ECOLE
POLYTECHNIQUE
DE BRUXELLES

Advances in the automatic modular design of control software for robot swarms

Using neuroevolution to generate modules

Thesis presented by Ken HASSELMANN

in fulfilment of the requirements of the PhD Degree in Engineering Sciences 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 Artificielle

European Commission

Horizon 2020
European Union funding
for Research & Innovation

DEMIURGE
IRIDIA – Université libre de Bruxelles

erc
European Research Council
Established by the European Commission

Advances in the automatic modular design of control software for robot swarms: using neuroevolution to generate modules.

Ken Hasselmann

IRIDIA, Université libre de Bruxelles, Belgium.

2023

Composition of the jury

Prof. Marco DORIGO (chair)
Research Director of the FRS-FNRS and co-director of IRIDIA, École polytechnique de Bruxelles, Université libre de Bruxelles, Belgium.

Prof. Hugues BERSINI (secretary)
Full Professor and co-director of IRIDIA, École polytechnique de Bruxelles, Université libre de Bruxelles, Belgium.

Prof. Elio TUCI
Associate Professor in the Department of Computer Science, University of Namur, Belgium.

Prof. Anders Lyhne CHRISTENSEN
Full Professor at SDU Biorobotics, MMMI, University of Southern Denmark, Denmark.

Dr. Mary Katherine HEINRICH
Postdoctoral fellow at IRIDIA, École polytechnique de Bruxelles, Université libre de Bruxelles, Belgium.

Prof. Mauro BIRATTARI (supervisor)
Research Director of the FRS-FNRS at IRIDIA, École polytechnique de Bruxelles, Université libre de Bruxelles, Belgium.

The thesis

Using neuroevolution to generate the modules of automatic modular methods reduces the amount of expert knowledge needed to implement them, without impacting negatively their robustness to the reality gap.

Summary

In swarm robotics, the design problem is one of the most pressing issues of today's research. Robots in a swarm must be autonomous and are usually individually programmed. However, the collective behavior of the swarm, which is expected to emerge from the interactions between the robots and between the robots and their environment, is best described at the level of the swarm. A great deal of effort has been devoted in an attempt to find suitable solutions to create control software for individual robots so that they achieve a particular collective behavior. One solution is automatic design, in which control software is generated automatically by an optimization process that configures a certain control software architecture. Traditionally, automatic design is accomplished through neuroevolution, that is, training a neural network in a simulated environment using an evolutionary algorithm, directly on the mission to tackle. Neuroevolutionary approaches have shown to be very effective, and work quite well when control software is designed and assessed in a simulated environment. However, they have been shown to be susceptible to the reality gap, the intrinsic and unavoidable difference between reality and the simulation environment used to generate the control software.

In recent years, another approach has been proposed: modular design. In modular design, the control software is composed of modules that can be created once and for all and re-used for multiple missions. Restricting the space of possible behaviors, by limiting the control software to a combination of existing modules, effectively increases the robustness to the reality gap of modular methods in comparison to neuroevolutionary ones (Francesca et al. 2014b). However, modular methods require human expertise in swarm robotics in order to create the modules, making them more challenging to implement than neuroevolutionary methods.

The objective of this thesis is to combine the advantages of neuroevolution and modular methods: the ease of implementation of neuroevolution, which requires minimal human expertise and domain knowledge, and the robustness to the reality gap of modular methods. Is it possible to create an automatic design method that is both as robust as a hand-crafted modular method, and as versatile as a purely neuroevolutionary method ?

In this thesis, we present our experimental work on the robustness of automatic design methods to the reality gap, in a thorough comparison of the most classical neuroevolutionary methods and the most classical modular ones. We show that the presented neuroevolutionary approaches are all greatly affected by the reality gap and outperformed by the modular method. Then, in the form of two methods, `Arlequin` and `Nata`, we present our advances in modular automatic design. These methods use neural networks as modules of the control software architecture and require less human expertise than their predecessors. We show that they outperform the neuroevolutionary method, and are able to generate control software that crosses the reality gap relatively better than neuroevolutionary methods.

Contributions

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

Critical review of the state of the art in swarm robotics: We provide a critical review of the state of the art in automatic design of robot swarms focusing on offline methods and with a special attention to modular methods.

Novel categorization of automatic design methods: We propose a novel categorization of automatic design methods, to disambiguate between semi- and fully-automatic design methods.

Comparison of automatic design methods: We present the most extensive comparative analysis of offline automatic design methods currently available in the literature, supported by simulations and experiments with physical robots.

`Arlequin`: We introduce `Arlequin`, a novel modular automatic design method that uses pre-trained neural networks as modules of a probabilistic finite-state machine.

`Nata`: We introduce `Nata`, a novel modular automatic design method that uses repertoires of behaviors and principled design to automatically generate the modules, with the aim to reduce expert knowledge required in the implementation of automatic design methods for robot swarms.

Robot experiments: We present empirical analyses with experiments conducted on physical robots. The results discussed in this thesis are based on more than 550 runs for Chapter 4, 60 for Chapter 5, and 120 for Chapter 6, for a total of more than 730 experimental runs.

Hardware and software implementations: We implemented and maintain the software for the methods presented in the thesis, notably `Arlequin`, `Nata`, `EvoStick`, `NEAT`, `xNES`, and `CMA-ES` for the ARGoS3 simulator and on the e-puck robots. The software is available as open source software on the public repository of the DEMIURGE project.

Statement

This thesis presents an original work that has never been submitted to the Université libre de Bruxelles or to 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.

Part of the state of the art in 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.
- 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.

The experiments presented in Chapter 4 are based on:

- **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.

The definition and the analysis of the automatic design method `Arlequin` presented in Chapter 5 are based on:

- Ligot, A., **Hasselmann, K.**, & Birattari, M. (2020). AutoMoDe-Arlequin: Neural networks as behavioral modules for the automatic design of probabilistic finite state machines. *Swarm intelligence: 12th international conference, ANTS 2020* (pp. 109–122). Springer.

The definition and the analysis of the automatic design method `Nata` presented in Chapter 6 are based on:

- **Hasselmann, K.**, Ligot, A., & Birattari, M. (2022). Automatic modular design of robots swarms based on repertoires of behaviors generated via novelty search. *Swarm and Evolutionary Computation*, *(under review)*.

The author also contributed to research projects not presented in this thesis, which he, together with co-authors, has published in the scientific literature:

- **Hasselmann, K.**, Robert, F., & Birattari, M. (2018). Automatic design of communication-based behaviors for robot swarms. *Swarm intelligence: 11th international conference, ANTS 2018* (pp. 16–29). Springer.

- **Hasselmann, K.**, & Birattari, M. (2020). Modular automatic design of collective behaviors for robots endowed with local communication capabilities. *PeerJ Computer Science*, *6*, e291.

- Salman, M., Garzón Ramos, D., **Hasselmann, K.**, & Birattari, M. (2020). Phormica: Photochromic pheromone release and detection system for stigmergic coordination in robot swarms. *Frontiers in Robotics and AI*, *7*, 195.

- Garzón Ramos, D., Bozhinoski, D., Francesca, G., Garattoni, L., **Hasselmann, K.**, Kegeleirs, M., Kuckling, J., Ligot, A., Mendiburu, F. J., Pagnozzi, F., Salman, M., Stützle, T., & Birattari, M. (2021). The automatic off-line design of robot swarms: Recent advances and perspectives. *R2t2: Robotics research for tomorrow's technology.*

Implementations of software used throughout the work are released as open source software and are described in:

- 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.

- **Hasselmann, K.**, Ligot, A., Francesca, G., Garzón Ramos, D., Salman, M., Kuckling, J., Mendiburu, F. J., & Birattari, M. (2018). *Reference models for AutoMoDe* (tech. rep. TR/IRIDIA/2018-002). IRIDIA, Université Libre de Bruxelles.

- Kuckling, J., **Hasselmann, K.**, van Pelt, V., Kiere, C., & Birattari, M. (2021). *AutoMoDe Editor: A visualization tool for AutoMoDe* (tech. rep. TR/IRIDIA/2021-009). IRIDIA, Université Libre de Bruxelles.

Acknowledgements

I acknowledge support from the DEMIURGE project, funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 681872). Illustrations presented at the beginning of chapters were created using MidJourney a machine learning model that creates images from textual descriptions.

It was a long process for what has been the hardest and longest project of my life.

I want to express a special thanks to Prof. Mauro Birattari. I feel privileged to have had the opportunity to work with such a brilliant and supportive mentor. We always had awesome conversations, sometimes lasting for hours about various topics, from electronics and science, to jiu-jitsu or cooking. This has helped me grow as a researcher and engineer but also as a person.

A big thank you to the members of my jury; their expertise and constructive feedback have been instrumental in helping me to improve this thesis.

I also wish to thank the other professors and researchers at IRIDIA for the stimulating discussions and for their friendship, this lab is a big family, and I feel really grateful to have been able to join a lab with such diverse, interesting, and clever people: Hugues, Marco, Thomas, Leslie, Roman, DJ, Fed, Alberto, Giovanni, Marcolino, Volker, Guillaume, Garazi, Alex, Edu, MK, and all researchers and students whom I forgot to mention!

Of course, one cannot mention IRIDIA without mentioning all the wonderful researchers and friends that worked on the DEMIURGE project. A special thanks to Antoine, my closest workmate and dear friend. We have gone through this together; I will always remember the long robot experiment sessions that we did. I am not sure I would have finished those experiments or the PhD if you were not there, for the fun, the entertainment, and the occasional mandatory seriousness. Thank you also to the whole DEMIURGE team: Gianpiero, Lorenzo, Salman, David, Jonas, Fernando, Miquel, Guillermo, and to all other researchers and students that participated in this project. From the beginning of the project when we were only a few, to the large and great team of researchers ready to take over the world and convince everyone that automatic design is the way!

Before and during my time at IRIDIA, I was also part of the BEAMS lab. I wish to thank Prof. Dragomir Milojevic and Prof. Frédéric Robert for bringing me to Belgium. I could not have joined ULB if you did not trust me with your courses. I also met awesome people and friends while teaching electronics and computer science. To all researchers and staff members, Anh Vu, Yannick, Raoul, Quentin, Axel, Michel, François, Sami, Youssef, and to students that I met while teaching at BEAMS, thanks!

I recently joined a new institution, I wish to also thank all my new coworkers at ECAM for welcoming me, with a special mention for the computer science and electronics department, such a great team!

To all my friends from Toulouse, net7, and 7robot: Nim, Delphine, Palkeo, djanos, Dib, Joffrey, Adri, and to all the friends I made in Belgium, a very welcoming country: Antoun, Anto, Thanh, Anaïs, Albert, Laurène, Quentin, Erica, Mat, Lou, John, Soph, the whole club of la Fine Plume, and all the others who crossed my path in the last 8 years. I am grateful for the nights, the beers, and the moments of laughter that helped me through the challenges of this process. Thanks also to all other friends and family friends, the Leclerc family, Pierre, Jutta and all the ones that I forgot to mention!

To my parents-in-law, Pascale and Michel, for your constant support, the fun, and the family time, I wish Michel would have been able to read those lines, words are missing to describe how much we miss you.

To my parents, Bernard and Danièle, for your love, support, and encouragement throughout my life. Your sacrifices and constant belief in me have been the foundation of my journey and have allowed me to reach this point.

Finally, the last words are for you, Florence, you have supported me over hills and valleys, in difficult times and happy moments. Contrary to me, you never doubted that I would finish this work. Now is a new beginning, a version of myself that does not have a PhD to do, something new to you. I am very lucky to have you in my life. I love you.

Contents

List of Figures

1. Introduction

Robots and automatons are part of our daily lives. They may appear to be a recent development, yet the concepts of robots and artificial life date back to antiquity. The ancient Greek myth of *Talos* contains the earliest known reference to artificial life. This giant bronze warrior, created by the Greek god *Hephaestus*, was tasked to patrol and defend the island of Crete from invaders (Mayor 2018). This myth, rather than having to do with technology, tells the tale of human's desire to engineer artificial life, to augment, surpass or simply imitate nature in order to free themself from the most tedious tasks.

In history, the earliest signs of automatons start from mechanical devices designed to animate statues in ancient times, and Leonardo da Vinci's descriptions and rudimentary builds of mechanical humanoid machines in the 1500s. It was only in 1961 that the first industrial robot was introduced. The company Unimation designed the PUMA (Programmable Universal Manipulator Arm) robot arm. As of today, scientific and technological advancements feed science-fiction stories and science-fiction stories, in turn, nourish our collective imagination. In science fiction, robots are frequently depicted as sentient humanoids, whether it is C-3PO in Star Wars, the Terminator, or Bender from Futurama, they behave and make decisions similarly to humans. Nevertheless, as demonstrated by the PUMA robot, the first robots were industrial robots, primarily robotic arms, mostly remotely operated or with pre-programmed kinematics. In our collective imagination, however, robots are often characterized by their cognitive abilities such as how they interact with humans, or their ability to make decisions. They are not merely remote-operated machines, but are endowed with some sort of artificial intelligence. The fields of robotics and artificial intelligence are tightly connected, as we want both intelligent and physically capable machines to help us. The challenge is not only to create clever mechanical structures, but also to give them the ability to achieve their intended purpose. Advances in artificial intelligence and robotics are leading to new applications, with different levels of automation and autonomy: from the Mars rover Curiosity, wandering the red planet, to your own personal dust cleaning or lawnmower robot.

However, some tasks are difficult for a single robot to accomplish, as it can become lost, or damaged. A single and complex robot might not be the ultimate solution for complex missions requiring, for instance, the parallel execution of multiple tasks over a relatively extended area. Certain missions are best accomplished by teams or groups of robots. This is how, inspired by ants and other social animals, the first robot swarms were imagined (Dorigo et al. 2014).

Swarm robotics is the study of the design and deployment of large, cooperative, self-organized groups of robots. It began as an application of swarm intelligence, the study of self-organized systems and the modeling of collective behaviors, but rapidly developed into an engineering discipline on its own (Brambilla et al. 2013). Swarm robotics systems should produce complex emergent behaviors from simple robot-level rules. A well-designed swarm must be adaptive, robust, and scalable (Dorigo et al. 2021). Adaptability is the capability of the swarm to adapt to changes in its environment or mission; robustness is the capability of the swarm to be robust to robot failures; scalability is the capability of the swarm to accommodate to changes in terms of number of robots and size of problems.

Recent research has demonstrated promising results in a variety of applications including task sequencing (Garattoni and Birattari 2018), re-configurable swarms (Xie et al. 2019), collective locomotion (Li et al. 2019), underwater swarm coordination (Berlinger et al. 2021), robot self-construction (Boudet et al. 2021), and aerial swarm navigation (Xin et al. 2022). Notwithstanding these promising results and all the research conducted over the past two decades, swarm robotics is a field that is still in its infancy; it has yet to find its way out of labs and into real-world applications.

Swarms robotic systems are advantageous with respect to centralized multi- or single-robot systems in adaptability, robustness, and scalability. The main challenge resides in their design (Dorigo et al. 2021; Schranz et al. 2021). Typically, the missions to be tackled by the swarm are defined at the swarm level; they describe the emergent swarm behavior—what the entire swarm should achieve. As the system should not have a central authority, and as interactions in the swarm should be local, the designer of the swarm usually operates at the individual-robot level. The design problem stems from the fact that it is hard to predict the emergent swarm behavior from the individual behavior of the robots in the swarm. Some principled design methods have been proposed to address this issue (Berman et al. 2011; Brambilla et al. 2014; Kazadi 2009; Lopes et al. 2014; Pinciroli and Beltrame 2016; Reina et al. 2015a; Spears et al. 2004). However, they are constrained to specific classes of missions: for the moment, no general engineering framework is available and manual *trial-and-error* continues to be the most prevalent design method.

Optimization-based design, or automatic design, is one of the promising solutions to this problem (Dorigo et al. 2021; Francesca and Birattari 2016). In automatic design, the design problem is cast into an optimization problem. The design parameters of the swarm, that is, the control software of the robots, and possibly some hardware characteristics, define the search space to be explored by an optimization algorithm. The optimization algorithm then maximises a mission-specific objective function, defined at the collective level, which measures the performance of the swarm. Automatic design methods can be categorized using two (orthogonal) categorization systems (Birattari et al. 2020). A method can be classified as *online* or *offline*, and as *semi-automatic* or *(fully-)automatic*.

The first categorization is defined as follows: in online methods, the optimization of the control software happens while the robots operate in the target environment. The swarm is directly used and designed in its final production environment, without any prior design phase. In offline methods, the optimization of the control software happens before the deployment of the robots, usually using computer simulations. This categorization is, of course, not strict and rigid; hybrid methods exist.

The second categorization classifies design methods as either semi-automatic or (fully-)automatic. In semi-automatic design, a human designer operates the optimization and iterates in a "human-in-the-loop" approach. Typically, in an offline semi-automatic design method, the designer would use their experience and knowledge to guide the optimization software by adjusting design parameters to achieve the best possible final design. In automatic design, the design parameters are predetermined, iterations are not permitted—there should be no human intervention in the design process. Typically, an automatic design method is expected to work on a class of missions—characterized by constraints, mission-types, operation environment—without requiring any manually applied mission-specific modifications. One can imagine any combination of these categories, the offline/online and semi-automatic/(fully-)automatic categorizations are orthogonal and describe different characteristics of a design method. Offline semi-automatic design is currently the most researched category in the scientific literature, followed by online semi-automatic and offline automatic design.

In this thesis, we focus on offline (fully-)automatic design, not because we believe that this category is superior to the others, but because we feel that it is a promising design approach that deserves to get more attention than it currently has.

Indeed, the core challenges faced by the semi- and fully-automatic approaches differ: in semi-automatic design, the focus is on the complexity of the specific mission and the role of the human; whereas in automatic design, the focus is on the complexity of the class of mission, the diversity of missions that can be tackled.

A standard approach to the offline automatic design of robot swarms is neuroevolution (Trianni 2008). In this approach, one uses neural networks as the control software of the robots, and one designs their parameters, and possibly their topology, using an evolutionary algorithm. It is the most adopted approach in offline automatic and semi-automatic design of robot swarms. Its advantages include: the ease of implementation and utilization, and its high portability. Once the robotic platform is selected, one only has to map the robot's sensors and actuators to inputs and outputs of the neural network. The main drawback of the neuroevolutionary approach is its sensitivity to the reality gap. The reality gap is the intrinsic and unavoidable difference between the training and deployment environments, typically, simulation and reality (Jakobi 1997). Since neuroevolutionary methods are known to be quite sensitive to the reality gap (Francesca et al. 2014b), they perform poorly when control software generated in simulation is deployed to physical robots. Efforts have been made to mitigate the effects of the reality gap (Bongard and Lipson 2004; Jakobi 1997; Koos et al. 2013; Miglino et al. 1995), but no definitive solution has emerged, yet (Ligot and Birattari 2020; Silva et al. 2016).

Francesca et al. (2014b) conjectured that the reality gap problem resembles the generalization problem of machine learning. During simulation, the control software *overfits* the characteristics of the simulated environment, leading to a performance drop once it is ported to real robots. In accordance with the *bias-variance trade-off*, it is possible to decompose the generalization error of a learning algorithm into two terms: bias and variance (Geman et al. 1992; Wolpert 1997). Bias and variance are known to be correlated with the complexity of learning algorithms: typically, high-complexity algorithms have high variance and low bias, whereas low-complexity ones have low variance and high bias. The high representational power of neuroevolutionary approach, and their high complexity, is conjectured to be responsible for the *overfitting* of such algorithms and, consequently, their sensitivity to the reality gap (Floreano et al. 2008) (details on the use of *overfitting* in this context can be found in Section 2.3.4). With the intention of reducing the impact of the reality gap, Francesca et al. (2014b) created AutoMoDe. In AutoMoDe, injecting bias, in the form of expert knowledge in hand-crafted and pre-defined software modules, lowers the complexity of the control software.

The disadvantage of this approach it that designing such methods requires specialized knowledge. The modules must be created and tested by an expert in swarm robotics after selecting the hardware platform.

Keeping this in mind, we highlight some current issues in the literature. According to Birattari et al. (2020), the first issue we identified is the lack of emphasis on the empirical analysis and comparison of methods: in the field of automatic design, few studies were devoted to assessing solutions with physical robots. We acknowledge that this can be costly and time consuming but we are convinced that the state of the art will not be defined until systematic rigorous tests and comparisons are conducted as part of the methodology for creating new design methods.

It is our contention that, as the reality gap problem is central to assessing the quality of a design method, the scarcity of empirical studies hinders the development of the field.

As a first contribution of this thesis, we propose a comparison of existing design methods with extensive experimental work. This comparison highlights the significant effect of the reality gap on various design methods. All subsequent presentations and analyses of design methods proposed in this thesis include empirical assessments with physical robots. As a second contribution, we corroborate the conjecture on the bias/variance tradeoff and the principles of modularity. For this, we propose two design methods whose modules are not hand-crafted but automatically generated. We compare them to other existing design methods. As a last issue, we believe that popular modular design methods could benefit from reducing the amount of expert knowledge required to implement them. We propose two methods that aim to combine the portability of neuroevolutionary methods with the robustness of modular ones, in an effort to reduce the need for expert knowledge in swarm robotics in the design of modular methods.

Both methods use neural networks as modules. In the first one, we train behavioral modules in the form of neural networks to desired behaviors by defining specific objective functions, once and for all, before combining those modules into probabilistic finite-state machines.

In the second one, we use a quality-diversity algorithm (Cully and Demiris 2018) to create a repertoire of behaviors, and rule-based design to create the linking conditions before combining them, also, into probabilistic finite-state machines. By rule-based, we mean that we defined rules which are derived, based on the reference model of the robot in use, into conditional modules.

This thesis has the following structure: in Chapter 2, we propose an overview of the current swarm robotics literature, beginning with general considerations before delving deeper into automatic design of robot swarms and the specific previous research that is relevant to this thesis. We first discuss the different categorization systems we have presented, namely offline, online, semi-automatic and automatic design, and propose a description and a classification of the most relevant research papers of the field. We present the challenges faced by these methods and introduce modular automatic design methods and the relevant literature.

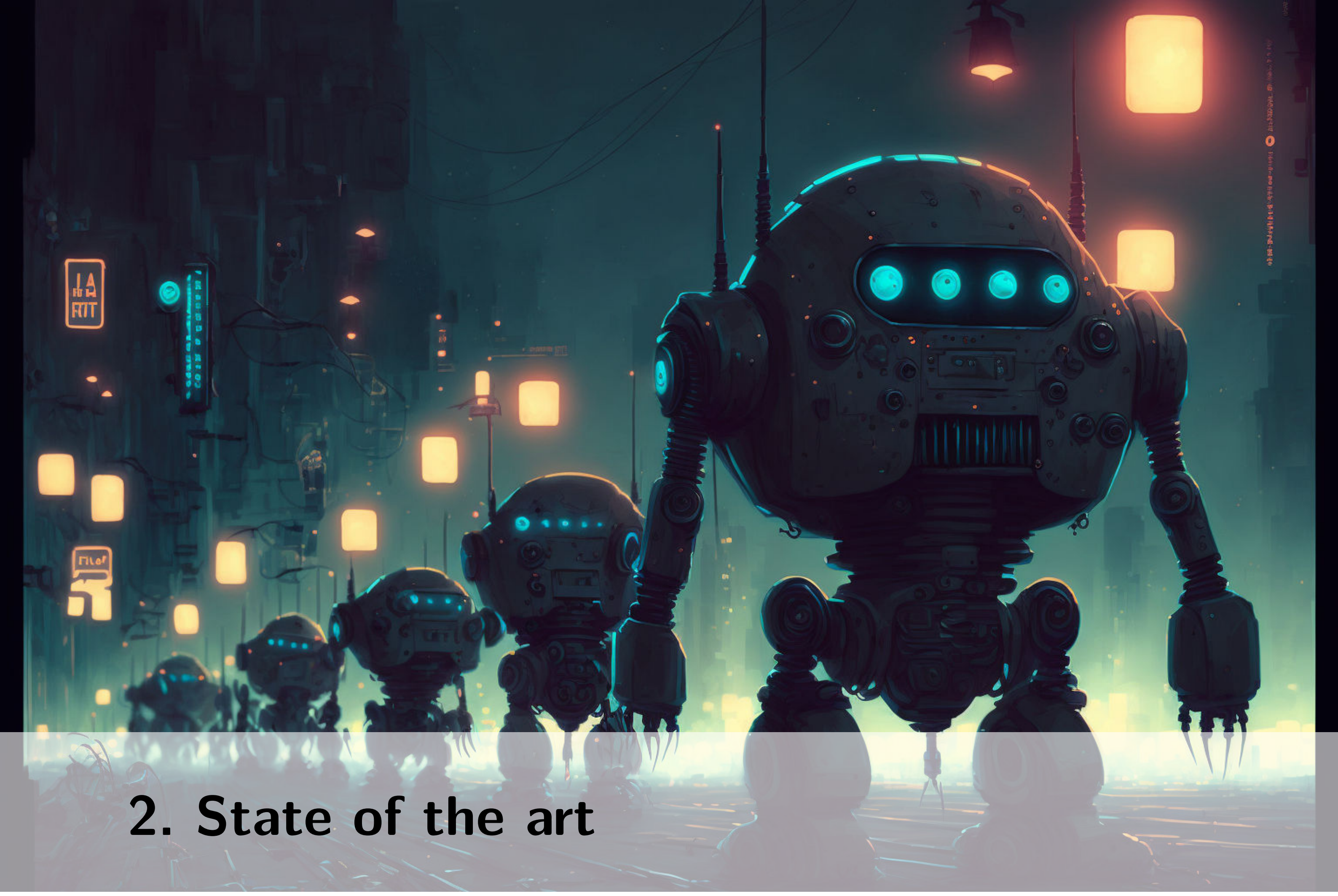
In Chapter 3, we present the materials and methods that are common to all experiments and analyses presented in this thesis.

