vespignani A (eds) Artificial Life X: Proceedings of the Tenth International Conference on the Simulation and Synthesis of Living Systems, MIT Press, Cambridge, MA, USA, Complex Adaptive Systems

Reina A, Valentini G, Fernández-Oto C, Dorigo M, Trianni V (2015) A design pattern for decentralised decision making. PLOS ONE 10(10):e0140950, DOI 10.1371/journal.pone.0140950

Rubenstein M, Cornejo A, Nagpal R (2014) Programmable self-assembly in a thousand-robot swarm. Science 345(6198):795–799, DOI 10.1126/science.1254295

Salman M, Ligot A, Birattari M (2019) Concurrent design of control software and configuration of hardware for robot swarms under economic constraints. PeerJ Computer Science 5:e221, DOI 10.7717/peerj-cs.221

Schlegel C, Lotz A, Lutz M, Stampfer D, Inglés-Romero JF, Vicente-Chicote C (2015) Model-driven software systems engineering in robotics: covering the complete lifecycle of a robot. it - Information Technology 57(2):85–98, DOI 10.1515/itit-2014-1069

Schranz M, Umlauft M, Sende M, Elmenreich W (2020) Swarm robotic behaviors and current applications. Frontiers in Robotics and AI 7:36, DOI 10.3389/frobt.2020.00036

Silva F, Urbano P, Correia L, Christensen AL (2015) odNEAT: an algorithm for decentralised online evolution of robotic controllers. Evolutionary Computation 23(3):421–449, DOI 10.1162/EVCO_a_00141

Silva F, Duarte M, Correia L, Oliveira SM, Christensen AL (2016) Open issues in evolutionary robotics. Evolutionary Computation 24(2):205–236, DOI 10.1162/EVCO_a_00172

Sokal RR, Rohlf FJ, et al. (1995) Biometry: the principles and practice of statistics in biological research

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

Spaey G, Kegeleirs M, Garzón Ramos D, Birattari M (2020) Evaluation of alternative exploration schemes in the automatic modular design of robot swarms. In: Bogaerts B, Bontempi G, Geurts P, Harley N, Lebichot B, Lenaerts T, Louppe G (eds) Artificial Intelligence and Machine Learning: BNAIC 2019, BENELEARN 2019, CCIS, vol 1196, Springer, Cham, Switzerland, pp 18–33, DOI 10.1007/978-3-030-65154-1_2

Stanley KO, Miikkulainen R (2002) Evolving neural networks through augmenting topologies. Evolutionary Computation 10(2):99–127, DOI 10.1162/106365602320169811

Tabor A (2017) Mars rover tests driving, drilling and detecting life in chile's high desert. https://www.nasa.gov/feature/ames/mars-rover-tests-driving-drilling-and-detecting-life-in-chile-s-high-desert, accessed: 2022-03-11

Trianni V (2008) Evolutionary Swarm Robotics. Springer, Berlin, Germany, DOI 10.1007/978-3-540-77612-3

Trianni V (2014) Evolutionary robotics: model or design? Frontiers in Robotics and AI 1:13, DOI 10.3389/frobt.2014.00013

Trianni V, Dorigo M (2006) Self-organisation and communication in groups of simulated and physical robots. Biological Cybernetics 95:213–231, DOI 10.1007/s00422-006-0080-x

Trianni V, Nolfi S (2009) Self-organizing sync in a robotic swarm: a dynamical system view. IEEE Transactions on Evolutionary Computation 13(4):722–741, DOI 10.1109/TEVC.2009.2015577

Trianni V, Labella H Thomas, Dorigo M (2004) Evolution of direct communication for a swarm-bot performing hole avoidance. In: Dorigo M, Birattari M, Blum C, Gambardella LM, Mondada F, Stützle T (eds) Ant Colony Optimization and Swarm Intelligence – ANTS 2004, LNCS, vol 3172, Springer, Berlin, Heidelberg, pp 130–141, DOI 10.1007/978-3-540-28646-2_12

Tuci E, Trianni V, Dorigo M (2004) 'feeling' the flow of time through sensory/motor coordination. Connection Science 16:1–24

Urzelai J, Floreano D (2000) Evolutionary robotics: coping with environmental change. In: Whitley LD, Goldberg DE, Cantú-Paz E, Spector L, Parmee IC (eds) Proceedings of Conference on the Genetic and Evolutionary Computation Conference, GECCO, Morgan Kaufmann Publishers, San Francisco, CA, USA, pp 941–948

Usui Y, Arita T (2003) Situated and embodied evolution in collective evolutionary robotics. In: Proceedings of the 8th International Symposium on Artificial Life and Robotics, pp 212–215

Wahby M, Hamann H (2015) On the tradeoff between hardware protection and optimization success: A case study in onboard evolutionary robotics for autonomous parallel parking. In: Mora AM, Squillero G (eds) Applications of Evolutionary Computation, Springer International Publishing, Cham, Switzerland, pp 759–770

Waibel M, Keller L, Floreano D (2009) Genetic team composition and level of selection in the evolution of multi-agent systems. IEEE Transactions on Evolutionary Computation 13(3):648–660, DOI 10.1109/TEVC.2008.2011741

Watson RA, Ficici SG, Pollack JB (1999) Embodied evolution: embodying an evolutionary algorithm in a population of robots. In: IEEE Congress on Evolutionary Computation, CEC, IEEE, Piscataway, NJ, USA, vol 1, pp 335–342, DOI 10.1109/CEC.1999.781944

Watson RA, Ficici SG, Pollack JB (2002) Embodied evolution: distributing an evolutionary algorithm in a population of robots. Robotics and Autonomous Systems 39(1):1–18, DOI 10.1016/S0921-8890(02)00170-7

Werfel J, Petersen K, Nagpal R (2014) Designing collective behavior in a termite-inspired robot construction team. Science 343(6172):754–758, DOI 10.1126/science.1245842

Yang GZ, Bellingham J, Dupont PE, Fischer P, Floridi L, Full R, Jacobstein N, Kumar V, McNutt M, Merrifield R, Nelson BJ, Scassellati B, Taddeo M, Taylor R, Veloso M, Wang ZL, Wood R (2018) The grand challenges of Science Robotics. Science Robotics 3(14):eaar7650, DOI 10.1126/scirobotics.aar7650

Zagal JC, Ruiz-del Solar J (2007) Combining simulation and reality in evolutionary robotics. Journal of Intelligent & Robotic Systems 50(1):19–39, DOI 10.1007/s10846-007-9149-6

Zagal JC, Ruiz-del Solar J, Vallejos P (2004) Back to reality: crossing the reality gap in evolutionary robotics. IFAC Proceedings Volumes 37(8):834–839, DOI 10.1016/S1474-6670(17)32084-0

Zhou X, Wen X, Wang Z, Gao Y, Li H, Wang Q, Yang T, Lu H, Cao Y, Xu C, Gao F (2022) Swarm of micro flying robots in the wild. Science Robotics 7(66):eabm5954, DOI 10.1126/scirobotics.abm5954

Zhu X, Vondrick C, Fowlkes C, Ramanan D (2016) Do we need more training data? International Journal of Computer Vision 119:76–92, DOI https://doi.org/10.1007/s11263-015-0812-2