In Chapter 4, we present our work of assessing the most popular automatic design methods in the literature. On five distinct classical swarm robotics missions, we assess nine automatic design methods, and a random walk as a yardstick. We test the designed control software both in a simulation environment and in reality. The results show that neuroevolutionary methods all suffer greatly from the reality gap and therefore that real-robot experiments are essential to assess the performance and quality of design methods. The modular method suffered the least from the reality gap, thus leading us to explore the possibility to combine the ease of use of neuroevolutionary methods and the robustness of modular ones.

In Chapter 5, we present our first attempt to reduce the need for human intervention in the implementation of a modular design method. We propose an automatic design method that combines neural networks (as behavior modules) into probabilistic finite-state machines. In `Arlequin`, the neural network modules are generated once and for all, for a specific behavior, defined by an explicit objective function, in a mission-agnostic way. The results show that `Arlequin` is capable of outperforming the classical neuroevolutionary approach. This suggests that using neural networks as modules is a promising approach to the automatic design of robot swarms.

In Chapter 6, we present our second attempt to reduce even further the human intervention needed in the implementation of a design method by proposing `Nata`. `Nata` uses a quality-diversity algorithm (Cully and Demiris 2018) to automatically generate behavior modules in the form of neural networks, and rule-based design to automatically generate conditional modules to be combined in probabilistic finite-state machines. The results show that `Nata` outperforms both `Arlequin` and the classical neuroevolutionary approach, while being the method that requires the least human knowledge and intervention to be implemented.

In Chapter 7, we reflect and conclude on the topic of this thesis. We also, present potential follow-up research directions.



2. State of the art

This chapter provides an overview of the literature on swarm robotics, focusing on automatic design methods for robot swarms. As described in the previous chapter, one of the main challenges in swarm robotics is the design problem. This problem stems from our inability to predict the collective behavior of a swarm based on the behaviors of individual robots. This makes designing robot swarms a challenging problem. When designing robot swarms, one must operate at the individual-robot level, that is, defining the behavior of each independent robot. However, the behavior of the swarm is best described at the collective level, that is, as seen as a single compound entity.

One possible and promising approach to the design of robot swarms is automatic design, where the design problem is cast into an optimization problem. Under this approach, the characteristics of the swarm, the control software of the robots, and possibly the specification of their hardware components, are generated by maximizing a given objective function, which describes the quality of the solution at the collective level. This approach, classified as machine learning for robot swarms by Dorigo et al. (2021), is deemed to be a promising research direction to overcome the unpredictable contingencies that characterize swarm robotics systems (Dorigo et al. 2021).

In this thesis, we explore new methods for the automatic design of robot swarms; therefore, for the purpose of the state of the art presented in this chapter, we will focus on automatic design methods.

For an in-depth introduction to the field of swarm robotics, we refer the reader to Hamann (2018) and Garattoni and Birattari (2016); for a structured presentation of the engineering approach to designing robot swarms, we refer to Brambilla et al. (2013); for a description of current applications of robot swarms, we refer to Schranz et al. (2020); for a reflection on the present and future of swarm robotics, we refer to Dorigo et al. (2020, 2021).

2.1 Swarm robotics

Swarm robotics is the study of swarm intelligence applied to groups of robots, as well as of how to operate and design them (Dorigo et al. 2014). Individually, robots in a swarm have limited capabilities, but their cooperation results in the emergence of complex behaviors. Collectively, the swarm can perform missions that no single robot could perform on its own. Swarm robotics was historically seen as a way to model biological systems, such as social animals (Beni 2005). The field has since evolved into an engineering discipline whose objective is to create swarm robotics systems that can solve real-world problems (Brambilla et al. 2013).

The main properties of a robot swarm are: (i) fault tolerance, (ii) scalability, and (iii) flexibility (Dorigo et al. 2014). Fault tolerance is achieved by introducing redundancy and decentralization in the swarm. This implies that the loss of an individual robot should not harm the whole swarm. In other words, there should be no single point of failure. In addition, the system must also be scalable: it should be able to adapt to changing problem dimensions without jeopardizing performance. This is achieved by relying solely on local sensing and communication between robots; robots in the system rely only on their neighboring environment. Finally, the system must be flexible; it should adapt to different environments and working conditions. This property is also enabled by the decentralized nature of the swarm and the limited local sensing.

Over the past two decades, swarm robotics has allowed significant advancements in our understanding of natural and artificial swarms. Due to these characteristics and despite the fact that real-world applications are still limited, a vast number of future potential engineering applications are anticipated (Dorigo et al. 2020). For instance, a robot swarm could be particularly effective in a situation that requires rapid exploration of a large geographical area. In agriculture, swarms of aerial robots could be utilized to apply fertilizer or eliminate weeds, enabling precision agriculture and potentially reducing the amount of chemicals required (Schranz et al. 2020). Robot swarms could also be utilized for planetary exploration, search-and-rescue, cleaning dangerous waste, and generally in large hostile environments. For a comprehensive look at current applications of robot swarms, we refer the reader to Schranz et al. (2020).

Even though the field of swarm robotics has risen to a prominent position in the scientific literature (Berlinger et al. 2021; Boudet et al. 2021; Garattoni and Birattari 2018; Li et al. 2019; Rubenstein et al. 2014; Slavkov et al. 2018; Werfel et al. 2014; Xie et al. 2019; Xin et al. 2022; Yu et al. 2018), it still suffers from the lack of methodological and engineering principles for designing collective behaviors (Brambilla et al. 2013). The design of these systems remains a challenging endeavor.

2.2 The design of robot swarms

The properties and characteristics of swarms, such as decentralization, and limited sensing, make robot swarms loosely coupled systems. There is no global centralized system that controls or monitors the whole swarm, the collective behavior of the swarm emerges from the local interactions between the robots, and between the robots and their environment. These interactions between the large numbers of robots and environmental features make robot swarms complex systems. The behavior of these systems is best described at the collective level, but their decentralized nature forces the behaviors of the robots to be defined at the individual level. This makes the design of swarm robotics system particularly challenging.

The most prevalent method for designing swarm robotics systems is manual design. In manual design, an expert designs the behavior of the individual robots. For this, no general engineering framework is available, yet, although a few principled design methods have been proposed (Beal et al. 2012; Berman et al. 2011; Brambilla et al. 2014; Hamann 2018; Kazadi 2009; Lopes et al. 2014, 2016; Pinciroli and Beltrame 2016; Reina et al. 2015a,b; Spears et al. 2004). These methods, however, are constrained to specific classes of missions. Therefore, trial-and-error is still the most common way of designing robot swarms today. In this method, the designer works by iteratively testing its design on the swarm of robots, either in a simulated environment or with real robots, until the desired swarm behavior is achieved (Garattoni and Birattari 2016).

A promising approach to the design of robot swarms is automatic design (Francesca and Birattari 2016). In automatic design, the design problem is cast into an optimization problem. The optimization algorithm searches the space of possible instances of control software for the robots, with the aim to maximize or minimize a given metric. This metric is usually referred to as the *objective function*—a metric indicating the performance of the swarm in a given mission.

2.3 Automatic design

In this section, we discuss automatic design methods and disambiguate between offline, online, semi-automatic and fully-automatic design methods. It is important to note that the distinction we make between automatic design and manual design is not to be taken as a rigid categorization, but rather as a general way of thinking about different methods; hybrid methods are possible.

In automatic design, as described in the previous section, the design problem is cast into an optimization problem. The optimization algorithm searches in the space of possible solutions to maximize (or minimize) an objective function. The objective function is an indicator of the quality (or cost) of a given instance of control software at the swarm level; a measure of the performance (or inability) of the swarm to perform a mission. The objective function thus contains expert knowledge on how to measure the success of a mission.

The most prevalent method for automatically designing robot swarms is neuroevolution (Nolfi and Floreano 2000), in which the actuators and sensors of each robot are mapped to the input and output neurons of a neural network (Trianni 2008). In neuroevolution, the parameters of the neural network, such as the synaptic weights, and the topology of the network are optimized using an evolutionary algorithm that maximizes the objective function. Automatic design can be analysed based on two distinct categorizations: online or offline design, and semi-automatic or automatic design.

2.3.1 Online and offline design

In online design, the control software is designed while the swarm operates in the target environment. The optimization of the parameters of the control software is distributed among the robots and occurs in real time. This approach allows the swarm to adapt to changing features in the environment by updating its behavior in real-time.

Nonetheless, this approach also has some drawbacks. First, the limited time available for the robot to adapt makes it impossible to explore large search spaces. In this regard, online adaptation generally works best with fewer parameters than other approaches on relatively small search spaces (Bredeche et al. 2018). Second, the capabilities of the swarm and the class of missions it can perform is limited by the objective function that can be computed locally by the robot. Indeed, in online design, robots must be able to individually compute the value of the objective function to optimize the parameters of the control software. Finally, when executed on real robots, control software with sub-optimal parameters, especially during the early stages of the design phase, may contain behaviors that could be dangerous for the robots, and damage them (Francesca and Birattari 2016).

By contrast, in offline design, the control software is designed before it is deployed in the target environment. The optimization of the parameters of the control software can be done in a simulated environment or on real robots, using a replica of the final target system. In most cases, however, the design phase takes place in a simulated environment. This allows for a broader class of missions to be performed as the simulation offers a global perspective over the entire swarm system. Objective functions whose computation requires a holistic view over the swarm can be used, and are not restricted to what an individual robot can compute. Simulations are also usually less expensive and faster than real-robot experiments, allowing more computing time to be allocated to the design phase. Instances of control software with sub-optimal parameters do not put the robots and the environment at risk as they are usually eliminated in simulation during the design phase.

However, offline design methods are known to suffer from the reality gap, that is, the unavoidable difference between simulation and reality. Indeed, offline methods tend to overfit the characteristics of the simulated environment used to design them and might transfer poorly to the real world (see Section 2.3.3 for more details on the reality gap).

2.3.2 Semi-automatic and fully-automatic design

Another way to classify automatic design methods is to categorize them as semi-automatic or (fully-)automatic design. This categorization is orthogonal to the online and offline categorization, which means that any automatic design method

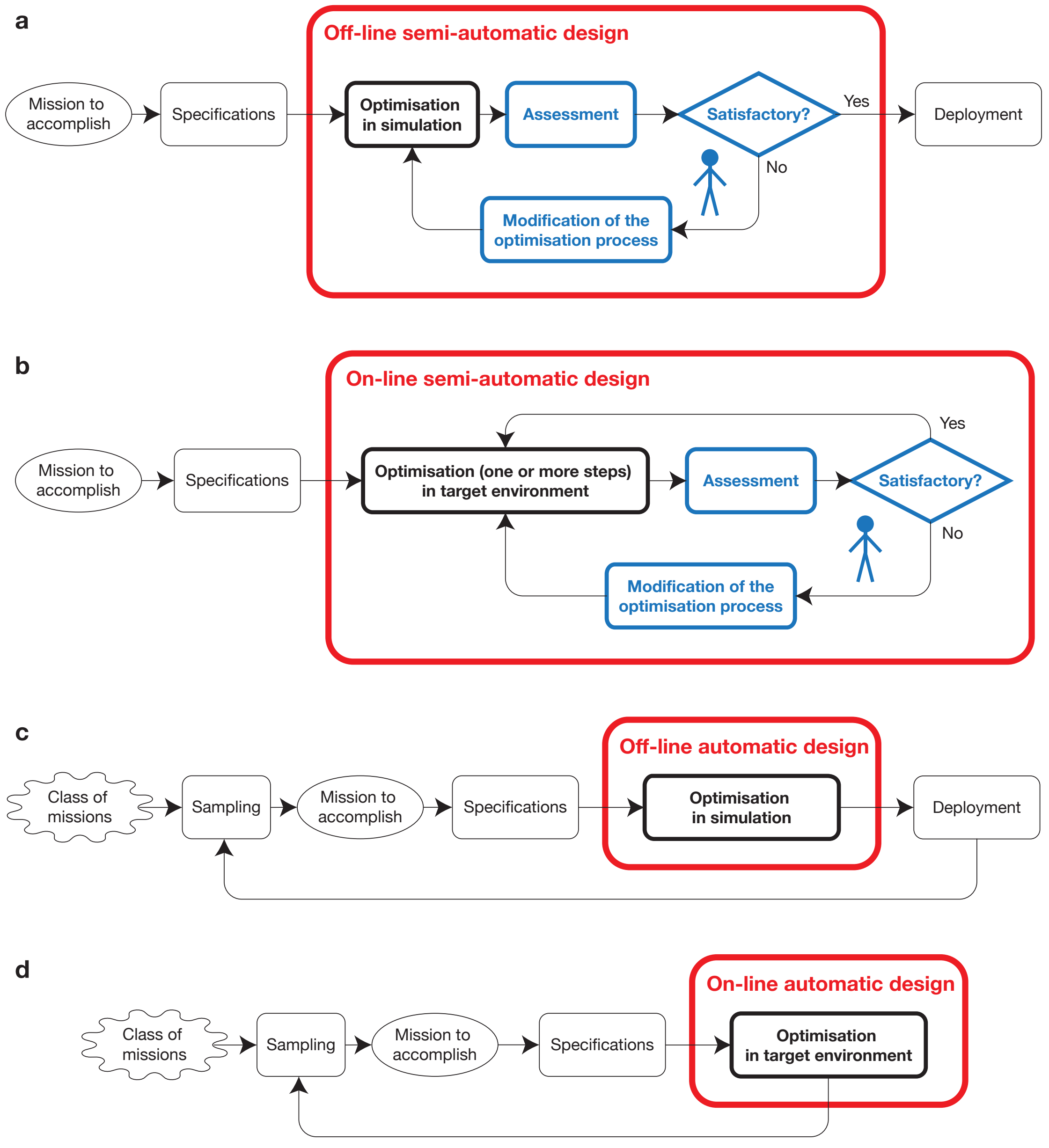

a
Off-line semi-automatic design
Mission to accomplish
Specifications
Optimisation in simulation
Assessment
Satisfactory?
Yes
No
Deployment
Modification of the optimisation process
b
On-line semi-automatic design
Mission to accomplish
Specifications
Optimisation (one or more steps) in target environment
Assessment
Satisfactory?
Yes
No
Modification of the optimisation process
c
Off-line automatic design
Class of missions
Sampling
Mission to accomplish
Specifications
Optimisation in simulation
Deployment
d
On-line automatic design
Class of missions
Sampling
Mission to accomplish
Specifications
Optimisation in target environment

Figure first published in Birattari et al. (2020)

Figure 2.1: **Flowcharts of typical workflows for different approaches of automatic design.** **a**, offline semi-automatic design. **b**, online semi-automatic design. **c**, offline automatic design. **d**, online automatic design. In the flowcharts, the design process is contained in the red box. Outside the red box are represented other parts of the life-cycle of an automatic method, like specifications or deployment. In semi-automatic design, the human is inside the loop (the red box), whereas in automatic design the human does not intervene.

can be classified as either online or offline design (or any hybrid) as well as automatic or semi-automatic, at the same time.

Consequently, four cross-categories can be identified, online semi-automatic, online automatic, offline semi-automatic, and offline automatic design. In this section, we analyze the distinction between automatic and semi-automatic methods.

In semi-automatic design, a human designer operates the optimization algorithm. Therefore, human intervention is part of the design process. This process is iterative and the optimization algorithm is used as a tool. In other words, the search for a suitable solution—a control software instance—is guided by the experience and knowledge of the designer. Typically, the designer would make an initial attempt to find an instance of control software, evaluate it in the target environment, observe the results, and adjust the design method's parameters to steer the process and achieve better results.

The designer can modify any parameter of the design method, such as the sensor filtering, the control software architecture, the neural network topology, the simulation model, the optimization algorithm, and the objective function used to measure the performance of the swarm. Once updated, the designer can observe the impact of their modifications in the target environment by reevaluating the performance of the swarm, and iterating, if needed, at will. Iterations are repeated until the swarm demonstrates a satisfactory behavior.

Both online and offline semi-automatic design involve a human in the loop. The difference is that in the online case, the performance of the swarm is evaluated directly in the target environment, whereas in the offline case, the performance is evaluated, first in a simulation environment, and once the desired performance is achieved, in the target environment by using real robots. A flowchart depicting the two cases is presented in Figure 2.1. The degree of human intervention can vary from method to method, from fine-tuning the optimization algorithm to maximize performance on a mission, to hand-crafting robotic behaviors used as high-level functions tailored to a specific mission. Despite that, we draw a clear line of separation between categories: whenever humans are in the design loop, regardless on how much they intervene, we categorize it as semi-automatic design.

As a particular example, one of the most classical ways of designing robot swarms is neuroevolution. In fact, a large part of neuroevolutionary design methods belong to offline semi-automatic design. In the literature, few studies explain the extent of the required human intervention. It is out of contention that, in order to improve the reproducibility of studies on design methods for robot swarms, it should be clear how much intervention and how many iterations were needed to obtain specific results.

In contrast to the semi-automatic approach, in (fully-)automatic design, no human intervention is allowed in the design process. The mission to be performed by the swarm is formally specified by defining an objective function. Then, the (fully-)automatic design process uses this mission specification to explore the space of solutions for a suitable instance of control software. Finally, the selected instance is deployed in the target environment.

The automatic design method is predetermined for a whole class of missions. In this regard, a class of missions is defined by the capabilities of the robots, the environmental cues and the type of mission. After that, the method does not require any modification to be used with different missions within this class. There is no human intervention after the constraints for the automatic design method have been set, neither per-mission nor during the design phase for a given mission. This is true for both offline and online (fully-)automatic design. The difference between the two cases is that, in offline design, the design phase is carried out using simulations, whereas, in online design the design phase is carried out directly in the target environment. A flowchart depicting the two cases is presented in Figure 2.1.

In the literature, few studies fall into the category of automatic design methods; the lack of details regarding the nature of the optimization process makes it difficult to categorize them definitively as either semi- or (fully-)automatic. Still, the challenges faced by semi-automatic and automatic methods are different.

On the one hand, semi-automatic design is employed when designing control software for a single, complex mission, that would be too difficult or too time-consuming for a human to design. In this case, it would be counterproductive to develop a fully-automatic design method, since they are more complex to develop and tend to work on relatively simpler missions. The semi-automatic design approach is used, here, as a tool to assist the designer in completing a single design task for a single mission, not a class of missions. The designer includes their expert knowledge to bias the creation of the automatic design method towards the specific mission at hand. On the other hand, (fully-)automatic design is used when designing control software for a class of missions. It is especially useful when it would be unfeasible for economical or practical reasons that a human designer supervises a design process that has to be executed repeatedly. Within the class of mission, in fully-automatic design, there is no need for any modification in the method before design and deployment. The challenge in semi-automatic design resides in the complexity of the single mission for which control software must be generated. In fully-automatic design, the challenge resides in the complexity of the class of missions; the diversity of missions within the class. These two categories serve different purposes and ultimately solve different problems. In Figure 2.2 we illustrate, using two examples of potential applications, the different purposes that the two categories serve.

Still, there is a substantial overlap between the fields of semi-automatic and fully-automatic design. Developments in optimization algorithms, software architectures, and simulation models are examples of matters whose development would benefit both fields. However, research questions that are specific to each field remain: in semi-automatic design, they are primarily concerned with the role of the human designer. This role is difficult to measure but important to quantify if we want to make valid comparisons between design methods. By contrast, in fully-automatic design, the primary research question is to characterize the class of missions that can be addressed by a given design method. Both fields are promising areas of research; it is our contention that a crucial step for their development is to establish a proper state of the art, tailored research protocols, and sound evaluation criteria so future research can be built upon. A first step in this direction is presented in Ligot et al. (2022), where the authors present directives for better empirical practice in offline automatic design.

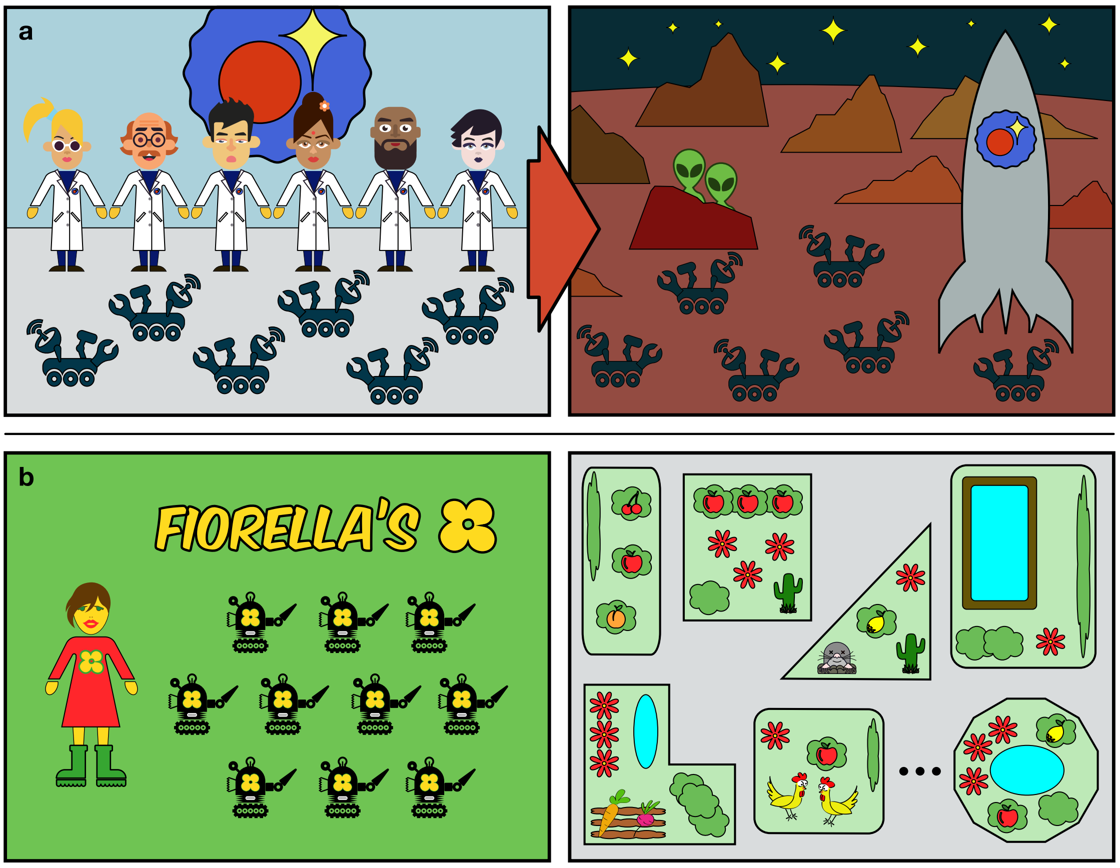
In Section 2.4, we present and classify some of the notable research in the field.

2.3.3 The reality gap

The concept of reality gap is central to the discussion on the design of robot swarms. Crossing the reality gap is one of the main issues faced by any designer when generating robot swarms by relying on simulation (Brooks 1992; Jakobi et al. 1995). The reality gap is the intrinsic and unavoidable difference between simulation and reality. A consequence of the reality gap is an inevitable difference between the behavior of a swarm in the simulated environment and in reality.

Several techniques have been proposed to mitigate this issue at the level of the simulator such as: increasing the realism of the simulator by using actual sampled data from the robot's actuators and sensors (Miglino et al. 1995), adding noise to the simulation model of the robots (Jakobi 1997), or alternating between simulations and real-robots tests while designing control software (Nolfi et al. 1994; Zagal and Ruiz-del-Solar 2007).

As presented, in offline design, the control software is designed in a simulated environment. In Francesca et al. (2014a), the authors argue that the simulated environment can be seen as a training environment and the reality as a test



Design created by Mauro Birattari to illustrate the content of Birattari et al. (2020)

Figure 2.2: **Potential applications of semi-automatic and automatic design.** First, imagine a large international organization deciding to build a swarm based outpost on Mars. The mission to perform is complex and requires a fully-functional fine-tuned swarm, that will fulfill a single mission. Fortunately, a lot of money is devoted to the project and simulations, and exhaustive testing can be done, as well as building an actual mock-up outpost. This situation would perfectly fit the conditions for applying the semi-automatic approach.
For the other situation, imagine a small gardening business owned by a single entrepreneur. The mission to perform is less complex but requires fine-tuning to the specificities of the garden to take care on. For maximum efficiency and under time and money constraints the owner reconfigures its robot swarm before each intervention, and deploys it immediately after, without further tests. This situation fits the conditions for applying the fully-automatic approach.

environment. Therefore, they conjectured that the reality gap problem could be seen, from a machine learning perspective, as a generalization problem.

The generalization error of a learning algorithm can be decomposed into two terms: bias and variance (Geman et al. 1992; Wolpert 1997). Bias and variance are known to be correlated with the complexity of learning algorithms: high-complexity algorithms typically have high variance and low bias, whereas low-complexity ones have low variance and high bias. In other words, high complexity algorithms can learn more complex functions but are more susceptible to overfitting the idiosyncrasies of the training set, whereas low complexity algorithms can learn less complex functions but are less susceptible to overfitting. This is know as the *bias-variance trade-off.*

In the case of offline design, the idiosyncrasies of the training set represent the differences between the simulated environment and the target environment. It follows that, according to the conjecture of Francesca et al. (2014b), high-complexity automatic design methods tend to suffer more from the reality gap than low-complexity ones because they are more likely to overfit the simulator employed during the design phase. The AutoMoDe approach they proposed has a lower complexity with respect to neuroevolution in the sense that the control software it can produce is restricted to what can be obtained by selecting and combining a set of pre-defined modules into a given architecture, and by fine-tuning a small set of parameters—these restrictions are effectively a form of bias introduced to lower the variance.

In Birattari et al. (2016), the authors showed that during design, when overfitting occurs, the difference between the performance of the swarm in simulation and reality increases with the training effort. The authors suggested early stopping as a potential solution to this problem. Following on this idea, in Ligot and Birattari (2018), the authors showed that any environment used to design control software, regardless how realistic, will cause the control software to overfit.

In Garattoni and Birattari (2018), the authors actually observed some kind of “simulation gap”, in which the performance of control software that was hand-crafted using real robots performed worse after being ported back to the simulation environment. In Ligot and Birattari (2018, 2020, 2022), the authors leverage the concept of “pseudo-reality”: the use of two different simulation environments with different characteristics, to be used during the design phase of control software. One to design the control software, and the other acting as a mock-up target environment to test the robustness of the control software to a different environment.

2.3.4 On the use of the term “overfitting”

In machine learning, a learning algorithm is said to “overfit” when it is overly adapted to the training dataset and generalizes poorly to unseen data (Russell and Norvig 2020). Throughout the thesis, we use the term “overfit” when the control software generated by an automatic design method performs well in simulation but does not generalize to reality. In our case, training is performed in simulation, while the generated control software is eventually executed on real robots. What we expect is that control software generated in simulation generalizes to reality in a similar way as we expect that a learning method generalizes to unseen data.

In automatic design of control software for robot swarms, overfitting is illustrated in Birattari et al. (2016), where the authors present empirical results and show that when the design effort increases after a certain point, the performance of control software will continue increasing in simulation, while decreasing in reality (Figure 2.3a). The authors qualify this specific type of overfitting as *overdesign*. Similarly, in Birattari (2009), the author presents and shows results in algorithm configuration, and specifically metaheuristics tuning, and show that as the tuning effort increases after a certain point, the performance of the tuned algorithm on the tuning instance continues increasing while the performance on a test instance decreases (Figure 2.3b). The author qualifies this specific type of overfitting as *overtuning*.

Throughout this thesis, we can observe performance drops in the results of assessments of automatic design methods. Different factors can contribute to these performance drops, however, we focus on the issue of overfitting because other factors, like discrepancies between simulation and reality, affect all design methods in the same way, while overfitting is a method-specific factor; to the best of our understanding, the only method-specific one. We are particularly interested in understanding why some design methods are misled by the simulator while others are not, which translates to why some methods overfit the simulation more than others.

2.4 Notable research in automatic design

We have presented four different cross categories: offline semi-automatic, online semi-automatic, offline automatic, and online automatic design. These categories did not benefit from the same attention in the literature: the most studied category is offline semi-automatic, followed by online semi-automatic, then offline automatic,

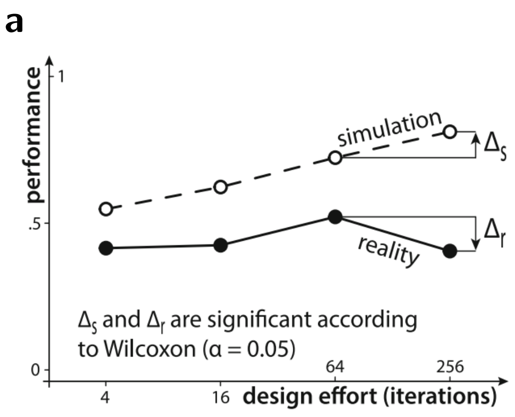


Figure published in Birattari et al. (2016)

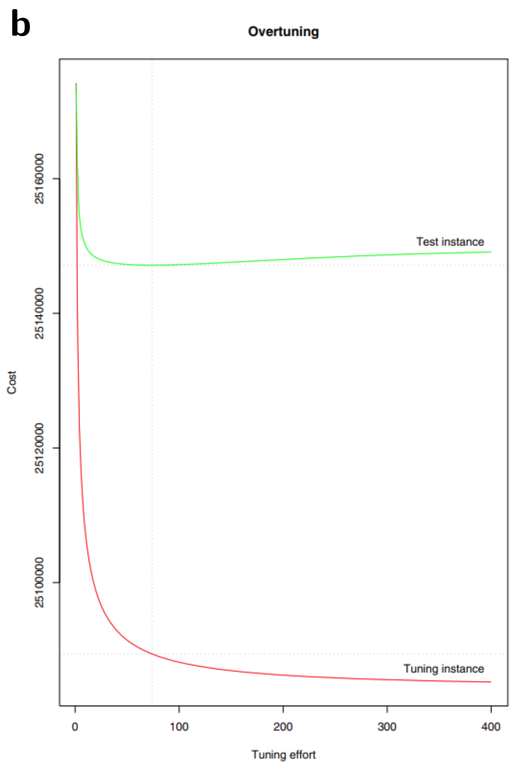


Figure published in Birattari (2009)

Figure 2.3: **Overdesign and overtuning.** **a**, *overdesign* (Birattari et al. 2016): when the design effort increases after a certain point, the performance of the automatic design method continues increasing in simulation, while the one in reality decreases; **b**, *overtuning* (Birattari 2009): when the tuning effort increases after a certain point, the performance of the tuned algorithm on the tuning instance continues increasing, while the one on a test instance decreases.

Table 2.1: **Overview of automatic design methods.** In the real robot experiments an indication of *limited* experiments indicates that they are limited to a proof of concept. An indication of semi* in the semi/fully automatic column indicates that we lack information to definitely classify the study as belonging to fully-automatic design.

	on/off-line	semi/fully automatic	number of robots	missions	real robot experiments
Watson et al. 1999, 2002	online	semi*	8	phototaxis	yes
Quinn et al. 2003	offline	semi	3	coordinated navigation	limited
Usui and Arita 2003	online	semi*	6	collision avoidance	yes
Bianco and Nolfi 2004	online	semi	64	self-assembly	no
Dorigo et al. 2003	offline	semi	4-8	aggregation and coordinated navigation	no
Pugh et al. 2005	offline	semi	2	obstacle avoidance	no
Christensen and Dorigo 2006	offline	semi	3	hole-avoidance and phototaxis	limited
Trianni and Dorigo 2006	offline	semi	4	hole-avoidance	yes
Trianni and Nolfi 2009	offline	semi*	96 (2-3 in reality)	synchronization	limited
Hauert et al. 2009	offline	semi*	20	network connectivity	no
Ampatzis et al. 2009	offline	semi	2	self-assembly	yes
Pugh and Martinoli 2009	online	semi	10	obstacle avoidance	yes
König et al. 2009	online	semi*	26	collision avoidance and gate passing	no
Waibel et al. 2009	offline	fully	10	foraging	yes
Hettiarachchi and Spears 2009	hybrid	semi*	40-100	obstacle avoidance	no
Hecker et al. 2012	offline	semi	100 (1-3 in reality)	foraging	limited
Bredeche et al. 2012	online	semi	20	survival and consensus	yes
Gomes et al. 2013	offline	semi	5	aggregation and resource sharing	no
Gauci et al. 2014a	offline	semi*	10-1000	object clustering	no
Gauci et al. 2014b	offline	semi*	10-1000 (40 in reality)	aggregation	limited
Duarte et al. 2014b	offline	semi	1000	patrolling	no
Haasdijk et al. 2014	online	semi*	100	objects foraging	no
Francesca et al. 2014a,b	offline	fully	20	aggregation and foraging	yes
Francesca et al. 2015	offline	fully	20	shelter w/ constrained access, largest covering network, coverage w/ forbidden areas, surface & perimeter coverage, and aggregation w/ cues	yes
Ferrante et al. 2015	offline	semi	5-20	foraging	no
Silva et al. 2015	online	semi	5	aggregation, navigation, and phototaxis	no
Duarte et al. 2016a	offline	semi	10	homing, dispersion, clustering, and area monitoring	yes
Montanier et al. 2016	online	semi	100-500	foraging	no
Fernández Pérez et al. 2017	online	semi	200	foraging	no
Jones et al. 2018	offline	semi	25	foraging	yes
Hasselmann and Birattari 2020; Hasselmann et al. 2018b	offline	fully	20	aggregation, stop, and decision	yes
Kuckling et al. 2018; Ligot et al. 2020b	offline	fully	20	foraging and aggregation	yes
Jones et al. 2019	online	semi	20	collective foraging	yes
Garzón Ramos and Birattari 2020	offline	fully	20	aggregation, stop, and foraging	yes
Spaey et al. 2020	offline	fully	20	aggregation, grid exploration, and foraging	yes
Kuckling et al. 2020b	offline	fully	20	aggregation and foraging	no

and online automatic design. To the best of our knowledge, no study on online fully-automatic design has been published, yet. In this section, we present a review of notable research in the automatic design of robot swarms. We classify studies according to the framework of categorization that we introduced previously. A summary of the presented works, including fundamental characteristics, is available in Table 2.1. Some of the works presented in this section could not be classified with absolute certainty as belonging to either semi-automatic or fully-automatic design. Indeed, despite the fact that some of these works do not mention any human intervention in the design process, we cannot classify them as fully-automatic design when: (i) it is not clearly explained how the values of some parameters were fixed and whether this required the intervention of a human at design time, (ii) the design method is tackling only one specific mission and its performance is assessed only on this mission.

2.4.1 Offline semi-automatic design

Offline semi-automatic design is the most studied category of the four. The majority of the notable works in semi-automatic design use neuroevolution to evolve control software for robot swarms. In neuroevolution (Trianni 2008), typically, each robot is controlled by a neural network whose inputs are mapped to the robot's sensory inputs and whose outputs are mapped to the robot's actuators. The neural networks are typically trained using an evolutionary algorithm.

The majority of the studies dealing with semi-automatic design share common characteristics. Typically, the robot swarm is homogeneous, meaning that all robots within the swarm execute an instance of the same control software. The objective function used to optimize the control software and assess the performance of the swarm is computed off-board the robots and from a global perspective—that is, the performance metrics are defined at the swarm level.

Quinn et al. (2003) were the first to use neuroevolution to design control software for robot swarms. They evolved three robots to perform a coordinated navigation mission. The robots were only equipped with infrared sensors and started from a random position. The authors presented results of 10 evolutionary runs, with 3000 to 5000 generations per run. Out of the 10 evolutionary runs, four of them were stopped early due to unsatisfactory behavior. The best control software was assessed for 100 trials in simulation and demonstrated in reality, however no quantitative results obtained in reality were reported.

Dorigo et al. (2003) presented a swarm system called *swarm-bot*, composed of four to eight *s-bots* robots. They explored the design of control software using neural networks for two tasks, aggregation and coordinated motion. For each mission, the neural networks' structure is different and the evolved controllers are validated using simulation. The authors planned to test the best ones on real robots, however their performance was assessed using simulation only. They reported satisfactory results and simple and generally robust behaviors.

Pugh et al. (2005) designed control software for a swarm of 2 *Khepera* robots for an obstacle avoidance mission using particle swarm optimization. The authors explored how particle swarm optimization can be used to train a neural network. They compared two genetic algorithms with 3 variants of the particle swarm optimization algorithm. The parameters of the optimization algorithm were adjusted during the design phase based on empirical results. They ran the design once per scenario and compared the performance of the best instances of control software they obtained. Results were reported in simulation only. They showed that particle swarm optimization is a viable alternative to genetic algorithms to design robot swarms.

Christensen and Dorigo (2006) designed control software simultaneously for hole-avoidance and phototaxis using a group of *s-bots* robots using neuroevolution. They used three different evolutionary algorithms to train four different neural networks using simulations; 20 designs of 1000 generations were run for each of them. The fitness function, neural network structure, and algorithm parameters were specifically tailored to the missions at hand. The best instances of control software obtained were then ported to a swarm of three real robots. Adjustments in the speed of robots, sensor noise and environment were made to mitigate the reality gap. The control software allegedly successfully transferred to the real robots, however no quantitative results were provided.

Trianni and Dorigo (2006) designed control software for navigation with hole-avoidance using neuroevolution for a swarm of physically connected *s-bots* that exploit direct communication. They compared three neural network structures, representing three different communication strategies. The design process was run 10 times for each communication strategy, the best control software obtained for each strategy was tested 30 times on a swarm of 4 robots. They showed that direct communication can be trained without explicit reward using neuroevolution. After selection based on a carefully crafted performance metric, adjustments were made to run the bests possible control software on real robots. The authors report small differences between simulation and reality.

Trianni and Nolfi (2009) designed control software for different robot synchronization strategies for a swarm of *s-bots* using neuroevolution. The authors studied synchronization strategies in simulation with up to 96 robots. Results show that all strategies scale effectively. They tested the best control software they obtained on real robots; they ran 20 trials with two robots, and 20 trials with three robots. These experiments showed that the behavior is limited by the collision avoidance system of the swarm. The authors applied dynamical system analysis to explain the evolution mechanism and predict the swarm behavior. The authors claimed that their analysis can be used to bring insight on how to design robots swarm, using automatic or manual design, for self-organised synchronization.

Hauert et al. (2009) designed control software for a swarm or UAVs for a network connectivity mission using neuroevolution. The mission consisted in establishing and maintaining a multi-hop wireless network between users on the ground using UAVs. The UAVs only rely on heading measurements and local communication. They evolved a neural network with four inputs, one output and 4 hidden nodes using an evolutionary algorithm. The design was ran 15 times. The analysis was focused on the performance of the best control software they obtained. Results were reported in simulation using 20 UAVs. Results showed that the system was able to find user stations and maintain an active network.

Ampatzis et al. (2009) designed control software for a self-assembly task between two *s-bots* robots using neuroevolution. The objective functions that were used were crafted not only to encourage self-assembly, but also to encourage aggregation, minimize collisions, and encourage straight movement when robots are close, the latter being reported as been added to ease transferability to real hardware. The design process was run 20 times and the best control software obtained were then tested on real robots. The authors reported good experimental results both in simulation and reality.

Hecker et al. (2012) designed control software for a swarm of *iAnt* robots for a foraging mission using evolutionary algorithms. They fine-tuned the parameters of mission-specific modules used in finite-state machines in order to obtain the desired behavior. Experiments were conducted using 1 and 3 robots both in simulation and reality. The authors reported satisfactory behaviors both in simulation and reality. The system was able to scale in the simulation setup to up to 100 robots.

Gomes et al. (2013) designed control software for a swarm of 5 robots for one aggregation and one resource sharing mission using novelty search. The algorithm they used is NEAT (Stanley and Miikkulainen 2002), combined with a special objective function that favors novelty instead of a mission specific fitness, to train a simple neural network. They compared this method with a classical neuroevolutionary approach and a hybrid approach. They showed that novelty search is a promising approach, as it achieved similar results for the aggregation mission and better results in the resource sharing mission than the other methods. Some parameters of the algorithm were chosen per-mission. The number of generations was 100 for the first mission and 400 for the second mission, also the behaviour characterization differs in the two missions. The experiments were conducted in simulation only.

Gauci et al. (2014a) designed control software for a swarm of e-puck robots for an object clustering mission using an evolutionary algorithm. The robots sensing capabilities were limited to a minimum, with only one line-of-sight sensor. The control software architecture consisted of an array directly associating the states of the line-of-sight sensor to motor speeds. The robot did not require any computation. Three cases were compared, where the line-of-sight sensor could: (i) detect and distinguish between objects and robots (ii) only detect objects (iii) detect objects and robots but not distinguish them. The control software was designed in simulation using the `CMA-ES` (Hansen and Ostermeier 2001) algorithm, three designs per case were done, of 1000 generations. The control software they obtained were then tested in simulation on various swarm sizes and number of objects. The best control software they obtained were then ported and tested 10 times on a swarm of 5 real robots. Results showed that, even with such a minimal setup, robots were able to cluster most of the objects.

Gauci et al. (2014b) developed the same idea presented in the previous article: they designed a swarm of e-puck robots with only one binary sensor for an aggregation mission. In this study the line-of-sight sensor can detect whether there is a robot in front, or not. Here, the adopted optimization algorithm is a simple grid search as the search space of candidate control software is small. It only consists of an array of four parameters, associating states of the sensor to motor speeds. The entire search space of parameters was searched with floating points numbers with a 0.1 resolution on each parameter, with 100 simulations per candidate. They tested the optimal solution with 10 to 1000 robots in simulation and in reality with 40 robots.

Duarte et al. (2014b) designed control software for a swarm of up to 1000 aquatic drones for a patrolling mission using hierarchical decomposition and neuroevolution. The authors first used manual hierarchical decomposition to define sub-tasks and then evolved low-level behaviors using neuroevolution on these sub-tasks. They combined these mission-specific low-level modules into a neural network arbitrator. The authors reported experimental results in simulation only. The same method is also described in Duarte et al. (2014a)

Ferrante et al. (2015) designed control software for a swarm of *foot-bot* robots for a foraging mission with task specialization using Grammatical Evolution (Ferrante et al. 2013). Two different methods were used, the first using pre-defined high-level behavioral modules and a second one using simpler behavioral modules. The modules differ in their complexity but are tailored to the specific mission at hand. Results showed the emergence of task-allocation in both scenarios with success. The experiments were conducted in simulation only using a swarm of 5 to 20 *foot-bot* robots.

Duarte et al. (2016a) designed control software for a swarm of 10 aquatic robots across four different missions using neuroevolution. The authors added noise to the simulation and tested multiple initial positions, number of robots, and environmental features to bootstrap the evolution. The design process was run 10 times per mission, the NEAT algorithm was used with its default parameters. The best control software of each generation was assessed in 100 simulations; 3 out of these 10 controllers were tested on real aquatic robots. Results reported good performance in all missions. The second part of this study introduced a proof of concept on a more complex mission. The mission was achieved by combining the instances of control software obtained in the first part of the study.

Jones et al. (2018) designed control software for a swarm of 25 *kilobots* on a foraging mission using behavior trees as the control architecture and an evolutionary algorithm. The authors defined high-level behaviors that were mission-specific. These behaviors were used as modules in the interface between the robot and the behavior trees. The results showed that a behavior tree was successfully evolved for the foraging task. The authors reported a significant difference between the performance in simulation and in reality, due to the reality gap. The robots were still able to effectively forage.

2.4.2 Online semi-automatic design

In this section, we present the state of the art in online semi-automatic design. As this thesis focuses primarily on offline methods, we selected only the most relevant studies in online automatic design. For a more comprehensive overview of the state of the art we refer the reader to Bredeche et al. (2018).

In the literature, online automatic design is also referred to as *embodied evolution*. It is the study of distributed learning methods for the online automatic design of robot swarms. A majority of studies use embodied neuroevolution, in which the control software of the robots, possibly a neural network, is expressed in the form of a genome. This genome evolves while the swarm operates by mutation and/or recombination and by sharing genetic information with neighboring robots.

Watson et al. (1999, 2002) were the first to use embodied evolution for evolutionary robotics. They designed software for a swarm of eight robots for a phototaxis mission. The control software architecture was a fully connected neural network whose topology was tailored to the mission at hand. The robots sent their genome and a performance score to other neighboring robots to share genetic information. The experiments were run on real robots, the results showed that an effective phototaxis strategy emerged that outperformed the manually designed strategy.

Usui and Arita (2003) designed control software for a swarm of six *Khepera* robots for collision avoidance using embodied neuroevolution. They used a simple two-layer neural network as control architecture. They compared four cases by varying gene pool sizes. The gene pool size is the number of instances of genomes stored in each robot and that they can memorize and share. The experiments were run on real robots. Results showed that sharing well-performing instances of control software between robots is beneficial for swarm performance.

Bianco and Nolfi (2004) proposed a framework for robot self-assembly. The authors chose a set of simple rules and let the swarm evolve freely, without any explicit objective function. The robots are controlled using a feed-forward neural network with no hidden layer. Three scenarios were imagined leveraging different robot capabilities to let different behaviors emerge. The design was run five times for each scenario. Results were reported in simulation only using 64 robots. Behaviors spontaneously emerged, leading to coordinated motion and assembled robot structures.

Pugh and Martinoli (2009) designed control software for a swarm of *Khepera* robots using particle swarm optimization for an obstacle avoidance mission. The authors explored how particle swarm optimization can be adapted for distributed design. The algorithm was compared to a distributed evolutionary algorithm. The optimization algorithms are used to train a fully-connected feed-forward neural network. The authors studied the effect of varying sizes of robot swarm and varying communication ranges. The best instances of control software were tested on 10 real robots. The experiments were adapted to better fit the simulation setup. Results showed that particle swarm optimization is a viable alternative to evolutionary algorithms for the online design of robot swarms.

König et al. (2009) designed control software for a swarm of 26 *Jasmine IIIp* robots for collision avoidance and gate passing using embodied neuroevolution and finite-state machines. Using different fitness functions, genome recombination strategies, number of parents for genome reproduction, and genome memory, the authors defined 8 different distributed experiments to evolve finite-state machines online. The experiments were run for 80000 cycles using simulation only. Results showed that increasing the number of parents associated with specific recombination strategies lead to improved performances.

Hettiarachchi and Spears (2009) designed control software for a swarm of 40 to 100 robots for obstacle avoidance using force laws and evolutionary algorithms. The parameters of the force laws determined how robots behaved and travelled through contingencies in the environment. The control software was defined directly by the force law parameters. It is first trained offline using an evolutionary algorithm in a simple environment. Then, the control software is ported on the robots to the target, more complex, environment. The control software adapts online to this environment. This method can be considered as a hybrid method with a focus on the *DAEDALUS* framework, introduced by the authors, for the online adaptation of the swarm. The control software architecture was heavily biased towards solving the specific mission at hand. Results were reported using simulation only. They compared two different force laws and showed that, in their experiments, the Lennard-Jones force law outperformed the Newtonian force law.

Bredeche et al. (2012) presented *mEDEA*, a distributed embodied neuroevolutionary algorithm using neural networks. Using this algorithm, robots could adapt to changes in their environment. The authors designed software for a swarm of 20 e-puck robots for two missions: a survival mission and a consensus mission. 21 experiments were run on real robots. Adjustments were made to the neural network topologies, and to the number of control software instance saved per robot, in order to port control software to real robots. The communication radius and the swarm size were key factors to the success of the mission. Eight supplementary experiments were run on the basis on the observation of the initial ones, to explore, in details, the performance of *mEDEA*. Results showed that *mEDEA* was efficient at providing adaptation in unknown environments and robust to changes in the environment.

Haasdijk et al. (2014) presented *MONEE*, a variant of the *mEDEA* algorithm. *MONEE* is a multi-objective and distributed neuroevolutionary algorithm that allowed robots to adapt to changes in their environment but also to adapt to a given mission. The mission that was considered was concurrent foraging with two types of objects. The robots executed a single-layer feed-forward neural network. This architecture was chosen based on trials with different architectures. The design process was run 64 times. Experiments were reported using simulation only on a swarm of 100 e-puck robots. They compared two variants of *MONEE* with *mEDEA*, results showed that is is possible to combine environmental and mission driven evolution, and that *MONEE* outperformed *mEDEA* on the considered mission.

Silva et al. (2015) presented a distributed neuroevolution algorithm called odNEAT, based on the NEAT algorithm (Stanley and Miikkulainen 2002). They compared it with rtNEAT, a previously presented offline method, on three missions: aggregation, obstacle avoidance, and phototaxis. The neural network topologies were adjusted for each mission. Results were reported using simulation only with a swarm of 5 robots. The results showed that odNEAT performed as well as rtNEAT on the considered missions. The robots exhibited better sensor fault tolerance when running odNEAT. Before deployment, tests across all missions were conducted to determine suitable parameters for the odNEAT algorithm.

Montanier et al. (2016) studied the emergence of behavioral specialization of robots in a swarm for a foraging mission, with multiple resources. Two variations of the algorithm were assessed using simulation with a population of 100, 200, and 500 robots. Results showed that robots must be isolated, or that the swarm must have a large population size, in order to exhibit behavioral specialization in the presented mission. The article only presented one task and did not provide any information about potential mission-specific adjustments.

Fernández Pérez et al. (2017) presented a modified version of the *mEDEA* algorithm, with task-driven selection pressure. It was tested on a swarm of 200 robots using simulation, for a cooperative food foraging mission. The robots successfully evolved control software that was able to forage food cooperatively. The parameters of the algorithm for the mission were determined using preliminary experiments.

Jones et al. (2019) designed control software for a swarm of 20 eXtended e-puck robots for a collective foraging mission using behavior trees evolved by an embedded simulation-based evolutionary algorithm. The robots designed candidate control software using simulations on their on-board computers while simultaneously executing the best control software instance so far. They exchanged information about the best candidate control software instance with their neighboring peers. The objective function was adjusted to bootstrap the evolution. Real-robot experiments were run before deployment to adjust the friction and collisions model of the simulation to the reality. An obstacle avoidance mechanism was also added. Twenty experiments were run using real robots. The authors reported that robots could find efficient solutions in less than 15 real-time minutes.

2.4.3 Offline automatic design

In this section, we present offline fully-automatic design methods. This category contains works that do not involve any human intervention in the design process nor any per-mission modification of the method during the design process. However, human expertise is still required when implementing the method, for example by creating modules in modular methods (see Section 2.5 for details). Human expertise is also needed when defining the specifications of a mission, including the definition of the objective function and target environment.

Waibel et al. (2009) designed control software for a swarm of 10 robots for multiple variants of a foraging mission using neuroevolution. They studied different evolutionary conditions to create both homogeneous and heterogeneous swarms using team-level and individual-level selection. The robots are controlled by a feed-forward neural network with one hidden layer. The design process was run 20 times for each of the four evolutionary conditions and three tasks, for 300 generations. Each instance of control software was then evaluated 10 times using real robots. Results showed that, although significant differences between evolutionary conditions were observed and that all evolutionary conditions evolved the foraging behavior, none of them was optimal in all missions.

Francesca et al. (2014b) designed control software for a swarm of 20 e-puck robots for an aggregation and a foraging mission using modular design. This study is the first that presented a method of the AutoMoDe family. The idea of AutoMoDe is to leverage the bias-variance trade-off by injecting bias into the design process by using pre-existing modules. The authors presented AutoMoDe-`Vanilla`, a method that uses pre-existing hand-crafted modules to assemble and fine-tune probabilistic finite state machines. The method is compared to a classical neuroevolutionary method called `EvoStick`. The design process was run 20 times with 3 different simulation budgets for each mission and each method. The optimization algorithm is the *F-race* algorithm (Birattari et al. 2002). The results were reported using both simulation and real robots. The authors reported good results on all missions, `Vanilla` outperformed the classical neuroevolutionary approach and was also more robust to the reality gap. This study laid the foundation of many other automatic design methods introduced in the following years and presented hereafter.

In a subsequent study, Francesca et al. (2014a), added a comparison with human experts. The two methods of the previous study were compared to control software produced by five different human experts under the same experimental setting. Humans conceived control software using two manual methods: *U-Human* where they were unconstrained, meaning they designed the control software from scratch, and *C-Human* where they were constrained, meaning they designed the control software by assembling the same modules of `Vanilla`, effectively creating and fine-tuning probabilistic finite-state machines. In this second case, they took the same role as the optimization algorithm of `Vanilla`. The results were reported using simulations and real robots. The constrained humans (*C-Human*), outperformed all other methods, followed by `Vanilla`, itself outperforming `EvoStick` and the unconstrained humans (*U-Human*).

These results were the starting point for the creation of AutoMoDe-`Chocolate` (Francesca et al. 2015). As *C-Human* used the same modules as `Vanilla` did, the authors had the intuition that adopting a more powerful optimization algorithm could yield better performance for `Vanilla`. The authors tested this hypothesis by adopting the irace (López-Ibáñez et al. 2016) optimization algorithm to replace *F-race*. The methods were tested for five different missions and `Chocolate`, `Vanilla`, and *C-Human* were compared using simulations and real robots. The results showed that `Chocolate` outperformed *C-Human* and `Vanilla`. Making `Chocolate` the first automatic design method to outperform human designers.

Following the success and good results of `Chocolate`, other methods of the AutoMoDe family were published. These novel methods all leverage the idea of modular design to increase their robustness to the reality gap. They explored different aspects of automatic modular design, by introducing new modules (Garzón Ramos and Birattari 2020; Hasselmann and Birattari 2020; Hasselmann et al. 2018c; Spaey et al. 2020), trying different control architectures (Kuckling et al. 2018; Ligot et al. 2020b), or different optimization algorithms (Kuckling et al. 2020b). These studies, presented hereafter, all use a swarm of 20 e-puck robots. All results are supported by simulations and real-robot experiments and all methods were compared them other design methods, often neuroevolutionary ones, on several missions.

In Hasselmann et al. (2018b), and the subsequent work, Hasselmann and Birattari (2020), we presented AutoMoDe-`Gianduja`. This method extended `Chocolate` by introducing modules that allowed explicit communication between robots. The results showed that it is possible to automatically design a swarm of communicating robots, without any explicit reward for communication. The modular method outperformed the neuroevolutionary one, `EvoCom`, a communication-enabled variant of `EvoStick`.

In Kuckling et al. (2018), and the subsequent work, Ligot et al. (2020b), the authors introduced AutoMoDe-`Maple`. They explored the effect of using behavior trees (Champandard 2007) as the control architecture on the performance of the design method, instead of probabilistic finite-state machines. The results showed that `Maple` performed similarly to `Chocolate`; using behavior trees did not degrade the results in any sensible way. The two modular methods presented in this study outperformed the neuroevolutionary one, `EvoStick`.

Garzón Ramos and Birattari (2020) presented AutoMoDe-`TuttiFrutti`, an extension of `Chocolate` allowing robots to display and perceive colors using RGB LEDs and cameras. This method enhanced the capabilities of `Chocolate`: it gave the ability to a swarm of e-puck robots to use color information for communication and navigation.

Spaey et al. (2020) presented AutoMoDe-`Coconut`, an extension of `Chocolate` with configurable exploration schemes. They compared multiple random-walk and exploration schemes and explored how these affect the performance of the design method. Results indicated that the exploration scheme already implemented in `Chocolate` is a suitable choice as `Coconut` and `Chocolate` performed similarly in the missions considered.

Kuckling et al. (2020b) presented AutoMoDe-`IcePop`. In this study, the authors explored how the optimization algorithm used in `Chocolate` influenced the performance of the design method. They replaced the irace algorithm used in `Chocolate`, by a stochastic local search metaheuristic, the simulated annealing algorithm (Nikolaev and Jacobson 2010). `IcePop` outperformed `Chocolate` in simulation on one of the two missions, but not in the *pseudo-reality* (Ligot and Birattari 2018). Results indicated that simulated annealing is a viable replacement for irace for automatic design.

The works presented in Chapters 4, 5, and 6 of this thesis introduce an empirical comparison of automatic design methods and two novel design methods that belong to this category.

2.5 Modular methods for fully-automatic design

In the previous section, we presented notable research in automatic design. A subset of the presented automatic design methods belong to modular design. In modular design, the optimization algorithm operates on modules. During the design process the algorithm combines modules into a high-level arbitrator architecture. For example, behavioral modules that execute low-level actions on the robots, like going to a light, or avoiding an obstacle, could be assembled into a probabilistic finite-state machine. Modules are software components, that can be executed by the robots. Their creation may be manual or automatic. They may be hand-crafted modules, neural networks, or any other type of software architecture. The implementation of these modules, depends of the robotic platform selected to use with the method.

In fully-automatic design, modules have additional constraints: In order to qualify as fully-automatic, a method must not undergo any per-mission modification, that is why the modules must be defined once-and-for-all and be subsequently used for all designs runs of the method. They are thus part of the definition of the method. Modules may be created manually or automatically, provided that, once defined, they are not modified on a per-mission basis. Modular design methods are widespread in fully-automatic design, especially since the development of AutoMoDe (Francesca et al. 2014b). One major advantage of modular methods is that research suggests that they are more robust to the reality gap than the classical neuroevolutionary ones (Francesca et al. 2015; Hasselmann and Birattari 2020; Kuckling et al. 2020a). The main challenge in modular design lies in the creation of the modules. For instance, the level of abstraction of modules (high-level or low-level modules) and the number of modules, are factors that define the class of missions that the swarm could perform. The way the modules are designed, and the way they are combined, are factors that define the performance of the swarm and the robustness of the method to the reality gap.

We present here a review of modular methods, the ones already mentioned in Section 2.4 are briefly summarized:

Ferrante et al. (2013) and Ferrante et al. (2015) used modules in the form of hand-crafted low-level behaviors. These are then assembled offline using grammatical evolution (O'Neill and Ryan 2003). The evolved high-level arbitrator corresponds to a set of rules that controls how the control software switches from one low-level behavior to another. These rules are based on the internal or sensory state of the robot.

Francesca et al. (2014b) presented AutoMoDe, a family of methods, and AutoMoDe-`Vanilla` a modular method that used hand-crafted and pre-defined modules, then assembled into probabilistic finite-state machines. Modular methods belonging to the AutoMoDe family have shown to be more robust to the reality gap than traditional neuroevolutionary methods. Several methods from the AutoMoDe family were presented in the recent years (Francesca et al. 2015; Hasselmann and Birattari 2020; Kuckling et al. 2020a,b; Mendiburu et al. 2022; Spaey et al. 2020).

Duarte et al. (2014a) and Duarte et al. (2014b) used low-level behaviors in the form of neural networks trained using neuroevolution, and manually designed modules. The authors first decomposed missions manually using hierarchical decomposition and then created low-level modules for these sub-missions. They either: designed the module manually when the mission had no immediate objective function, or trained a neural network using neuroevolution if an objective function could be easily defined.These modules were then combined offline in a high-level arbitrator using neuroevolution.

Duarte et al. (2016a) started with simple missions with control software in the form of neural networks evolved via neuroevolution in the first part of the study. Then, for a more complex mission, they assembled, offline and manually, the modules created in the first part of the study.

Jones et al. (2018) used behavior trees as the arbitrator architecture and assembled high-level behavioral modules that were mission-specific. The modules are created manually, and assembled offline using an evolutionary algorithm.

Gomes and Christensen (2018b) presented an offline method to generate a repertoire of diverse swarm behaviors. They used a quality-diversity (Pugh et al. 2016) algorithm relying on a quality metric and a behavior characterization to create a mission-agnostic set of neural network modules (the repertoire). They tested the modules by assessing them on eight missions. Results showed that control software suitable for all missions was present in the repertoire. Experiments were done in simulation only and modules were not assembled.

Jones et al. (2019) used behavior trees evolved online using an embedded simulation-based evolutionary algorithm. The robots executed the best candidate control software, while simulations were running on their on-board computer.

Neupane and Goodrich (2019) used behavior trees evolved using using grammatical evolution (O'Neill and Ryan 2003). The authors defined actions and conditions nodes, used as modules. These were hand-crafted. The behavior tree was then generated by combining these low-level modules.

Cambier and Ferrante (2022) used several evolutionary algorithms to assemble hand-crafted modules. The authors used the same behavior and condition modules as the ones of `Chocolate` (Francesca and Birattari 2016) and generated probabilistic finite-state machines using three different algorithms: a genetic algorithm, grammatical evolution, and differential evolution. Experiments were presented using simulation only.

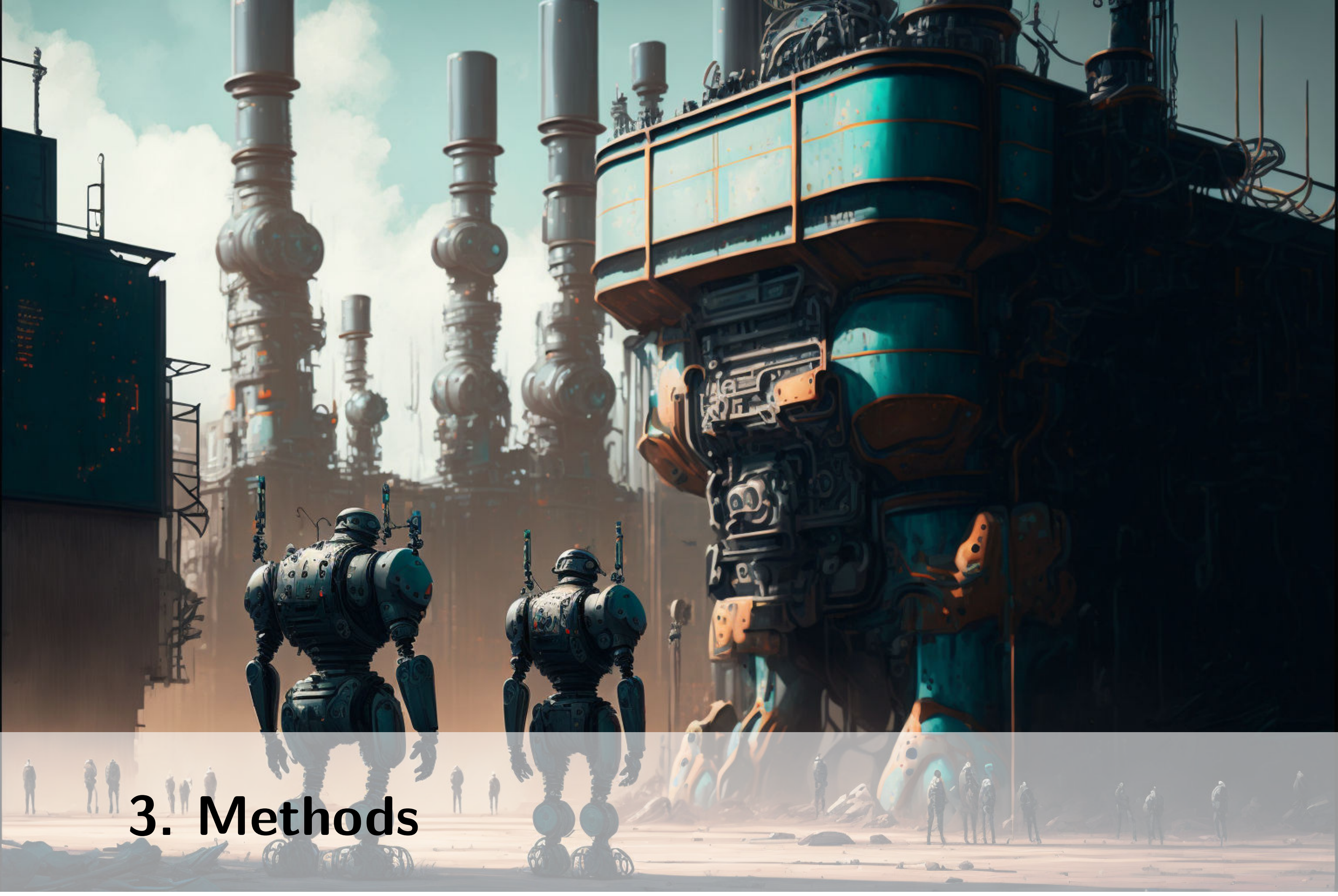
2.6 Discussion

In this chapter, we presented a general introduction to swarm robotics as an appealing approach for a wide variety of robotic applications. The properties of swarm robotic systems—fault tolerance, scalability, and flexibility—make them particularly suitable for complex applications that benefit from the distributed nature of these systems. The design of these systems is one of the main challenges of the field. Deriving individual behavior from the collective behavior of robots is a difficult engineering problem.

Automatic design for robot swarms is a promising approach to tackle the design problem. We proposed a novel classification of automatic design methods, separating semi-automatic methods from fully-automatic design methods. The main difference between these approaches is whether or not a human designer was involved in the design phase of a method. Semi-automatic methods being described as "human-in-the-loop" methods, while fully-automatic design methods prohibit it.

We reviewed the most notable research in the automatic design of robot swarms. We focused on the description of methods and we determined if these methods could be classified as either semi- or (fully-)automatic design. It is our contention that the advancement of the field of automatic design would benefit from systematic real-robot experiments, as we have seen that the reality gap problem is a significant issue when assessing the performance of control software designed using simulation only. Therefore, we gave details on experimental protocols, and specifically on the adoption of real-robot experiments to validate simulation results.

The last part of the chapter focuses on modular automatic design methods; we believe that modular design is a promising approach to the creation of robust automatic design methods. However, as of today, the implementation of popular modular automatic design methods still requires expert knowledge. The following chapters of this thesis aim to shed light on the issues that we highlighted. We present the most extensive comparative analysis of offline automatic design methods currently available in the literature. This study follows an experimental protocol that emphasizes the importance of real-robot experiments for the mitigation of the effects of the reality gap. The two automatic modular design methods introduced afterwards were created as an attempt to bridge the gap between neuroevolutionary methods and modular methods, to reduce expert knowledge needed in the implementation of modular methods while maintaining sufficient robustness to the reality gap.

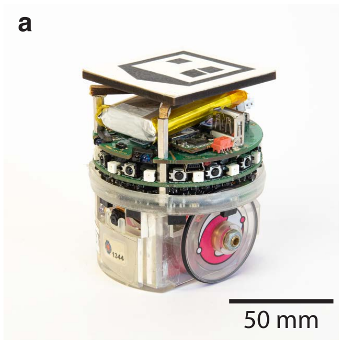


3. Methods

This chapter presents the materials and methods that are common to all experiments in this thesis. We describe the robotic platform we use, its reference model, the simulator, the arena setup, the statistical tools and two automatic design methods that are present in all our experiments as comparison points: `EvoStick` and `Chocolate`.

3.1 The e-puck robot

The robot used in the research is the e-puck (Mondada et al. 2009), a small differential-drive robot that measures 50 mm of height and 70 mm of diameter. The e-puck is equipped with several sensors and actuators: It has eight infrared transceivers positioned all around its body to detect the presence of surrounding obstacles and/or measure the intensity of the ambient light. It has three ground sensors to read the gray-scale color of the ground beneath. For the purposes of the research presented in this thesis, the e-puck was enhanced with two extension boards: (i) the range-and-bearing (Gutiérrez et al. 2009), which enables a robot a



b

Input	Value	Description
$\text{prox}_{i\in\{1,\ldots,8\}}$	$[0,1]$	reading of proximity sensor i
$\text{light}_{i\in\{1,\ldots,8\}}$	$[0,1]$	reading of light sensor i
$\text{gnd}_{j\in\{1,2,3\}}$	$\{\text{black, gray, white}\}$	reading of ground sensor j
n	$[0,19]$	number of neighbouring peers perceived
V	$([0.5,20],[0,2\pi]\ \text{rad})$	range-and-bearing vector

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

Period of the control cycle: 100 ms

Figure 3.1: **The robot and its reference model.** **a**, the e-puck robot in the configuration used for the experiments presented in this thesis. Details are provided in Section 3.1. **b**, the reference model RM 1.1, which formally describes the programming interface through which the control software interacts with the robot's hardware.

to sense the presence of neighboring peers and estimate their relative position; and (ii) the Overo Gumstix, a Linux board that increases the computing power and flexibility of the robot. The robot is also equipped with two batteries and its autonomy is around 35 min. A picture of the e-puck in the configuration adopted in the experiments is given in Fig. 3.1a.

3.2 Reference model

This section describes the reference model used for the e-puck robot in this thesis. A reference model formally describes the programming interface through which, in the experiments presented in the thesis, the control software interacts with the underlying hardware.

In this thesis, the e-puck is formally described by the reference model RM 1.1 given in Figure 3.1b. All design methods comprised in this thesis generate control software that interacts with the e-puck exclusively through the variables defined in RM 1.1.

According to RM 1.1, the reading of a proximity sensor i is stored in the variable $prox_i$, which ranges between 0 and 1. When sensor i does not perceive any obstacle in a 0.03 m range, $prox_i = 0$; while when sensor i perceives an obstacle closer than 0.01 m, $prox_i = 1$. Similarly, the reading of a light sensor i is stored in the variable $light_i$, which ranges between 0, when no light source is perceived, and 1, when the sensor i saturates. The readings of the three ground sensors are stored in the variables gnd_1, gnd_2 and gnd_3. These variables can take three different values: black, gray and white. The e-puck robot uses the range-and-bearing board to perceive other e-pucks in its neighborhood. The variable n stores the number of the neighboring e-pucks. The e-puck detects the relative position of its neighbors by the mean of a range-and-bearing vector. The range-and-bearing vector points to the aggregate position of the neighboring peers perceived; its magnitude increases with the number of neighboring peers perceived and decreases with their distance. Formally, this vector is computed as follows:

$$V_b = \begin{cases} \sum_{m=1}^{n} (\frac{1}{1+r_m}, \angle b_m), & \text{if robots are perceived;} \\ (1, \angle 0), & \text{otherwise.} \end{cases}$$

where r_m and $\angle b_m$ are range and bearing of neighbor m, respectively (see Figure 3.1b). If no neighboring peer is perceived, the vector points to the front of the robot and has unitary magnitude; formally, $V = (1, \angle 0)$.

The wheel actuators are operated by the control software through the variables v_l and v_r, in which it writes the target linear velocity for the left and right wheel, respectively. The linear wheel velocity ranges between $-0.12\,\mathrm{m/s}$ and $0.12\,\mathrm{m/s}$.

Although all automatic design methods presented in this thesis use RM 1.1, different reference models, used by other methods of the AutoMoDe family, can be found in Hasselmann et al. (2018a).

3.3 Simulator

Within an automatic design method, we use computer simulations to optimize control software models. For this, all simulations are performed using ARGoS (Pinciroli et al. 2012), a simulator specifically conceived to simulate robot swarms. We used version 48 of ARGoS, along with the ARGoS-Epuck library (Garattoni et al. 2015), which provides software support for all extension boards. The library also enables the cross-compilation of the control software for the e-puck platform so that it can be ported to the robots without any modification (Ligot et al. 2017). The models of the e-puck's components have been conceived on the basis of real-world data sampled from the robot's sensors and actuators, according to best practices (Jakobi et al. 1995; Miglino et al. 1995).

The modular design of ARGoS allows control software generated by automatic design methods to be directly used in the simulator and with real robots. All

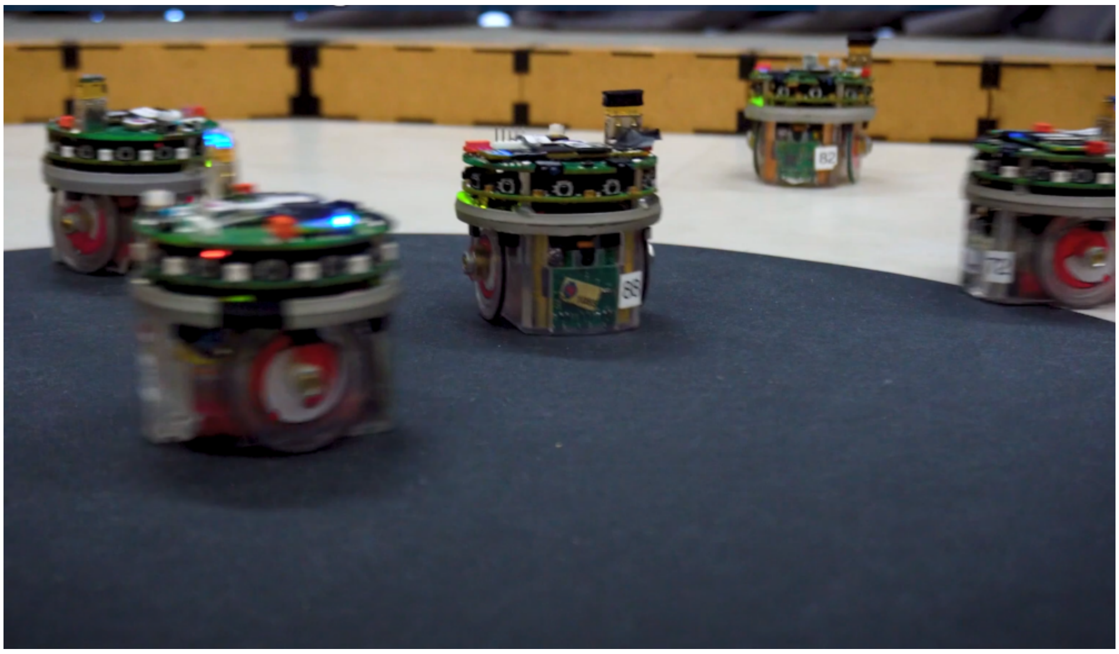


Image extracted from Supplementary Video C7

Figure 3.2: **Close shot of robots in the arena.** Five robots can be seen in the foreground; on the floor a black patch is placed, identical to the ones used during our experiments; in the background the wooden walls of the arena are placed.

automatic design methods presented in this thesis use the ARGoS simulator and their implementations were released as open-source software.

3.4 Arena

The robots operate in a dodecagonal arena of $4.91\,\mathrm{m}^2$ surrounded by walls and possibly containing obstacles. The floor is gray, with some regions that are white or black, depending on the mission to be performed. The gray floor is part of the vinyl flooring that covers the entire experimental room. Black and white areas are made out of high density paper and are glued to the floor. They are placed with special care in order to minimize friction in boundary areas between black or white areas, and the vinyl floor.

In some missions, a single light source, placed next to the arena, is on for the whole duration of an experimental run. This light source is filtered with a red gel to avoid overexposure of the overhead camera of the tracking system (see Section 3.4.1), but still be detectable by the robots' light sensors, which are particularly sensitive to the infra-red and the lower range of the visible spectrum.

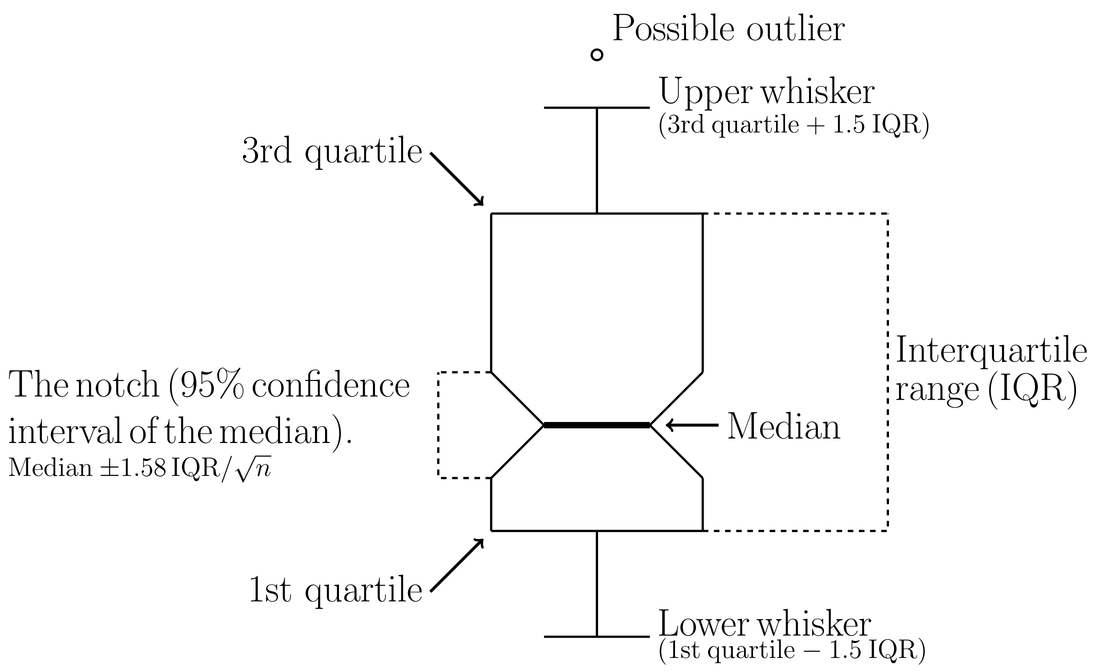


Figure 3.3: **Example of box-and-whiskers plot.** The thick horizontal line represents the median; the box extends to the upper and lower quartile; the upper/lower whiskers extends to the maximal/minimal observation that falls between the upper/lower quartile and 1.5 times the interquartile range; circles represent outliers, that is observations that fall beyond the whiskers. Notches on the box represent a 95% confidence interval on the median, and extend to $\pm 1.58\,\mathrm{IQR}/\sqrt{n}$, where IQR is the interquartile range and n is the number of observations. Circles outside of the whiskers represent outliers.

3.4.1 Tracking system

For all experiments presented in this thesis, the performance of the swarm was computed automatically using data provided by a tracking system (Stranieri et al. 2013) that registered the position of the robots throughout the duration of each experimental run. The position of the robots was not communicated to the robots themselves, which had only a local perception of the environment, coherently with the tenets of swarm robotics. The tracking system is based on an overhead camera and recognizes custom tags mounted on the robots—see Figure 3.1a.

3.5 Statistics

In this section, we summarize the statistical tools that we will use to analyze all the experiments in the thesis. Special statistical analyses realized for some experiments are detailed in the specific chapters where they are used.

In all experiments presented in this thesis, we test automatic design methods using real robots. Real robot experiments are costly and time consuming; it is therefore very important to determine the most efficient strategy for testing methods in order to minimize the variance in the estimation of the performance of automatic design methods on a given mission. If the design process is automatic and can run efficiently on a high-performance computing cluster, we can assume that the cost of running the design process is negligible compared to the one of performing real robot experiments. To assess the performance of a given method, we generate d control software instances, and we test each of these instances n times. Given a maximum of N experimental runs, we thus have $N = d \cdot n$. The theorem presented in Birattari (2020) states that given this maximum of N experimental runs, the experimental design that minimizes the variance of the estimate is the one where $d = N$, and $n = 1$ (for details and a formal proof, we refer the reader to Birattari (2004, 2020) and Cotorruelo et al. (2021)). This is why, in all experiments presented in this thesis, when generating d instances of control software for a given method, we assess its performance by testing each control software instance once in reality and once in simulation.

To represent the performances of the different automatic design methods that we study, we use notched box-and-whiskers plots. An example of box-and-whiskers plot is presented in Figure 3.3. In these plots, the thick horizontal line represents the median; the box extends to the upper and lower quartile; the upper/lower whiskers extend to the maximal/minimal observation that falls between the upper/lower quartile and 1.5 times the interquartile range; circles represent outliers, that is observations that fall beyond the whiskers. Notches on the box represent a 95% confidence interval on the median, and extend to $\pm 1.58\,\mathrm{IQR}/\sqrt{n}$, where IQR is the interquartile range and n is the number of observations. The difference between the medians of two boxes is significant with a confidence of at least 95% if the notches of the respective boxes do not overlap (Chambers et al. 1983).

We also execute Friedman tests (Conover 1999), which aggregates all results by ranking the performance of all methods across all missions. We present the results in a plot that represents an estimate of the expected rank of each method and the relative 95% confidence interval. The performance of two methods is significantly different with confidence of at least 95% if the corresponding intervals do not overlap.

3.6 AutoMoDe

The original methods presented in this thesis belong to the AutoMoDe family of methods. As introduced in Section 2.3.3, the reality gap is a central concept in automatic design and one of the main motivations for the creation of AutoMoDe. The inability for neuroevolution-based automatic design methods to satisfactorily cross the reality gap is conjectured to be the result of a too high representational power of neural networks.

The creation of AutoMoDe in Francesca et al. (2014b) is based on this conjecture. In AutoMoDe, robots are controlled by a modular control software architecture (e.g., a probabilistic finite-state machine) that is generated by assembling pre-defined software modules. The control software created by AutoMoDe features a lower representational power than neuroevolutionary methods because bias is injected in the design process by restricting the design space using modules. AutoMoDe is a framework, it needs to be *specialized* for a specific robotic platform. To implement a *specialization* of AutoMoDe, an expert needs to: (i) provide a set of modules and implement them for the specific robotic platform, (ii) define a control software architecture, and (iii) select an optimization algorithm to assemble modules in the target control software architecture. The optimization algorithm searches the space of possible instances of the control architecture by assembling the pre-defined modules. The modules are created in a mission-agnostic way, they are not created to solve a specific problem or tackle a specific mission. Rather, they are created based on the capabilities of the considered robotic platform and are used to tackle a wide range of missions within a class. The modules cannot be manually modified or adapted to a specific mission during the design process. All specializations of AutoMoDe thus belong to fully-automatic design methods, as defined in Section 2.3.2.

The definition and implementation of the modules are based on the reference model of the robotic platform that is considered. A reference model, as defined in Section 3.2, is a description of the capabilities of a robotic platform. It describes the interface between the software and the hardware of the robot. Since the creation of the modules is dependent on the reference model, the reference model also implicitly defines the class of missions that a robot swarm can tackle. For instance, a mission that requires robots to aggregate on a floor of a specific color cannot be completed by robots that cannot detect the floor's color.

There are multiple specializations of AutoMoDe; in the following sections, we will focus on `Vanilla` and `Chocolate`. `Vanilla` (Francesca et al. 2014b) is the first specialization to have been defined and a proof of concept. `Chocolate` (Francesca et al. 2015), an enhanced specialization based on `Vanilla`, is the most classical and the most studied AutoMoDe method (Francesca et al. 2015; Hasselmann and Birattari 2020; Hasselmann et al. 2021, 2018b; Kuckling et al. 2018, 2020a, 2019; Ligot and Birattari 2018, 2020; Ligot et al. 2020a,b; Mendiburu et al. 2022).

3.6.1 `Vanilla`

`Vanilla` demonstrates the core idea behind AutoMoDe. This method selects, combines, and fine-tunes pre-defined modules into control software in the form of probabilistic finite-state machines. `Vanilla` generates control software for the e-puck robot (see Section 3.1). In `Vanilla` the states of the probabilistic finite-state machines are low-level behavior modules and edges are condition modules. A low-level behavior module is an action that a robot performs and a condition module is a criterion for transitioning from the current low-level behavior to another one. The period of the control cycle in `Vanilla` is 100 ms. At every step, the low level behavior associated with the current state is executed, and the outgoing conditions of the current state are evaluated. If any of the conditions is evaluated as true, the current state is replaced by the one associated to the edge of the condition. The modules adopted in `Vanilla` are pre-defined based on the reference model of the e-puck robot (defined in Section 3.2). They are hand-crafted by a human expert and were tested both in simulation and using real robots.

There are six behavior modules in `Vanilla`:

i) **exploration**: the robot performs a random walk, while avoiding obstacles;

ii) **stop**: the robot stands still;

iii) **phototaxis**: the robot goes towards the light source, if perceived;

iv) **anti-phototaxis**: the robot goes in the opposite direction;

v) **attraction**: the robot goes towards its neighboring peers;

vi) **repulsion**: the robot goes in the opposite direction.

And six condition modules:

i) **black-floor**: transition state if the floor is black;

ii) **white-floor**: transition state if it is white;

iii) **gray-floor**: transition state if it is gray;

iv) **neighbor-count**: transition state if sufficiently many neighboring peers are perceived;

v) **inverted-neighbor-count**: transition state if they are sufficiently few;

vi) **fixed-probability**: transition state with a fixed probability.

These modules might have free parameters that can be tuned by the optimization algorithm during the design process.

The optimization algorithm adopted in `Vanilla` is the *F-race* algorithm (Birattari et al. 2002). During the design process it explores the control software instance search space and evaluates potential candidate probabilistic finite-state machines. To limit the size of the search space, the size of probabilistic finite-state machines is limited to four states, with a maximum of four outgoing transitions per state.

In Francesca et al. (2014a), the authors compared `Vanilla` to human experts. In this experiment, `Vanilla` and human experts designed software for a swarm of e-puck robots. Human experts outperformed `Vanilla` when constrained to using the same software modules as `Vanilla`, but `Vanilla` outperformed human experts when they were given complete freedom over the control software. The modules used by the constrained human experts and `Vanilla` are the same, and therefore the search space of solutions that they explore is also the same. Since, the only difference between the constrained humans and `Vanilla` is the way the search space of possible solutions is explored, the authors conjectured that adopting a better optimization algorithm could improve `Vanilla`. This conjecture lead to the creation of `Chocolate`.

3.6.2 `Chocolate`

`Chocolate` (Francesca et al. 2015) uses an improved version of the *F-race* algorithm, iterated *F-race* (or irace (López-Ibáñez et al. 2016)) as its optimization algorithm. In all other respects, `Chocolate` is identical to `Vanilla`: it selects, combines and fine-tunes the same behavior and condition modules into control software in the form of probabilistic finite-state machines.

The promising results of Chocolate in Francesca et al. (2015) motivated the creation and study of the many subsequent variants of Chocolate (see Section 2.4.3 for a full list).

3.7 EvoStick

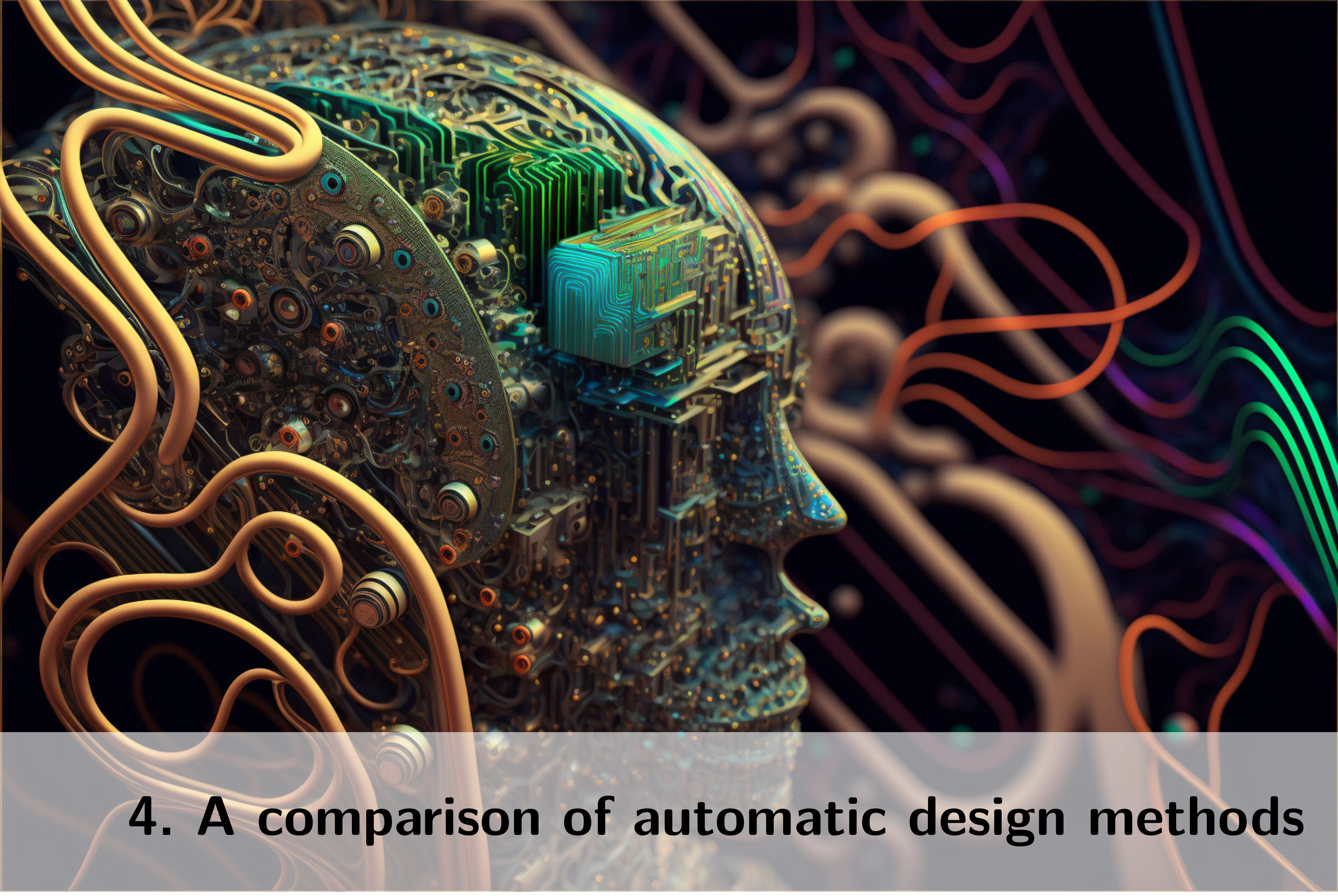
EvoStick is defined as a typical neuroevolutionary automatic design method. It was first introduced by Francesca et al. (2012) and defined as a "standard" neuroevolutionary method in Francesca et al. (2014a).

EvoStick generates control software in the form of neural networks based on reference model RM 1.1, defined in Section 3.2. Each robot is controlled by a fully-connected feed-forward neural network with no hidden layers.

The neural network has 24 input nodes, one bias node, and 2 output nodes. The 24 inputs are defined based on the reference model: 8 inputs for the proximity sensors, 8 inputs for the light sensors, 3 inputs for the ground sensors, 1 input encodes the number of neighboring robots perceived, and 4 inputs encode the projections of the range-and-bearing vector V on the four unit vectors that point at 45°, 135°, 225°, and 315° with respect to the head of the robot. The 2 output nodes are mapped to the velocities of the wheels. The activation function used on the output neurons is the sigmoid function. The neural network comprises a total of 50 parameters and the values of the synaptic weights range in $[-5; 5]$. The outputs of the neural network are evaluated at each period of the control cycle (100 ms).

EvoStick uses an evolutionary algorithm based on elitism and mutation. At the beginning of the process, a random population of 100 individuals is sampled, which are evaluated 10 times per generation. At each generation, the best 20 individuals are selected and passed unchanged to the following generation; random mutations are applied to these same 20 individuals to form the remaining 80 individuals of the new population.

EvoStick as been widely used as a yardstick to evaluate other design methods (Francesca et al. 2015, 2014b; Kuckling et al. 2018; Ligot and Birattari 2018, 2020; Ligot et al. 2020b). It is also used to this end in the various experiments presented in this thesis.



4. A comparison of automatic design methods

Neuroevolutionary robotics is an appealing approach to realizing collective behaviors for robot swarms (Brambilla et al. 2013; Nolfi 2021; Trianni 2008). In its typical application of offline automatic design, as seen in Chapter 2, each individual robot is controlled by a neural network that maps sensor readings to actuator commands. The parameters of the network, and possibly its topology, are obtained by optimizing a mission-specific performance measure via artificial evolution.

The neuroevolutionary approach appears to be appropriate in swarm robotics (Dorigo et al. 2014) because it bypasses the design problem: defining what the individual robots should do so that the desired collective behavior emerges from their interactions. This problem is particularly hard because of the complexity of the interactions between robots and the loosely-coupled nature of a robot swarm. More details about the design problem and automatic design methods can be found in Section 2.2.

In the literature on neuroevolutionary swarm robotics, empirical assessments and comparative analyses are rare (Francesca and Birattari 2016). In particular, to the best of our knowledge, no study has been published that compares any neuroevolutionary method on multiple missions and reports results obtained in experiments performed with real robots. Yet, there is a general understanding that, due to the reality gap, results obtained in simulation cannot be considered as a valid assessment of a neuroevolutionary method for the automatic offline design of robot swarms. Some recent results indicate that the reality gap is a relative problem with some design methods that are affected to a great extent, while others appear to be intrinsically more robust (Ligot and Birattari 2020), see Section 2.3.3 for details on the reality gap.

In this chapter, we present the results of an empirical study in which we assessed and compared some of the most advanced neuroevolutionary methods for the offline design of robot swarms.

The results indicate that all the neuroevolutionary methods under analysis are affected by the reality gap. This was possibly to be expected because of the aforementioned performance drop that has been often observed when moving from simulation to reality. What was not necessarily to be expected, because it had not emerged in any previous research, is that the extent to which the neuroevolutionary methods under analysis are affected by the reality gap is so conspicuous that all differences we observed in simulation disappeared in the real-robot experiments. Eventually, the control software they produced performed at most only marginally better than a trivial random walk behavior that we included in the study as a control.

We find compelling evidence that real-robot experiments are needed to reliably assess the performance of neuroevolutionary methods and that the robustness to the reality gap is the main issue to be addressed to advance the application of neuroevolution to robot swarms.

4.1 Experimental setup

We present here the different automatic design methods that are included in this study; details on the protocol and on the implementation of each method are given in Section 4.2.2. In this study we include:

i) Covariance Matrix Adaptation Evolutionary Strategy (`CMA-ES`) (Hansen and Ostermeier 2001), for generating both single- and multi-layer perceptrons. `CMA-ES` is widely considered as one of the most effective evolutionary algorithms available and is especially valued for its advanced search capabilities.

ii) Exponential Natural Evolution Strategies (`xNES`) (Glasmachers et al. 2010), for generating, also in this case, both single- and multi-layer perceptrons. `xNES` is closely related to `CMA-ES` and is sometimes preferred to the latter because it is considered to be more principled, as all the update rules needed for covariance matrix adaptation are derived from a single mechanism.

iii) Neuro-Evolution of Augmenting Topologies (`NEAT`) (Stanley and Miikkulainen 2002), initialized with either a fully-connected single-layer perceptron or a network in which input and output nodes are disconnected; in both cases, we studied two sets of hyper-parameters, one that allows the generation of recurrent networks and one that does not. `NEAT` is particularly valued for its capability to shape the network topology automatically.

iv) `EvoStick` (Francesca et al. 2014b), a straightforward implementation of the most basic ideas of the neuroevolutionary approach, this method is described in Section 3.7.

`EvoStick` is without any doubt less sophisticated and advanced than its competitors `CMA-ES`, `xNES`, and `NEAT`.

As baselines, we included in the study also:

i) `Chocolate` (Francesca et al. 2015), a design method that belongs to the AutoMoDe family (Francesca et al. 2014b). A full description of `Chocolate` is presented in Section 3.6. As already explained before, `Chocolate` is a method that generates probabilistic finite-state machine by assembling modules. `Chocolate` was explicitly conceived to be robust to the reality gap.

ii) `RandomWalk`, a trivial behavior in which robots move randomly in the environment. Contrary to all the other aforementioned methods comprised in the study, `RandomWalk` is not an optimization-based design method: no parameter/feature of the behavior is optimized. For a design method, being unable to improve over `RandomWalk` is to be considered as a major failure.

We tested the methods under analysis for their ability to generate control software for five missions, in a fully automatic way. The missions were formally specified via a performance measure to be maximized, and the methods under analysis were tested on them without undergoing any manually-applied mission-specific modification. The control software generated by the methods was automatically cross-compiled for the target platform and was deployed without undergoing any modification. All the methods designed software for the same target platform, used the same realistic physics-based simulator with the same simulation models, and were provided the same resources—notably, the same number of simulation runs to be performed within the design process. Also, all the methods adopted the same devices that are widely considered as the standard practice for reducing the impact of the reality gap and for increasing the robustness of the control software generated: the injection of noise in simulation models and the randomization of the initial conditions (Jakobi et al. 1995).

The five missions considered (Figure 4.1) are rather typical collective missions. Their level of complexity is comparable with the one of those that, at the moment of writing, are customarily studied in the automatic offline design of robot swarms. Admittedly, relatively more complex missions have been considered in the semi-automatic design literature—e.g., see Ferrante et al. (2015). This is understandable: semi-automatic design provides for human intervention within the design process and allows the designer to tailor the optimization process to the single specific mission considered. This eventually enables one to tackle relatively more complex missions that are out of reach for fully automatic design, at least at the current state of development of the field. As we have previously observed in Section 2.3.2, in (fully) automatic design, the challenge does not lie much in the complexity of each single mission, but rather in the fact that the design method must be able to produce control software for different missions without undergoing any modification.

We ran each method under analysis ten times on each of the five missions and we tested the control software they generated in real-robot experiments and also in simulation, so as to assess the impact of the reality gap on the different methods. A detailed description of all the methods, the robotic platform, the simulator, the five missions, the experimental design, and the statistics adopted is given in the following section.

4.2 Methods

In this section, we present the experimental setup and provide technical details on the different automatic design methods we compared in the study.

4.2.1 Protocol

We considered a swarm of 20 e-puck robots that operate in a dodecagonal arena of $4.91\,\mathrm{m}^2$ delimited by walls, as described in Section 3.4. The robotic platform used in this study is the e-puck robot. It is formally described, along with its reference model in Sections 3.1 and 3.2. A picture of the e-puck in the configuration adopted in the experiments is given in Figure 3.1a. We adopt the ARGoS simulator for all simulations in this study; details about the simulation setup are presented in Section 3.3. For each mission, each method was executed 10 times so as to obtain 10 instances of control software. Each design process was allowed the same budget of 200 000 simulation runs. The 10 instances of control software obtained were then tested once in simulation and once using robots. To avoid introducing any bias, robot experiments were randomized and no experimental run performed was discarded. The performance of the swarm was computed automatically using data provided by a tracking system; details about the tracking system are presented in Section 3.4.1. Videos of all the experimental runs were recorded using the camera of the tracking system and are available as Supplementary Videos C1 to C5 (Hasselmann and Birattari 2022).

Statistics

We used notched box-and-whiskers plots to represent the performance of the different methods. We refer to Section 3.5 for details on box-and-whiskers plots. To aggregate the performances of the different methods across all missions, we used the min-max normalization technique: for each mission, we normalized the performances obtained in reality and in simulation with the minimal and maximal performance obtained in reality across all design methods. As a result, the normalized performance in reality ranges between 0 and 1, but the normalized performance in simulation might exceed 1 if instances of control software performed better in simulation than the maximal performance value obtained in reality. We also executed a Friedman test (Conover 1999); details on this analysis are presented in Section 3.5.

4.2.2 Design methods under analysis

All neuroevolutionary methods under analysis generate neural networks with 2 output and 25 input nodes. The 2 outputs define the velocity of the wheels. Concerning the inputs, 1 is a bias node, 8 encode the readings of the proximity sensors, 8 those of the light sensors, 3 those of the ground sensors, 1 encodes the number of neighbors perceived, and 4 the projections of the range-and-bearing vector V on the four unit vectors that point at 45°, 135°, 225°, and 315° with respect to the head of the robot. Inputs and outputs are described by RM 1.1—see Section 3.1 and Figure 3.1b. The values of the synaptic weights range in $[-5; 5]$.

`CMA-ES-slp` is based on `CMA-ES` (Hansen and Ostermeier 2001), an evolutionary algorithm in which the population is described in statistical terms via the covariance matrix of its distribution—slp is the mnemonic for single-layer perceptron: the network generated has a fully-connected feed-forward topology without hidden layers. The population size λ and the initial step-size σ_0 are hyper-parameters of the optimization algorithm. We set $\lambda = 100$, a common choice in the literature (Francesca and Birattari 2016); and $\sigma_0 = 5$, that is, half the width of the parameter range, the initial population will therefore cover the entire search space—the same choice was made also in several other studies (Auger and Hansen 2005; Lunacek and Whitley 2006; Pagliuca and Nolfi 2019).

`CMA-ES-mlp` is derived from `CMA-ES-slp` and differs from it only in the topology of the network, which is here a fully-connected feed-forward neural network with one hidden layer composed of 14 nodes, including a bias node. The size of the hidden layer is the average of the number of nodes in the input and output layers, as recommended by Heaton (2008)—mlp is the mnemonic for multi-layer perceptron: the input nodes are initially all connected to the hidden nodes, which are in turn all connected to the output.

`xNES-slp` is based on `xNES` (Glasmachers et al. 2010), an evolutionary algorithm similar to `CMA-ES`, but in which the updates rules are defined in a principled way—that is, they are all derived from the principle of natural gradient ascent. The hyper-parameters and their values are the same as in `CMA-ES-slp`. Also the network topology is the same one adopted in `CMA-ES-slp`.

`xNES-mlp` is derived from `xNES-slp` and differs from it only in the network topology, which is here a fully-connected feed-forward neural network with one hidden layer of 14 nodes, including 1 bias node—the same topology adopted in `CMA-ES-mlp`.

`NEAT-A-slp` is based on `NEAT` (Stanley and Miikkulainen 2002), a neuroevolutionary algorithm that optimizes both the weights and the topology of the neural network. The design process is initialized with a fully connected feed-forward neural network with no hidden layers—slp is the mnemonic for single layer perceptron: the input nodes are initially all connected to the output. The hyper-parameters of `NEAT-A-slp` are those originally published in Stanley and Miikkulainen (2002) and recommended by the authors. They are labelled as pole2_markov in the original software package released by the authors.

`NEAT-A-nl` is derived from `NEAT-A-slp` and differs from it only in the initialization of the design process, which is here a disconnected network—nl is the mnemonic for no link: the input nodes are initially disconnected from the output.

`NEAT-B-slp` is similar to `NEAT-A-slp` and differs from it only in the value of a few hyper-parameters. The hyper-parameters of `NEAT-B-slp` are those labelled as params256 in the original software package published in Stanley and Miikkulainen (2002). The differences between set A (presented above) and B are that set B has a higher compatibility coefficient (leading to less species creation), set A penalizes old species whereas set B does not, and most importantly that set B can generate recurrent networks.

`NEAT-B-nl` is derived from `NEAT-B-slp` and differs from it only in the initialization of the design process, which is here a disconnected network: the input nodes are initially disconnected from the output.

`EvoStick` is a rather standard neuroevolutionary robotics method. A full description of `EvoStick` is presented in Section 3.7.

`Chocolate` belongs to the AutoMoDe (Francesca et al. 2014b) family of design methods. A full description of `Chocolate` is presented in Section 3.6.

`RandomWalk` is a ballistic-motion random walk: the robot moves straight until it encounters an obstacle. When this happens, the robot rotates on itself for a random number of timesteps and resumes it straight motion, if the path is clear; otherwise, it rotates for another random number of timesteps. This sequence is repeated indefinitely. `RandomWalk` is not an automatic design method as no parameter is tuned. It is included in the study as an expected lower bound on the performance.

4.2.3 Missions

XOR-Aggregation: the robots must choose one of two black areas and aggregate on it. The size of the black areas and their positions are given in Figure 4.1. The

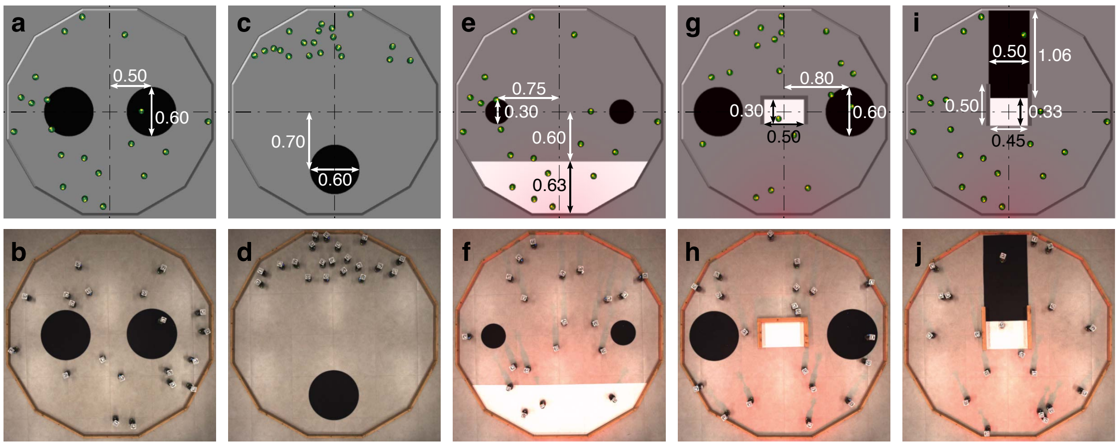


Figure first published in Hasselmann et al. (2021)

Figure 4.1: **Arenas for the five missions.** **a**, XOR-Aggregation, simulation; **b**, real robots. **c**, Homing, simulation; **d**, real robots. **e**, Foraging, simulation; **f**, real robots. **g**, Shelter, simulation; **h**, real robots. **i**, Directional-Gate, simulation; **j**, real robots. The 20 robots operate in a dodecagonal arena of 4.91 m^2, the red glow in **e**, **f**, **g**, **h**, **i**, and **j** indicates the presence of a light source at the bottom side of the arena. Dimensions (in meters) of the elements present in the arenas are given in **a**, **c**, **e**, **g**, and **i**. Details on the experimental setup are provided in Section 4.2 and videos of the robot experiments are available as Supplementary Videos C1 to C5.

performance of the swarm is measured by the following objective function:

$$F_{\mathrm{a}} = \sum_{t=1}^{T}\sum_{i=1}^{N} I_i(t); \qquad I_i(t) = \begin{cases} 1, & \text{if robot } i \text{ is in the area with the most robots;} \\ 0, & \text{otherwise.} \end{cases} \tag{4.1}$$

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

Homing: the robots start in the upper part of the arena and must aggregate on the black area situated at the bottom. The size of the black area and its position are given in Figure 4.1. The performance of the swarm is measured by the following objective function:

$$F_{\mathrm{h}} = \sum_{i=1}^{N} I_i(T); \qquad I_i(T) = \begin{cases} 1, & \text{if robot } i \text{ is in the black area at time } T; \\ 0, & \text{otherwise.} \end{cases} \tag{4.2}$$

$T = 120\,\mathrm{s}$ is the duration of the experimental run and $N = 20$ is the size of the swarm.

Foraging: the robots must find one of the black areas, which represent food sources, and go back to the white one, which represents the nest. A light source is positioned behind the nest. The size of the areas of interest and their positions are given in Figure 4.1. The performance of the swarm is measured by the following objective function:

$$F_{\mathrm{f}} = K, \tag{4.3}$$

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

Shelter: the robots must aggregate in the shelter, a rectangular white area positioned in the center of the arena and surrounded by walls on three sides. 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, next to the shelter. These areas do not have any predefined purpose/role in the definition of the mission: they are noise-features of the environment. The size of the shelter, the one of the black areas, and their positions are given in Figure 4.1. The performance of the swarm is measured by the following objective function:

$$F_\mathrm{s} = \sum_{t=1}^{T} \sum_{i=1}^{N} I_i(t); \qquad I_i(t) = \begin{cases} 1, & \text{if robot } i \text{ is in the shelter;} \\ 0, & \text{otherwise.} \end{cases} \tag{4.4}$$

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

Directional-Gate: the robots must traverse the gate, which is positioned in the center of the arena. They must do so from North to South. The gate is identified by white ground and the robots can follow a black corridor to reach it. The size of the gate, the one of the corridor, and their positions are given in Figure 4.1. The performance of the swarm is measured by the following objective function:

$$F_\mathrm{g} = K - K', \tag{4.5}$$

where K is the number of times robots traverse the gate in the correct sense and K' is the number of times they traverse it in the wrong one. The duration of an experimental run is $T = 120\,\mathrm{s}$ and the swarm size is $N = 20$.

4.3 Results

The results (Figure 4.2 and 4.3) show that, on the missions considered in the study, all the neuroevolutionary methods under analysis experienced a major drop in performance because of the reality gap. For each mission and method, the empirical distributions of all the data gathered in simulation and real-robot experiments are given in Figure 4.4.

When evaluated in simulation, the control software produced by the neuroevolutionary methods under analysis generally performed well, and comparably with the one of `Chocolate`; in some cases, even better. All methods under analysis performed significantly better than `RandomWalk` (Figure 4.2a). Results were different when the control software was evaluated in real-robot experiments. The performance of all design methods dropped due to the reality gap, as it is often the case. Only the performance of `RandomWalk` remained substantially stable—this is because, as observed above, `RandomWalk` is not a design method: no optimization process is involved and therefore no overfitting can occur. All the neuroevolutionary methods experienced a large drop, whereas the one of `Chocolate` is relatively smaller. Also

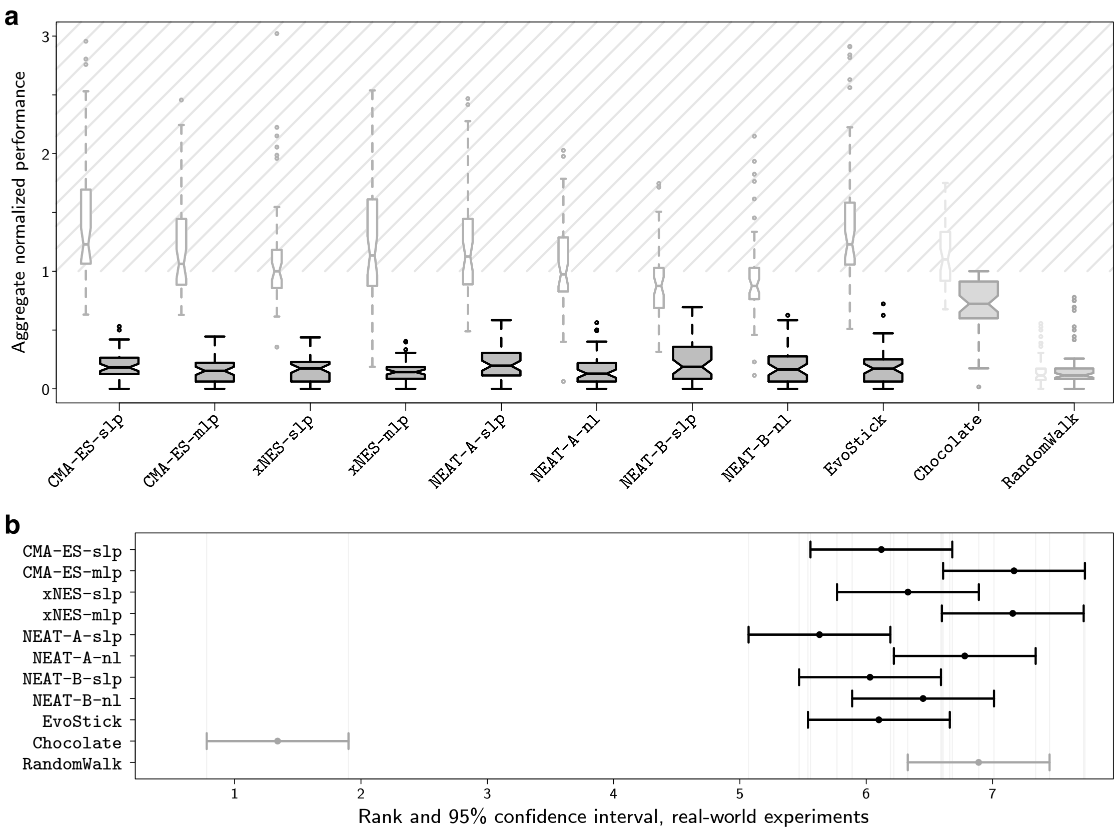


Figure first published in Hasselmann et al. (2021)

Figure 4.2: **Aggregated results. a**, aggregate performance in simulation (white narrow boxes) and in reality (gray wide boxes) across the five missions considered, represented by notched box-and-whisker 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 methods differ significantly, with a confidence of at least 95%. Graphical conventions adopted in box-and-whisker plots are described in Section 3.5. Prior to the aggregation and for each missions, the results are normalized between the lowest and highest performance observed in reality by any of the design methods. As a result, the normalized performance in reality ranges between 0 and 1, but the one in simulation might exceed 1 (shadowed area). Indeed, in many cases, the performance observed in simulation exceeded the best one observed in the real-robot experiments. The performance of `Chocolate` and `RandomWalk`, the two methods included in the study as yardsticks, is grayed out so as to focus the attention of the reader to the neuroevolutionary methods under analysis. **b**, Friedman rank-sum test on the performance in reality: expected rank and 95% confidence interval. If two segments do not overlap, the rank of the corresponding methods differ significantly, with a confidence of at least 95%. Also here, the performance of `Chocolate` and `RandomWalk` is grayed out to focus the attention to the neuroevolutionary methods. The videos of all robot experiments are available as Supplementary Videos C1 to C5.

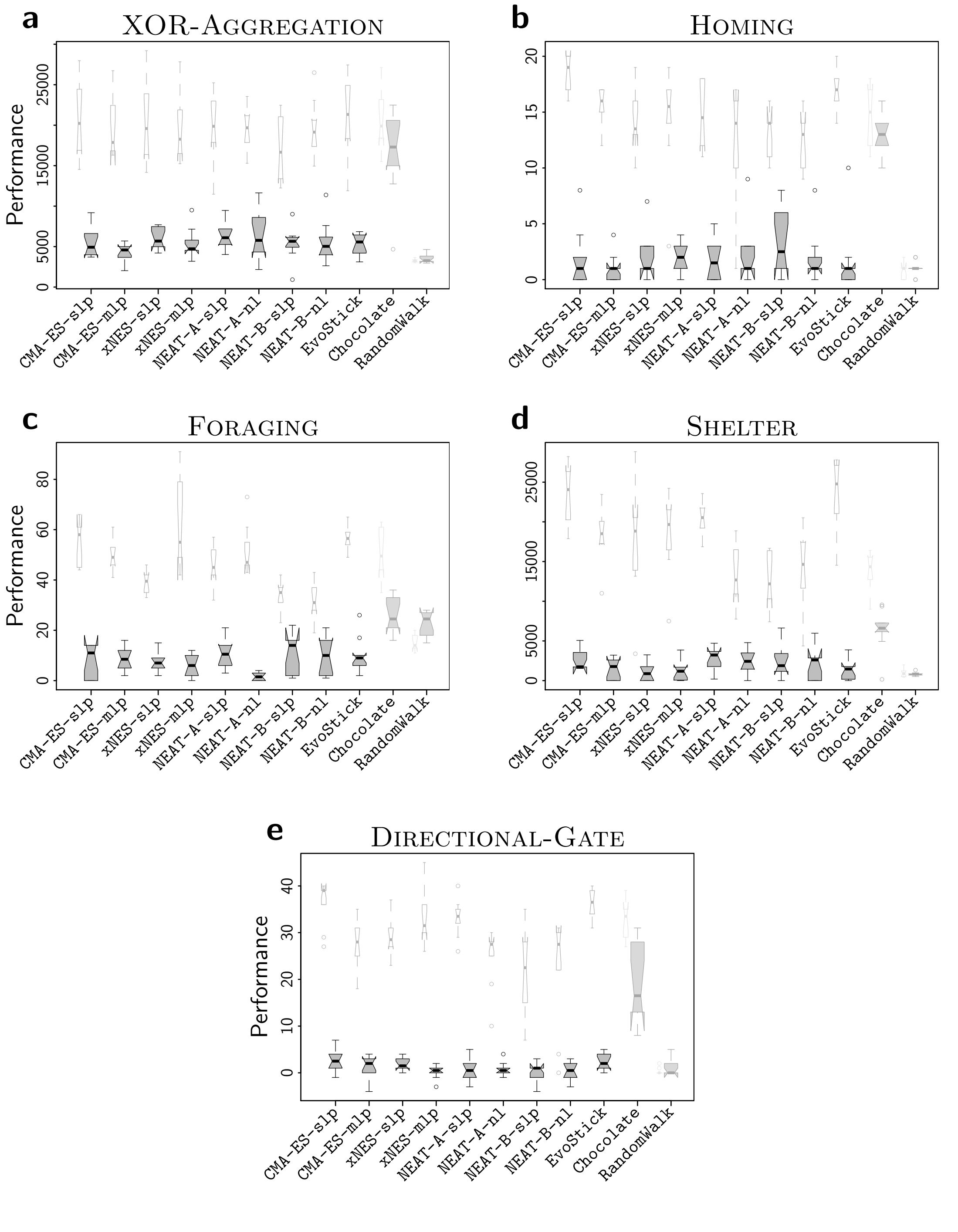


Figure 4.3: **Results per missions.** Performance obtained in simulation (white narrow boxes) and in reality (gray wide boxes) in all five missions, represented by notched box-and-whiskers plots, where the notches represent the 95% confidence interval on the median. If notches on different boxes do not overlap, the medians of the corresponding methods differ significantly, with a confidence of at least 95%. Graphical conventions adopted in box-and-whisker plots are described in Section 3.5. The performance of `Chocolate` and `RandomWalk` is grayed out so as to focus the attention of the reader to the neuroevolutionary methods under analysis. The videos of all robot experiments are available as Supplementary Video C1 to C5.

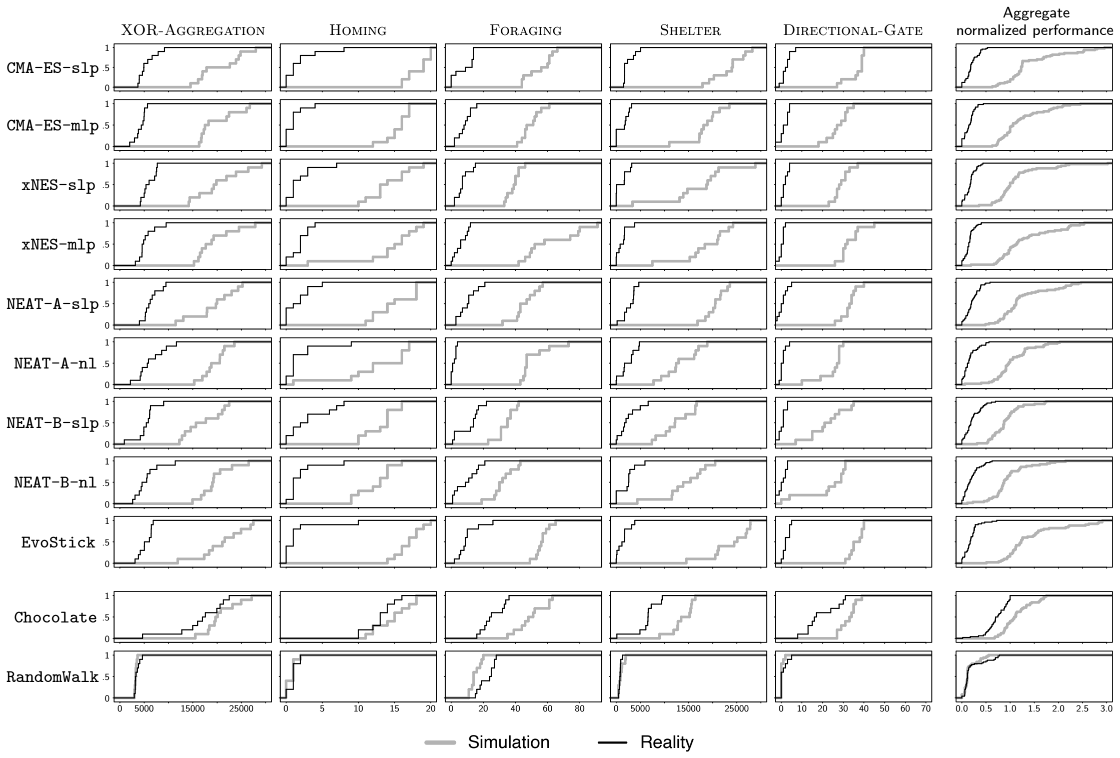


Figure first published in Hasselmann et al. (2021)

Figure 4.4: **Distributions.** Empirical distribution of the performance of the control software generated by each method under analysis on each of the five mission considered. The last column displays the normalized performance of each method, aggregated across the five missions. Aggregation is performed using the min-max normalization technique described in Section 4.2.1. The black line represents the empirical distribution of the performance observed in real-robot experiments; the gray one, the one obtained in simulation.

from a qualitative point of view, the control software produced by the neuroevolutionary methods displayed different behaviors in simulation and reality, whereas the one produced by `Chocolate` behaved similarly in simulation and reality, and even more so `RandomWalk`—see Figure 4.5 and Supplementary Video C6. Eventually, all neuroevolutionary methods performed significantly worse than `Chocolate` and their results were only marginally better than those of `RandomWalk` (Figure 4.2a).

It is important to notice that the reality gap faced by the methods under analysis is the same: they all adopt the same simulator and design control software for the same platform. Yet, the extent to which the methods were affected is different. As it has been already observed (Ligot and Birattari 2020), the reality gap problem is a relative problem, with some methods being more heavily affected and others being intrinsically more robust.

A further observation is that the five missions can be accomplished to a satisfactory extent under the experimental conditions considered. Specifically, they can be accomplished by the platform adopted and by a robot swarm of the given size. Moreover, control software to accomplish these missions can be produced automatically using the available resources: the simulator provides a reasonably faithful representation of reality (albeit not perfect, as no simulation does) and the number of simulation runs allotted to each design process was appropriate. This is shown by the satisfactory results obtained in the robot experiments by `Chocolate`—see Supplementary Videos C1 to C5. Concerning the neuroevolutionary methods, the fact that the number of simulation runs allotted was sufficiently large is confirmed by the satisfactory performance obtained in simulation by the control software they produced.

The performance of the different neuroevolutionary methods under analysis is similar. The few differences that can be observed between the results obtained in simulation disappeared when the control software generated was tested in real-robot experiments. A remarkable fact is that, on the missions considered, the more advanced methods—that is, `CMA-ES`, `xNES`, and `NEAT`—did not yield any relevant improvement over `EvoStick`, the straightforward implementation of the neuroevolutionary approach. This holds true both for simulation and robot experiments. The data indicate that, at least on the missions considered, neither the effective search of `CMA-ES` and `xNES`, nor the advanced abilities of `NEAT` to shape the topology of networks have the potential to improve the performance of the neuroevolutionary approach. The real issue to be addressed is the robustness to the reality gap.

In the missions considered, the main discrepancies between the behaviors

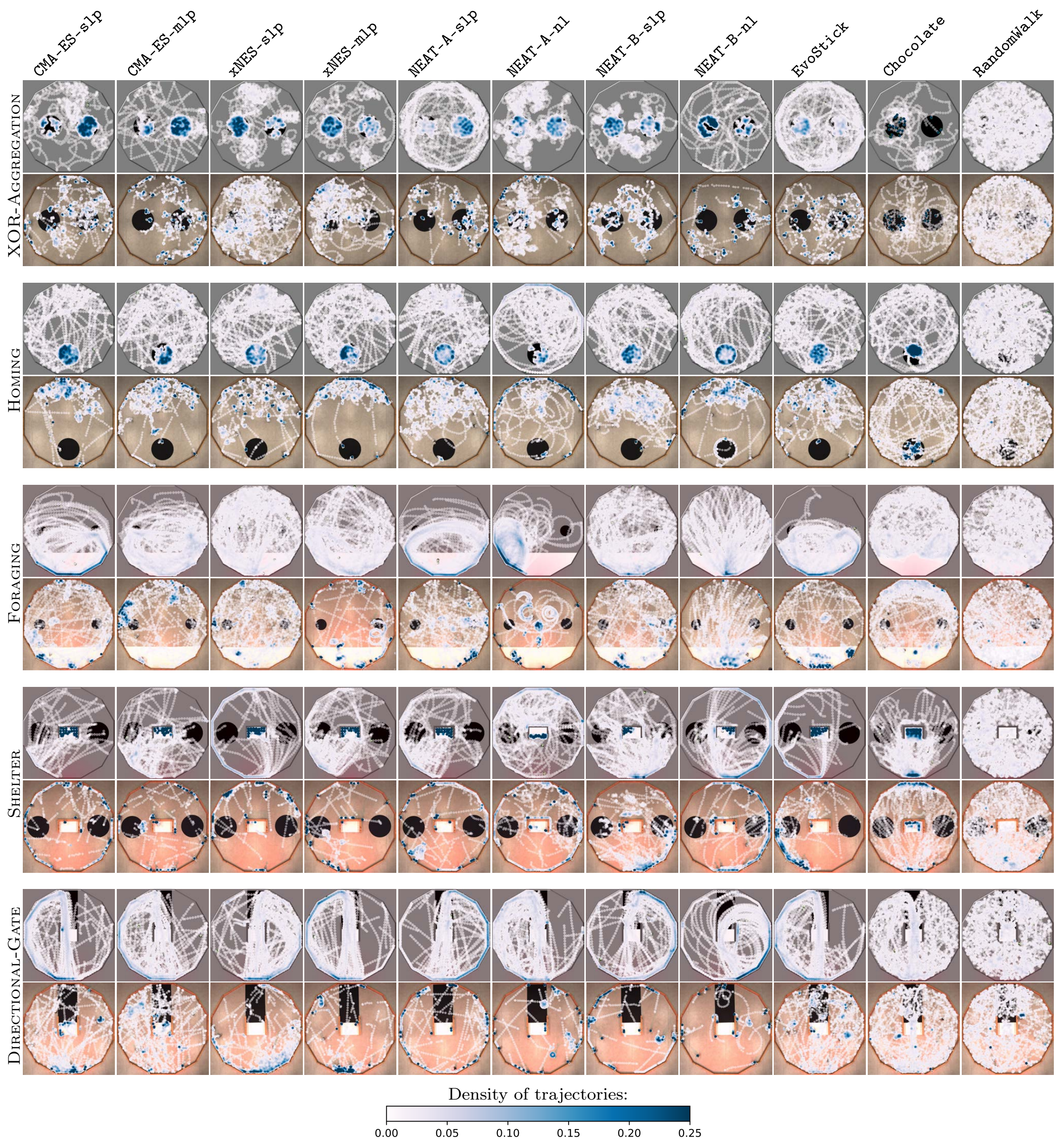


Figure first published in Hasselmann et al. (2021)

Figure 4.5: **Trajectories of the robots throughout the entire median runs.** For each method on each mission, we report the execution in simulation (top row for each mission) and reality (bottom row for each mission) of the instance of control software that obtained a median performance in reality, out of the instances produced by that method for that mission. The color of a spot represents the amount of time a robot spent on that spot during the execution. If a robot were to stay on a spot for more than a quarter of the entire execution, the color of that spot would be dark blue (value 0.25 in the color scale). The figure indicates that the control software produced by the evolutionary approaches cover the space differently in simulation and reality: in reality, the robots tend to form clusters, mostly against the walls. Differences between simulation and reality are less pronounced for `Chocolate` and barely noticeable for `RandomWalk`. For each mission and each method, a direct comparison of the behavior in simulation and reality is available in Supplementary Video C6.

observed in simulation and reality concern the way in which robots cover the space. Robots tend to cluster (mostly against the walls) in reality more than they do in simulation. This is likely due to the fact that friction between robots and between robots and walls is not modeled in a sufficiently accurate way: in simulation, robots slip against each other and against the walls; while in reality, they remain more easily stuck. Another discrepancy we observed concerns the shape of the trajectories. In simulation, all robots can be observed to move orderly, following circular trajectories; in reality, some robots display similar trajectories while those of others tend to be squashed and irregular. This is likely due to the fact that, although the swarm is in principle homogeneous and it is simulated as such, the real robots tend to differ slightly one from the other in their sensors and actuators. As a result, the real robots fail to display the ordered and cohesive collective motion that can be observed in simulation. The issue is particularly noticeable in FORAGING and DIRECTIONAL-GATE. Both discrepancies (clustering and irregular trajectories) are accrued by density: the more the robots converge to a same restricted area, the more the behavior observed in reality differs from the simulated one. Although the discrepancies observed can be used to improve the simulator for future applications, we do not think that they provide information that can contribute to address the reality gap problem in a universally valid way. It should be noted that reducing the differences between simulation and reality on the basis of the observation of control software produced by specific methods on specific missions could lead to ad hoc solutions that do not necessarily generalize to other methods, missions, platforms, environment, and scenarios (Bongard 2013; Silva et al. 2016; Watson et al. 2002). Also, reducing the differences between simulation and reality a posteriori—that is, after observing that the control software produced in simulation does not behave satisfactorily in reality—is not compatible with the spirit and purpose of automatic offline design as it requires human intervention and assessments on real robots.

The satisfactory results obtained in the real-robot experiments by `Chocolate`—both in absolute terms and relatively to those obtained by the neuroevolutionary methods under analysis—corroborate the validity of the original idea that motivated the definition of AutoMoDe and the development of `Chocolate` itself. Indeed, the results corroborate the original conjecture of Francesca et al. (2014b) that injecting bias, and therefore restricting the design space, might be an effective way to increase the robustness to the reality gap. It is our contention that, in the experiment presented above, `Chocolate` crossed the reality gap successfully, compared to neuroevolutionary methods, because of its relatively small design space. By the same token, we contend that, compared to `Chocolate`, the neuroevolutionary methods under analysis failed to cross the reality gap successfully because, in their definition, no explicit attention was made to restrict the size of the design space.

4.4 Ideas for future research

Supported by the results presented above, we contend that, to advance the application of neuroevolution to the automatic offline design of collective behaviours for robot swarms, the research community should focus on addressing the reality gap problem. A number of ideas have already been proposed in the literature and belong into two distinct approaches (Ligot and Birattari 2020): 1) increase the accuracy of simulators as much as possible; 2) conceive design methods that are intrinsically robust to the reality gap. The two approaches are not mutually exclusive and can profitably coexist within the same design method. We definitely agree that simulation accuracy must be pursued. Yet, as simulation models will never be perfect and the risk of overfitting cannot be eliminated altogether, the quest for accurate simulators does not eliminate the need for robust methods. It is therefore our contention that future research should aim at increasing the intrinsic robustness of design methods. Making the optimization algorithms more effective or enhancing their ability to automatically shape the topology of the networks appears to be a secondary concern, at least in this phase of the development of the field.

Although most of the ideas proposed to address the reality gap problem do not fit the framework of the automatic offline design considered here and, to the best of our knowledge, have never been applied in swarm robotics, they could be possibly adapted or could be the starting point to develop original methods for enhancing the robustness of neuroevolutionary robotics. For example, in Koos et al. (2013), the authors proposed a method that builds and updates a model of the differences between the performance in simulation and reality. The model is used to constrain the design process to generate only control software whose real-world performance is expected to be correctly predicted by the simulator. The method requires periodic runs with the robots during the design process, which cannot therefore rely on simulation only. In Floreano and Mondada (1996), the authors proposed a method that originally blends ideas from offline and online design: the update rules of the neurons and their parameters are defined offline in simulation; the synaptic weights are subsequently adapted online. Also the idea underlying the development of `Chocolate`—that is, restricting the design space to reduce the risk of overfitting—

could be possibly applied in the context of neuroevolutionary robotics. Indeed, restricting the design space is effectively a form of regularization and a variety of regularization techniques that could be ported to neuroevolutionary robotics have already been described in the neural network literature (Goodfellow et al. 2016; Hastie et al. 2009). For example, although further research is needed to develop a reliable method, previous results (Birattari et al. 2016) indicate that a popular regularization technique known as early stopping (Caruana et al. 2000; Morgan and Bourlard 1989; Prechelt 2012; Raskutti et al. 2014) has the potential to increase the robustness to the reality gap of neuroevolutionary methods for the automatic design of robot swarms. In the light of the results of our experiments, we are convinced that the adoption of an appropriate regularization technique is the most promising direction to be explored in the development of the neuroevolutionary approach to the automatic offline design of robot swarms.

4.5 Discussion

In this chapter, we presented the most extensive comparative analysis of offline automatic design methods currently available in the literature. We compared ten different design methods and one random walk, on five different swarm robotics missions. Nine design methods are neuroevolutionary ones, and one method is `Chocolate`, a modular method.

Our main conclusions are:

i) Experiments with real robots are of paramount importance to have a correct picture: simulation gives a falsely overoptimistic assessment of the methods under analysis.

ii) The advanced features of `CMA-ES`, `xNES`, and `NEAT` do not appear to provide any practical advantage over the straightforward `EvoStick`. In any case, possible (minor) differences observed between the methods in simulation disappear when the control software is ported to the robots.

iii) The real issue is the lack of robustness to the reality gap of the currently available neuroevolutionary methods for the automatic offline design of robot swarms. This is the issue on which, in our opinion, future research should focus.

iv) `Chocolate` experienced a much smaller performance drop than the neuroevolutionary methods under analysis. This confirms the validity of the idea that

> restricting the control software to be a combination of low-level, simple behaviors yields better results in reality than the traditional neuroevolutionary approach. Our conjecture is that this smaller performance drop can be ascribed to a reduced risk of overfitting that derives from restricting the design space, which is effectively a regularization technique. It is our contention that this technique—or another of the several regularization techniques previously described in the neural network literature—can be ported to neuroevolutionary robotics and is a promising avenue to address the reality gap problem in the application of neuroevolution to the automatic offline design of control software for robot swarms.

Neuroevolution is an appealing approach due to the simplicity of its implementation, but is not designed to be robust to the reality gap. At least not in the way it is implemented in the methods we have presented. Modular methods, and in particular `Chocolate`, require human expertise in the creation of its modules. The automatic design methods presented in Chapters 5 and 6 attempt to conjugate neuroevolution and modular design to enhance the robustness to the reality gap of methods using neuroevolution.

5. AutoMoDe-Arlequin

In this chapter, we make a first attempt to reduce the amount of expert knowledge required for the implementation or use of offline automatic design methods.

As discussed above, AutoMoDe is a family of modular automatic design methods that combines hand-crafted behavior and condition modules. `Chocolate`, the most studied instance of AutoMoDe, combines modules into probabilistic finite-state machines. In Chapter 4, we observed that neuroevolutionary methods outperform AutoMoDe when evaluated in simulation, but AutoMoDe outperforms them in reality. In popular instances of AutoMoDe, the modules and conditions are hand-coded a priori. During the design phase, the optimization algorithm combines these modules while maximizing a given objective function representing the performance of the swarm in a given mission. The optimization algorithm also fine-tunes the parameters of the modules. The downside of such method is that creating modules requires expert knowledge at the level of the robotic platform. The method is fully-automatic, as defined in Section 2.3.2, but the implementation of the instance of AutoMoDe requires expertise. This expertise is required for the creation of new modules in order to implement and test them. In `Chocolate`, modules were tested in simulation and in reality before attempting any design, in order to reduce the effects of the reality gap on the generated control software. In this chapter, we present `Arlequin`, a novel automatic design method of the AutoMoDe family. This method automatically combines behavior modules in the form of neural networks that were generated using neuroevolution. The objective behind the creation of `Arlequin` is to create a method that requires less human expertise during implementation and possibly when ported to a different robotic platform.

The possible shortcoming of generating the modules using neuroevolution is the introduction of a source of overfitting in the creation of the modules. This could cancel the benefits of modularity. Indeed, overfitting can occur at different levels: (i) during the implementation of the modules, (ii) during the fine tuning of the modules, or (iii) during the combination of the modules. The overfitting that occurs during the implementation of the modules is mission-agnostic: it does not depend on any specific mission, as modules are created during the implementation of the method and for a class of missions. Therefore, at the module level, a given behavior that overfits the simulation exhibits poor performances when transferred to the real robots. The overfitting that, on the other hand, occurs during the fine tuning and the combination of the modules, is mission-specific: it depends on the specific mission at hand; the combination and the fine tuning is carried out by the optimization algorithm that optimizes for an objective function that is specific to a mission. These sources of overfitting contribute to the effect of the reality gap experienced by the control software generated by the automatic design method. It is challenging to isolate these sources. It is also difficult to say which of the two contributes the most to the effect of the reality gap. So far, studies on AutoMoDe methods show that manually designing modules in simulation and validating them on real robots limits the performance drop due to the reality gap caused at the module level. With `Arlequin`, we investigate whether the principles of modularity still apply when the behavior modules are created using neuroevolution. We explore whether injecting bias by merely constraining the control software to use pre-defined modules is sufficient to cross the reality gap satisfactorily.

The results suggest that `Arlequin` suffers less from the reality gap than `Evo-Stick`. The results also suggest that there is room for improvement, as `Arlequin` suffers from a larger drop than `Chocolate`. Finally, even though `Arlequin` requires less human expertise to operate and to be ported from one robotic platform to another than existing modular methods, we did not completely exclude human intervention from its implementation. In fact, the generation of the behavioral modules required the elaboration of objective functions, which can be difficult (Floreano and Urzelai 2000). To completely eliminate the need of human expertise, it is necessary to investigate various design methodologies for the method and the generation of the modules. One possible methodology is presented in Chapter 6.

5.1 Modular design using neural networks

In this section, we present details about `Arlequin`. `Arlequin` generates control software for the e-puck robot, equipped with the same sensors and extensions board as described in Section 3.1. We consider the reference model RM1.1 presented in Figure 3.1 and Section 3.2. For the convenience of the reader, we list below the fundamental capabilities supported by RM1.1. RM1.1 gives the generated control software the ability to: detect the color of the ground under the robot (black, grey, or white) using the ground sensor; detect the presence of nearby obstacles and robots using the proximity sensor; detect the intensity of the ambient light using the light sensor; detect the direction of neighboring robots in a range of 0.7 m using the range-and-bearing module; and steer the robot using the wheels.

Similarly to `Chocolate`, `Arlequin` generates control software from a set of six conditions and six behaviors. Behaviors are software modules that describe actions performed by the robot. Conditions are software modules that describe events based on sensors of the robot that trigger a change of behavior. The modules are used to create probabilistic finite-state machines; behaviors are used as states of the probabilistic finite-state machines, whereas conditions are used as transitions between states. Thus, when executing the control software, a given behavior is run as long as a no condition is triggered. Once a condition is triggered, the state switches from the current state to the next state corresponding to the edge of the transition that has been triggered.

`Arlequin` is similar to `Chocolate` on various aspects (details about `Chocolate` can be found in Section 3.6). The two methods adopt irace as their optimization algorithm and generate probabilistic finite-state machines. The same constraints are imposed to the structure of the probabilistic finite-state machines: they are composed of a maximum of four states and each state can have a maximum of four outgoing transitions. The condition modules used in `Arlequin` are the same hand-crafted conditions as the ones of `Chocolate` (see Section 5.1.1 for details about condition modules).

The main difference between `Arlequin` and `Chocolate` lies in the definition of the behavior modules. `Chocolate` uses pre-defined hand-crafted behavior modules, whereas `Arlequin` uses automatically generated behavior modules. The behavior

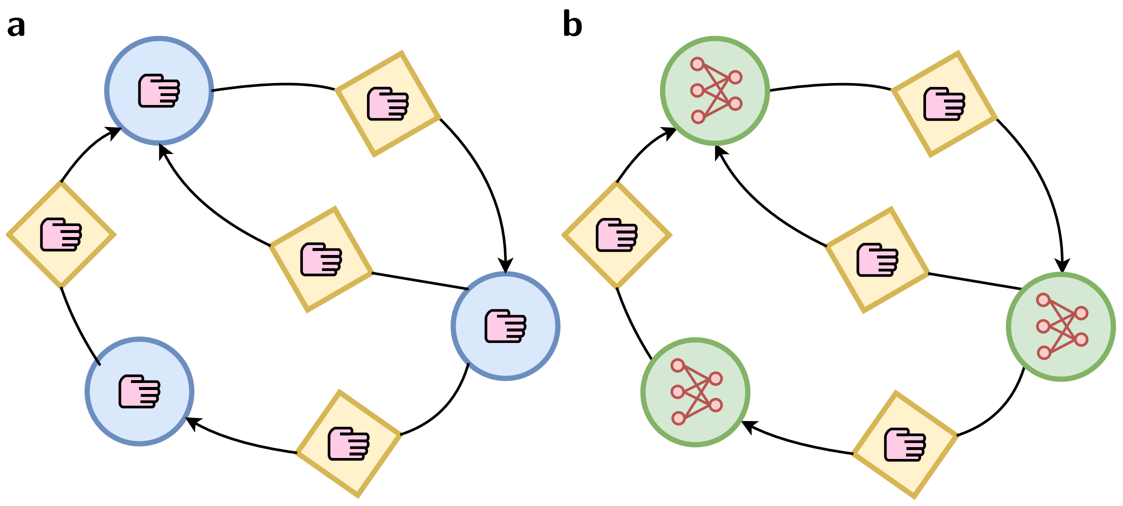


Figure 5.1: **Illustration of `Chocolate` and `Arlequin` probabilistic finite state machine structures.** The hand icon represents a hand-crafted module, the neural network icon represents an automatically generated module in the form of a neural network. **a**, In `Chocolate` both the behaviors and conditions are hand-crafted; **b**, In `Arlequin`, the conditions are hand-crafted but the behaviors are automatically generated neural networks.

modules of `Arlequin` are created using `EvoStick`, a classical neuroevolutionary design method. An illustration of the difference in the structure of probabilistic finite-state machines between `Chocolate` and `Arlequin` is presented in Figure 5.1. `EvoStick` is the rather simple neuroevolutionary design method that was already used in Chapter 4, and that is described in Section 3.7. The behavior modules that were created for `Arlequin` are based on the ones of `Chocolate`. Six objective functions were created based on the six different behavior modules of `Chocolate`. These objective functions are designed to yield high scores if a satisfactory behavior is obtained. They are used as performance metrics to optimize the control software for each module in `EvoStick`. It is important to understand that we are describing, here, the method that was used to generate the behavior modules, not the final control software.

`EvoStick` is used here in a mission-agnostic way, that is, the objective functions that are created describe the behaviors that were previously manually designed in `Chocolate` (details on the behavior modules and objective functions can be found is Section 5.1.2). `EvoStick` is also used as an automatic design method that directly generates software for the specific mission at hand. This automatic design method is used in this experiment and we compare the results of the two approaches (see Section 5.2.2 for details about `EvoStick`).

For the creation of the modules, we generated control software for a swarm of 20 e-puck robots during runs of 120 s. The allocated designed budget is 20 000: starting from a population of 100 individuals each evaluated 10 times per generation, we run 20 generations. For every objective function, or behavior module, we generate 10 instances of control software. Each of these 10 instances of control software is evaluated 20 times in simulation under various initial conditions. The instance with the highest average performance among the 10 is then selected. The selected instances are used as behavior modules in `Arlequin`. The objective functions used to generate the behaviors, the descriptions of the behaviors and the conditions are given in Sections 5.1.2 and 5.1.1. The behavior modules are generated once and for all, and once selected, they are used "as-is", in the automatic design process. The method thus qualifies as (fully-)automatic, the modules are created in a mission-agnostic way and used in the automatic design process without any per-mission modification.

5.1.1 Conditions

The condition modules are identical to the ones of `Chocolate`, for the convenience of the reader, we repeat their descriptions hereafter.

Black-floor. Evaluates to true if the floor situated below the robot is black with probability ρ, where $\rho \in [0, 1]$ is a module parameter that is intended to be tuned during the design process by the optimization algorithm on a per-mission basis.

Gray-floor. Evaluates to true if the floor situated bellow the robot is gray with probability ρ, where $\rho \in [0, 1]$ is a module parameter that is intended to be tuned during the design process by the optimization algorithm on a per-mission basis.

White-floor. Evaluates to true if the floor situated bellow the robot is white with probability ρ, where $\rho \in [0, 1]$ is a module parameter that is intended to be tuned during the design process by the optimization algorithm on a per-mission basis.

Neighbor-count. Evaluates to true with probability $z(n) = \frac{1}{1+e^{\eta(\xi-n)}}$, where n if the number of robots detected by the range-and-bearing module, and $\eta \in [0, 20]$ and $\xi \in \{0, 1, ..., 10\}$ are parameters of the module that are intended to be tuned during the design process by the optimization algorithm on a per-mission basis.

Inverted-neighbor-count. Evaluates to true with probability $1 - z(n)$.

Fixed-probability. Evaluates to true with probability ρ, where $\rho \in [0, 1]$ is a module parameter that is intended to be tuned during the design process by the optimization algorithm on a per-mission basis.

5.1.2 Low-level behaviors

We describe here the behaviors of the modules of `Chocolate` and `Arlequin`, and the objective functions that were created in order to generate them for `Arlequin`.

Exploration. In `Chocolate`, the robot performs a random walk. It moves in a straight line until its front proximity sensors perceive an obstacle. It then rotates for a random number of timesteps in $\{0, 1, ..., \pi\}$. The parameter $\pi \in \{0, 1, ..., 100\}$ is a module parameter that is intended to be tuned during the design process by the optimization algorithm on a per-mission basis.

In `Arlequin`, we discretize the environment into a two dimensional grid G. The objective function, to be maximized, computes the total number of individual cells visited. It is expressed as follows:

$$\sum_{r=1}^{N}\sum_{i=1}^{X}\sum_{j=1}^{Y} G_r(i,j), \quad \text{with } G_r(i,j) = \begin{cases} 1 & \text{robot } r \text{ visited cell } (i,j), \\ 0 & \text{otherwise;} \end{cases} \tag{5.1}$$

where N is the number of robots in the swarm; and $X = 20$ and $Y = 20$ are the numbers of rows and columns in grid G, respectively.

Stop. In `Chocolate`, the robot stands still.

In `Arlequin`, the objective function, to be minimized, computes the number of individual robots that move. It is expressed as follows:

$$\sum_{t=1}^{T}\sum_{r}^{N} ||P_r(t) - P_r(t-1)||; \tag{5.2}$$

where $P_r(t)$ is the position of robot r at time t, and T is the total time of the experimental run.

Phototaxis. In `Chocolate`, the robot goes towards the light source, if perceived; otherwise, the robot moves straight.

In `Arlequin`, the objective function, to be minimized, computes the distance from individual robots to the light. It is expressed as follows:

$$\sum_{t=1}^{T}\sum_{r=1}^{N}||P_r(t) - P_{light}||; \tag{5.3}$$

where $P_r(t)$ and P_{light} are the positions of robot r at time t and of the light, respectively.

Anti-phototaxis. In `Chocolate`, the robot goes in the opposite direction of the light source, if perceived; otherwise, the robot moves straight.

In `Arlequin`, the objective function, to be maximized, computes the distance of individual robots to the light. It is expressed as follows:

$$\sum_{t=1}^{T}\sum_{r=1}^{N}||P_r(t) - P_{light}||; \tag{5.4}$$

where $P_r(t)$ and P_{light} are the positions of robot r at time t and of the light, respectively. It is the same as the one of *phototaxis* except is it maximized.

Attraction. In `Chocolate`, the robot goes in the direction of its neighboring peers (V_d), if perceived; otherwise, it moves straight. A parameter $\alpha \in [1, 5]$ controls the convergence speed towards the neighboring detected peers. It is a module parameter that is intended to be tuned during the design process by the optimization algorithm on a per-mission basis.

In `Arlequin`, the objective function, to be minimized, computes the distance between each pair of robots within the swarm. It is expressed as follows:

$$\sum_{t=1}^{T}\sum_{i=1}^{N-1}\sum_{j=i+1}^{N}||P_i(t) - P_j(t)||; \tag{5.5}$$

where $P_i(t)$ and $P_j(t)$ are the positions of robot i and j, respectively.

Repulsion. In `Chocolate`, the robot goes away from its neighboring peers, if perceived; otherwise, it moves straight. A parameter $\alpha \in [1, 5]$ controls the divergence speed away from neighboring detected peers. It is a module parameter that is intended to be tuned during the design process by the optimization algorithm on a per-mission basis.

In `Arlequin`, the objective function, to be minimized, computes, for each individual robot, the distance to its closest peer. It is expressed as follows:

$$\sum_{t=1}^{T}\sum_{r=1}^{N}||P_r(t) - P_{r_{min}}(t)||; \tag{5.6}$$

where $P_r(t)$ is the position of robot r and $P_{r_{min}}(t)$ is the one of the closest robot to robot r at time t.

5.2 Methods

In this experiment, we generated control software with three design methods. We tested these methods on two classical swarm robotics missions, an aggregation with decision and a foraging mission. The following section presents details on the material and methods, experimental protocol, and missions under analysis in this experiment.

5.2.1 Protocol

We considered a swarm of 20 e-puck robots that operate in a dodecagonal arena of $4.91\,\mathrm{m}^2$ delimited by walls, as described in Section 3.4. The robotic platform used in this experiment is the e-puck robot. It is formally described, along with its reference model in Sections 3.1 and 3.2. A picture of the e-puck in the configuration adopted in the experiments is given in Figure 3.1a. All simulations in this study were performed using ARGoS; details about the simulation setup are found in Section 3.3. Each design method is run 10 times, producing 10 instances of control software for each mission. These instances of control software are then tested once in simulation and once on a swarm of 20 e-puck robots. Each design process was allowed the same budget of 200 000 simulation runs. In the real robots setup, the performance of the swarm was computed automatically using data provided by a tracking system; details about the tracking system are presented in Section 3.4.1. We present the results in the form of notched box-and-whiskers plots. We also present aggregated results using the Friedman (Conover 1999) test. We refer to Section 3.5 for details about statistical analyses.

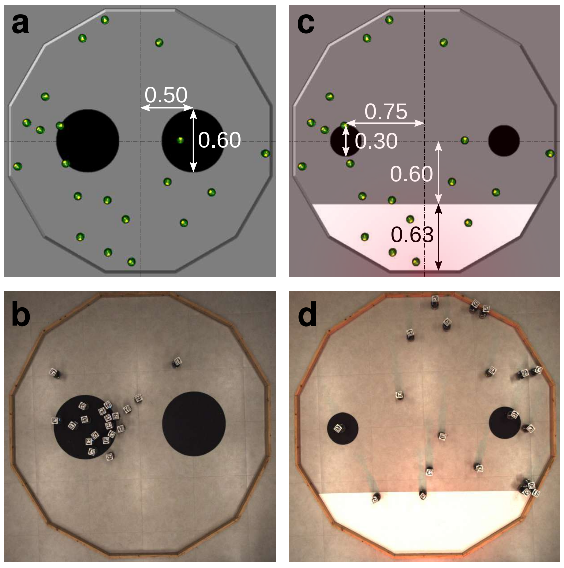


Figure 5.2: **Arenas for the two missions.** **a**, Aggregation-Xor, simulation; **b**, real robots. **c**, Foraging, simulation; **d**, real robots. The red glow in **c** and **d** indicates the presence of a light source at the bottom side of the arena. Dimensions (in meters) of the elements present in the arena are given in **a** and **c**.

5.2.2 Automatic design methods

In this experiment, we compare three automatic design methods: `Arlequin`, `EvoStick`, and `Chocolate`. `Arlequin` is our novel method; it is described in Section 5.1. `EvoStick` is the classical neuroevolutionary method that was tested among the ones in Chapter 4, and that is described in Section 3.7. `Chocolate` is the classical modular method with hand-crafted modules, that was also among the ones tested in Chapter 4 and is described in Section 3.6.

5.2.3 Missions under analysis

We conduct our experiments on two missions: Aggregation-Xor and Foraging. The Foraging mission is identical to the one described in Section 4.2.3; for the convenience of the reader, its description is repeated here.

Aggregation-Xor

The robots must aggregate on one of the two black areas in the arena. The size and positions of the different elements present in the arena are given in Figure 5.2. The performance of the swarm is computed based on the following objective function:

$$F_A = \max(N_l, N_r)/N; \tag{5.7}$$

where N_l and N_r are the number of robots located on the left and right black area, respectively; and N is the total number of robots. The duration of the experimental run is $T = 180\,\text{s}$ and $N = 20$ is the size of the swarm.

This mission differs slightly from the one described in Chapter 4, as it only considers the number of robots in each area at the end of the experimental run.

Foraging

The robots must find one of two food sources, represented by black areas, and return to the nest, represented by a white area. The size and positions of the different elements present in the arena are given in Figure 5.2. The performance of the swarm is computed based on the following objective function:

$$F_F = K; \tag{5.8}$$

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

5.3 Results

We present the results in Figure 5.3, the videos of the experimental runs are available as Supplementary Videos A1 and A2 (Hasselmann and Birattari 2022).

5.3.1 Aggregation-Xor

The results for the Aggregation-Xor mission are presented in Figure 5.3a. Simulation results show that the control software generated using `Arlequin` performs similarly to the one of `Chocolate`. However, the performance of `EvoStick` is significantly better than the ones of both `Arlequin` and `Chocolate`. This is typically the case in `EvoStick`, the performance in simulation tend to be significantly better than the one of modular methods. The performance of the control software that was generated by `Arlequin` and `EvoStick` drops significantly when it is evaluated on the physical robots. `EvoStick` is the method that suffers the most from the reality gap. The performance drop experienced by `Arlequin` is at most 0.475, whereas the one experienced by `EvoStick` is at least 0.55. The drop experienced

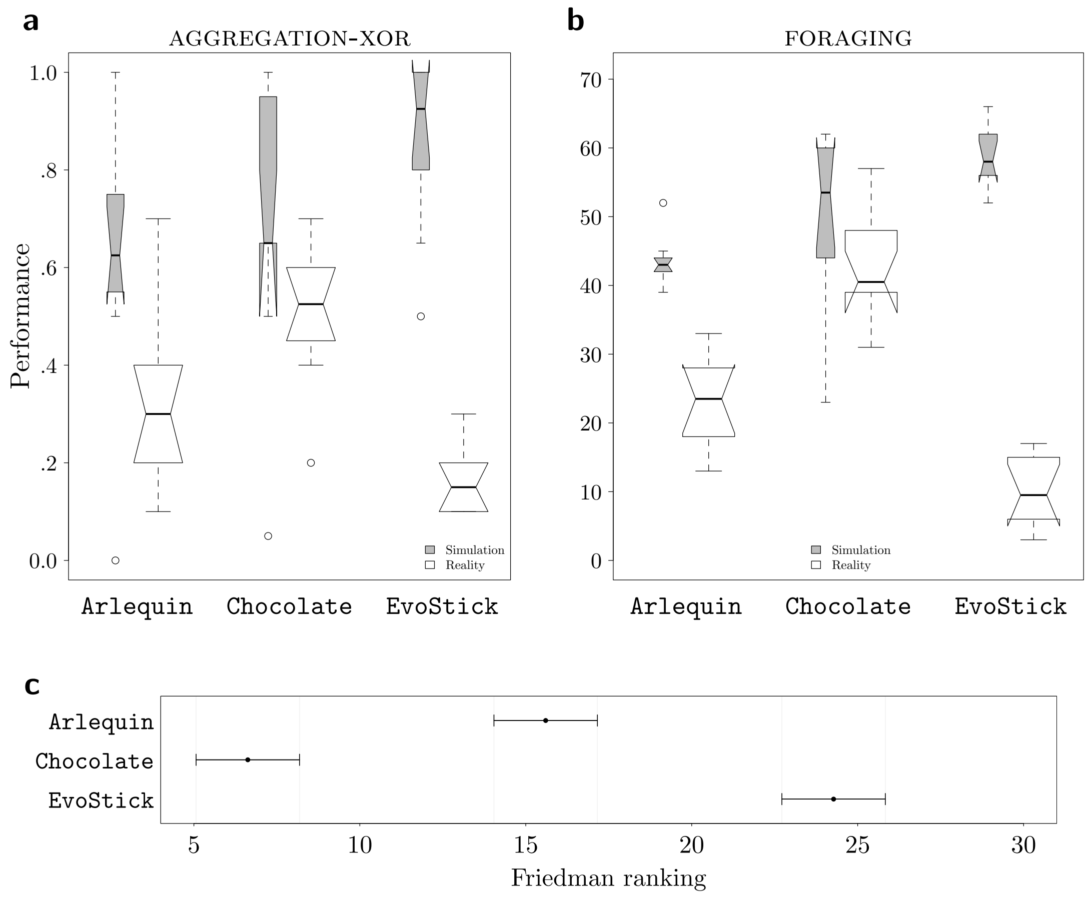


Figure 5.3: **Results.** **a**, Aggregation-Xor, performance obtained in simulation (narrow grey boxes) and in reality (thick white boxes) for all three methods (the higher the better); **b**, Foraging, performance obtained in simulation (narrow grey boxes) and in reality (thick white boxes) for all three methods (the higher the better); **c**, Friedman test results; Friedman test on the aggregate results in reality of the two missions, the plot shows the average rank and the 95% confidence interval (the lower the better).

by `Arlequin` is significantly lower than the one experienced by `EvoStick` (95% confidence computed with a paired Wilcoxon test).

The performance of the control software produced by `Chocolate` is similar in simulation and reality. The performance drop due to the reality gap in the three design methods is such that, in reality, `Arlequin` outperforms `EvoStick`, but is, in turn, outperformed by `Chocolate`.

5.3.2 Foraging

The results for the FORAGING mission are presented in Figure 5.3b. Simulation results show that `Arlequin` is outperformed by both `EvoStick` and `Chocolate`. When tested in reality, the three methods suffer from the reality gap, with `EvoStick` suffering the most, followed by `Arlequin` and `Chocolate`. The performance drop experience by `Arlequin` is at most 25.5, the one of `EvoStick` is at least 42. The performance drop experience by `Arlequin` is significantly lower than the one of `EvoStick` (95% confidence computed with a paired Wilcoxon test). Similarly to the results obtained in the AGGREGATION-XOR mission, the performance drop in the three methods is such that, in reality, `Arlequin` outperforms `EvoStick`, but is outperformed by `Chocolate`.

5.4 Discussion

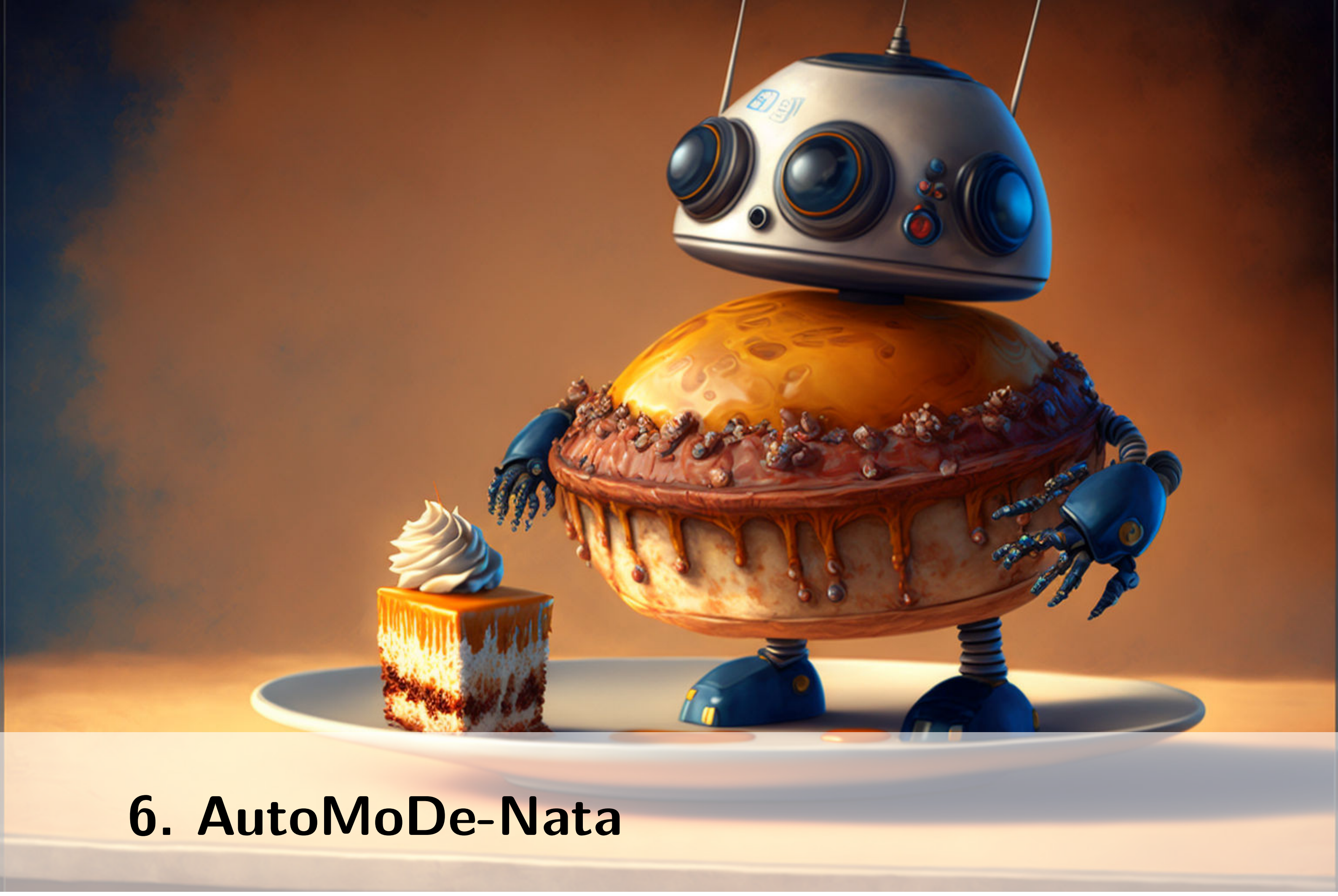
We presented `Arlequin`, a novel instance of AutoMoDe that differs from the previously presented ones by the nature of the predefined behavioral modules to be combined: `Arlequin` uses neural network modules generated via neuroevolution, whereas the others use hand-crafted ones. The behavioral modules of `Arlequin` were generated via `EvoStick`. We compared the performance of the control software generated by `Arlequin` with the one generated by `EvoStick` and `Chocolate` on two missions. In both missions, `Arlequin` is outperformed by `EvoStick` in simulation. This was expected and is explained by the fact that the representational power of `Arlequin` is reduced in comparison with the one of `EvoStick`. In one of the two missions considered, `Arlequin` is outperformed by `Chocolate` in simulation. This can be explained by the fact that the behavioral modules of `Chocolate` are parametric, which allows the optimization algorithm to fine-tune the modules to the problem at hand, whereas the behavioral modules of `Arlequin` are fixed. In both missions, the control software produced by `Arlequin` suffered from significant performance drops. However, the control software generated by `EvoStick` suffered from a significantly larger drop than the one produced by `Arlequin`, and as a result, `Arlequin` outperformed `EvoStick` in reality. This corroborates the conjecture of Francesca et al. (2014b): restricting the control software to be a combination of low-level, simple behaviors yields better results in reality than the traditional neuroevolutionary approach, despite being the other way around in simulation. Our results show that this also applies when the simple behaviors are controlled by neural networks.

We conjecture that the performance drop experienced by `Arlequin` in the two missions is to be ascribed to an overfitting that occurred at the module level, that is, during the implementation of the modules. This conjecture is based on the observation that, compared to `Chocolate`, the representational power of `Arlequin` is reduced as its behavioral modules are non parametric, which implies that the optimization algorithm cannot fine-tune their behavior to the problem at hand. We expect therefore that `Arlequin` is less likely to suffer from the overfitting that occurs at the combination level, that is, when the modules are combined to solve a specific mission. For the moment, this is just a conjecture that requires further investigation to be confirmed.

These results indicate that the idea behind `Arlequin` is a promising one. However, `Arlequin` still requires human expertise at the implementation level. As a next step, we wish to further reduce this required human expertise. Currently, despite being parametric and therefore fine-tuned by the optimization algorithm, the conditional modules of `Arlequin` are hand-crafted. Also, to obtain the neural-network modules of `Arlequin`, we devised objective functions that describe the hand-coded behaviors of `Chocolate`. Conceiving the appropriate objective functions to obtain the desired behaviors can be difficult and requires a reasonable knowledge of the platform at hand (Floreano and Urzelai 2000).

Gomes and Christensen (2018b) proposed an approach to conceive low-level behaviors in a completely automated fashion. Their approach is based on *repertoires* of behaviors obtained in a mission-agnostic fashion using a quality-diversity algorithm. The authors have shown that the repertoires they generated contain a wide variety of behaviors, and that some of these behaviors obtained performance that was close to one achieved by classical mission-specific automatic design. The authors state that such repertoires can be the cornerstone of a completely automated method to conceive complex swarm behaviors. The results we present in this chapter indicate that such method is indeed viable.

In the following chapter, we investigate the creation of such repertoires and how to automatically produce control software for robot swarms by combining behavioral modules selected from these repertoires.



6. AutoMoDe-Nata

In this chapter, we address the problem of automatically designing a modular method for the automatic design of robot swarms, which involves defining the modules that will be then automatically selected and assembled into an appropriate architecture (e.g., a finite-state machine or a behavior tree).

The definition and implementation of the modules that are to be combined automatically is critical to the success of a modular method. By success, we mean the ability of the method to exploit the capabilities of the robotic platform for which control software is conceived so that behaviors that are of interest to swarm robotics are produced (Birattari et al. 2021). A fully-automatic modular method is typically conceived to produce control software for a specific robotic platform, and cannot easily be ported to other ones. One can indeed imagine that robotic platforms of different nature (e.g., air-based, ground-based, water-based), or robotic platforms of the same nature but with different capabilities (e.g., communication, vision, actuation) would require specific modules to appropriately utilize their capabilities. One can also imagine that, if these modules were to be conceived manually, this should be done by an expert in (swarm) robotics to obtain the best possible results.

In this chapter, our goal is to conceive a new instance of AutoMoDe that requires less human expertise to be implemented or to be ported to different robotics platforms than the current instances, and to define a methodology to do so. In the previous chapter, we investigated the viability of replacing the hand-crafted low-level behavior modules of `Chocolate`, the state-of-the-art AutoMoDe method (Francesca et al. 2015), with automatically generated neural networks (see Chapter 5). We defined `Arlequin`, which produced control software that outperformed the one generated by a classical neuroevolutionary design method. Although these results were promising and opened the door to the exploration of novel design methods, we did not completely free ourselves from human expertise in the conception of `Arlequin`. Indeed, expert knowledge was involved in (i) the hand-coding of the condition modules and (ii) the definition of the performance measures that were used to generate the neural networks by artificial evolution—which is known to be particularly challenging (Doncieux and Mouret 2014; Floreano and Urzelai 2000).

To further dispense from human expertise in the definition of a modular method, we address here the two aforementioned points by (i) proposing a set of rules defined on the sensory capabilities of the robotic platform to automatically generate condition modules, and (ii) adopting an approach to generate repertoires of low-level behaviors in a mission-agnostic way, without the need to define an objective or a performance measure for each of the behaviors. Repertoires are large sets of low-level behaviors that are as diverse as possible. They are created using a quality-diversity algorithm that uses behavioral novelty of candidate behavior as the objective function of an optimization algorithm (Pugh et al. 2016). The approach presented in this chapter is based on Gomes and Christensen (2018b) for the creation of the repertoire of behaviors using novelty search with local competition (Lehman and Stanley 2011b) along with a set of hyperparameters that were selected to produce a repertoire with similar characteristics to the one presented in Gomes and Christensen (2018b).

We present `Nata`[1], a design method that automatically generates control software by selecting modules from a repertoire of mission-agnostic behaviors and assembling them into probabilistic finite state machines. We test `Nata` with physical robots on well-studied swarm robotics missions. `Nata` is, to the best of our knowledge, the

[1]The methods belonging to the AutoMoDe family all have food-related names: Vanilla, Chocolate, Gianduja, TuttiFrutti...The method presented in this chapter is based on the work of Gomes and Christensen (2018b) of the University of Lisbon. To acknowledge the source of inspiration and to celebrate the original and inspiring work of our colleagues, we named our method `Nata`, as in "pastéis de nata", the popular custard tarts from Belém, Lisbon.

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

As introduced in Chapter 3, AutoMoDe is a modular approach to the automatic design of control software for robot swarms that was introduced to address the issue of the reality gap. The creation of AutoMoDe was motivated by the observation that the reality gap problem resembles the one of *overfitting* encountered in machine learning (Francesca et al. 2014b). For details on the reality gap problem and the *bias-variance trade-off*, we refer the reader to Section 2.3.3. `Arlequin` (see Chapter 5) was introduced to test the conjecture that the robustness to the reality gap of AutoMoDe results from the restriction of the space of the control software that it can generate, rather than by the fact that the modules are skilfully hand-crafted. To this end, `Arlequin` assembles modules obtained via neuroevolution—also in this case, the modules are generated a priori and once and for all, in a mission-agnostic way. The modules of `Arlequin` were generated using performance measures defined to obtain behaviors that are qualitatively similar to the hand-crafted modules of `Chocolate`. Conceiving the appropriate performance measure so that the neuroevolutionary process produces the desired swarm behavior is challenging and is typically done via trial-and-error (Doncieux and Mouret 2014; Floreano and Urzelai 2000).

Repertoire-based methods using quality-diversity algorithms appear to be an interesting step forward in order to free ourselves from the burden of creating such performance measures.

Quality-diversity algorithms, such as MAP-elites (Mouret and Clune 2015) or novelty search with local competition (Lehman and Stanley 2011b), are algorithms that explore a given search space in order to find high-quality solutions to a problem while maximizing their spatial diversity. These algorithms have been already used to create repertoires of behaviors for legged robots (Cully and Mouret 2016; Duarte et al. 2018; Lehman and Stanley 2011b; Mouret and Clune 2015; Vassiliades et al. 2018), wheeled robots (Bossens and Tarapore 2021; Duarte et al. 2016b; Gomes and Christensen 2018a,b), robotic arms (Cully and Demiris 2018; Kim and Doncieux 2017; Mouret and Clune 2015), and UAV's (Engebråten et al. 2021). Most of the works considered *open-loop* controllers (Cully and Demiris 2018; Cully and Mouret 2016; Duarte et al. 2016b, 2018; Engebråten et al. 2021; Kim and Doncieux 2017; Mouret and Clune 2015; Vassiliades et al. 2018), that is, the control software modules in the repertoire do not use sensory information and only describe locomotor behaviors. The use of *closed-loop* controllers—that is, control software modules that make use of sensory information—was presented in subsequent works (Bossens and Tarapore 2021; Gomes and Christensen 2018a,b). In our research, we search the space of possible behaviors—in the form of neural networks—to build a repertoire of high-quality behavioral modules to be used as building blocks of the robots' control software.

Gomes and Christensen (2018b) were the first to use a quality-diversity algorithm to generate a repertoire of swarm behaviors. The authors evaluated all behaviors of the repertoire on eight missions to assess their quality. Results showed that the repertoire contained suitable solutions for all missions. In that work, behaviors of the repertoire were not assembled, the model of the robot was rather simple and no real-robot experiments were conducted. Subsequently, Gomes and Christensen (2018a) presented EvoRBC-II, a method to evolve repertoires and assemble the modules using a supervisor decision tree as an arbitrator that selects the low-level behavior to be executed. The method was assessed on nine single-robot missions but no real-robot experiment was conducted.

6.1 Modular repertoire-based automatic design

`Nata` belongs to the AutoMoDe family of modular methods. It combines two types of modules—behaviors and conditions—into probabilistic finite-state machines using the irace (López-Ibáñez et al. 2016) optimization algorithm (see Section 3.6 for details on AutoMoDe). A behavior is an action executed by a robot. A condition is a provision for switching from one behavior to another. The control software of each individual robot is a probabilistic finite-state machine in which each state is a behavior and each edge is associated with a condition that enables it depending on whether it is satisfied. Behaviors and conditions were generated automatically through procedures that will be detailed in the following of the section.

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

6.1.1 Generation of a repertoire of behaviors using novelty search with local competition

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

We created the repertoire of `Nata` following the idea introduced in Gomes and Christensen (2018b) for building a repertoire of behaviors for robot swarms. Each behavior in the repertoire is a neural network that can be used as control software on a robot.

We generated this repertoire of behaviors using an evolutionary process driven by novelty search with local competition (Lehman and Stanley 2011a,b). This process follows the framework introduced by Cully and Demiris (2018) and used by Gomes and Christensen (2018b), in which the selection of novel solutions and the construction of the repertoire are two independent steps; the algorithm is summarized in Algorithm 1. We first create an empty repertoire that will hold the set of best candidate neural networks and will eventually become the final repertoire. After generating the initial population, we evaluate all neural networks of the population in a set of randomly generated environments and compute the median behavior characterization and the mean quality score of each neural network. The randomly generated environments all share the same size, shape, and ground color but can have a floor patch of a random color (white, grey, or black) at a random position chosen between three possible ones, up to one obstacle box at the center of the arena, and up to one light source. We evaluate each neural network of the population on 10 random environments for 100 seconds. The behavior characterization is a vector that represents the behavior of the robots in the swarm when evaluated in different environments (Gomes et al. 2014). To characterize the behavior of the swarm, we compute for each individual robot the 7 following features: the linear and angular speed; the distance to walls/obstacles, to other robots, and to the closest robot; the ambient light and the ground color perceived. The behavior characterization vector is composed of 14 real values: the mean and the standard deviation for each of the features described above. These values are computed based on the evaluations of the swarm on the 10 random environments. The quality score represents the number of collisions that occurred during an evaluation of a neural network and is computed using the following function: $Q = 1 - C/(T \cdot N)$, where C is the number of collisions, T is the duration of the evaluation and N is the size of the swarm. After these steps, we update the repertoire with the newly evaluated neural networks based on their novelty score and local competition score. The novelty score is the mean distance of the current neural network to its k nearest neighbors. This distance is computed using the l^2-norm of the characterization vectors of the neural networks. The local competition score is the number of neural networks in the k nearest neighbors that are outperformed by the current neural network. In our case, we considered $k = 25$. The given neural network is then added to the repertoire if its nearest neighbor is sufficiently different from it (the distance between is greater than parameter l). The given neural network may also replace its nearest neighbors in the repertoire, if it is not sufficiently different (the distance between is less than l) but its quality score is strictly greater and the second nearest neighbor is sufficiently different from it. The following generation of neural networks is created by crossover and mutation on the genomes of the current generation. The evolutionary process is a modified version of the NEAT (Stanley and Miikkulainen 2002) algorithm to allow Pareto-based bi-objective optimization, where the two objectives are the novelty score and the local competition score.

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

6.1.2 Generation of rules for condition modules

Condition modules are generated automatically via a set of rules that operate on the reference model of the robot (see Section 3.2 for details about the reference model). More precisely, the input variables of the reference model, which represent sensory information, are used to define triggers for the condition modules. We separate input variables in three classes: categorical, continuous, or vector inputs. For each input variable, depending on the class to which it belongs, a specific ruleset is automatically applied to generate one transition module.

A condition module based on a categorical input triggers a transition with probability p when the input is detected to be C with $C \in \{c_1, c_2, ..., c_n\}$. If multiple sensors form the input, all the values must be detected to be C. A condition module based on a continuous input triggers a transition with probability p when the input is detected to be above ($d = 1$) or below ($d = 0$) the threshold θ with $\theta \in \mathbb{R}$. If multiple sensors form the input, the sum of the values need to be above or below the threshold θ. A condition module based on a vector input triggers a transition with probability p when the input is detected with angle in quadrant Q with $Q \in \left\{[0, \frac{\pi}{2}[, [\frac{\pi}{2}, \pi[, [-\pi, \frac{-\pi}{2}[, [\frac{-\pi}{2}, 0[\right\}$. In fact, for vector inputs, the magnitude of the vector is disregarded and the angle is used as a categorical input with four

Algorithm 1 Repertoire evolution with novelty search with local competition. In our case, $l = 1.5$ is the repertoire distance threshold, $s = 50$ is the repertoire growth per generation, $S = 2000$ is the archive capacity. The archive A is filled during the execution of the algorithm with randomly selected neural networks from the population to encourage uniform behavior exploration (Cully and Demiris 2018). It is used to compute the novelty and local competition scores.

```
1: $A \leftarrow \emptyset$, $R \leftarrow \emptyset$        ▷ Create empty archive $A$ and repertoire $R$
2: $P \leftarrow$ RandomInitialPopulation()        ▷ Initialise population $P$
3: for $g \in$ generation do
4:     for $i \in$ P do
5:         for $e \in$ E do        ▷ Evaluate individual $i$ in all environnements
6:             $q_e, b_e \leftarrow$ Evaluate($i, e$)
7:         $B(i) \leftarrow$ GeometricMedian($\{b_e : e \in E\}$)        ▷ Store median behavior characterization
8:         $Q(i) \leftarrow \sum_{e \in E} \frac{1}{|E|} q_e$        ▷ Store quality score
9:     for $i \in$ P do        ▷ Update repertoire with new individuals
10:        $\chi \leftarrow$ NearestNeighbors($i, A \cup P, k$)        ▷ $k = 25$ nearest neighbors in archive and population
11:        $N(i) \leftarrow \sum_{x \in \chi} \frac{1}{|\chi|}$dist($B(i), B(x)$)        ▷ Compute novelty score
12:        $LC(i) \leftarrow \sum_{x \in \chi} [Q(i) > Q(x)]$        ▷ Compute Local competition score
13:        $\chi_1, \chi_2 \leftarrow$ NearestNeighbors($i, R, 2$)        ▷ 2 nearest neighbors in repertoire
14:        if $|R| = 0$ or $|R| > 0$ and dist($B(i), B(\chi_1)$)$> l$ then        ▷ Individual different from nearest
15:            $R \leftarrow R \cup \{i\}$
16:        else if $|R| > 1$ and $Q(i) > Q(\chi_1)$ and dist($B(i), B(\chi_2)$) $> l \cdot 0.1$ then
17:            $R \leftarrow (R \setminus \{\chi_1\}) \cup \{i\}$
18:    if $|A| > S - s$ then        ▷ Update archive with new individuals, remove some if full
19:        $A \leftarrow A \setminus$ SelectRandom($A, |A| + s - S$)
20:    $A \leftarrow A \cup$ SelectRandom($P, s$)
21:    $P \leftarrow$ Breed($P$) based on $N(i)$ and $LC(i)$        ▷ Create next generation
22: return $R$
```

Table 6.1: Condition modules: Five condition modules were generated based on the five inputs of reference model RM1.1 presented in Table 6.2: proximity, light, ground, number of neighboring robots perceived, and attraction vector.

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

distinct values representing four equal quadrants of $\pi/2$ rad. The parameters p, C, d, θ, and Q are fine-tuned by the optimization algorithm, when they apply. The details specific to the robotic platform in use, including the classes of the inputs considered here are given in Section 6.2.2 and Table 6.2.

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

The specific condition modules generated for the e-puck robot for `Nata` using reference model RM1.1 are presented in Table 6.1.

6.1.3 Assembly of condition and behavior modules

As customary in most methods belonging to the AutoMoDe family, `Nata` generates control software by assembling behavior and condition modules into probabilistic finite-state machines. `Nata` uses the optimization algorithm irace (López-Ibáñez et al. 2016) to generate probabilistic finite-state machines that are limited in size: they can comprise up to 4 states and up to 4 outgoing transitions per state; this constraint was set in other AutoMoDe methods such as `Vanilla`, and `Chocolate`. For each state of the probabilistic finite-state machine, irace selects one of the neural networks of the repertoire. For each transition, irace configures one condition module.

6.2 Methods

In this experiment, we generated control software using four design methods. We tested these methods on three classical swarm robotics missions. Their complexity aligns with previous work in the fully automatic design of robot swarms (see Chapter 4). The following sections present details about the material and methods, experimental protocol, and missions under analysis in this experiment.

6.2.1 Protocol

We considered a swarm of 20 e-puck robots that operate in a dodecagonal arena of 4.91 m^2 delimited by walls, as described in Section 3.4. The robotic platform used in this experiment is the e-puck robot. It is formally described, along with its reference model, in Section 6.2.2. A picture of the e-puck in the configuration adopted in the experiments is given in Figure 3.1a. All simulations in this study were performed using ARGoS; details about the simulation setup are found in Section 3.3. Each design method is run 10 times, producing 10 instances of control software for each mission. These instances of control software are then tested once in simulation and once on a swarm of 20 e-puck robots. Each design process was allowed the same budget of 200 000 simulation runs. In the real robots setup, the performance of the swarm was computed automatically using data provided by a tracking system; details about the tracking system are presented in Section 3.4.1. We present the results in the form of notched box-and-whiskers plots. We present normalized aggregated results, aggregated results using the Friedman (Conover 1999) test, and normalized aggregated performance drop results. We refer to Section 3.5 for details about statistical analyses.

Table 6.2: Reference model RM1.1: capabilities of the e-puck robotic platform used in this study. For sensors, the category—that defines the derived condition modules—, is indicated in the last column.

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

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

Period of the control cycle: 100 ms

6.2.2 Reference model

The specific configuration of the e-puck robot used in this study is formally described by reference model RM 1.1. This reference model is identical to the one used in previous chapters of this thesis. Details about the reference model are given in Section 3.2. For the convenience of the reader, RM 1.1 is repeated in Table 6.2, with the added mention of the different classes of the sensors as defined in Section 6.1.2. All methods in this study use the variables defined in this reference model to generate control software.

6.2.3 Automatic design methods

In this experiment, we compare four automatic design methods: `Nata`, `Arlequin`, `EvoStick`, and `Chocolate`.

`Nata` is our novel modular method that generates control software by assembling modules from a repertoire. It is described in detail in Section 6.1. `Arlequin` is the modular method introduced in Chapter 5, that generates control software by assembling modules from a set of behavior-specific pre-trained neural networks. It is described in detail in Section 5.1. `EvoStick` is a classical neuroevolutionary method that generates control software in the form of neural networks. It is described in detail in Section 3.7. `Chocolate` is the classical modular method that generates control software by assembling hand-crafted modules. It is described in detail

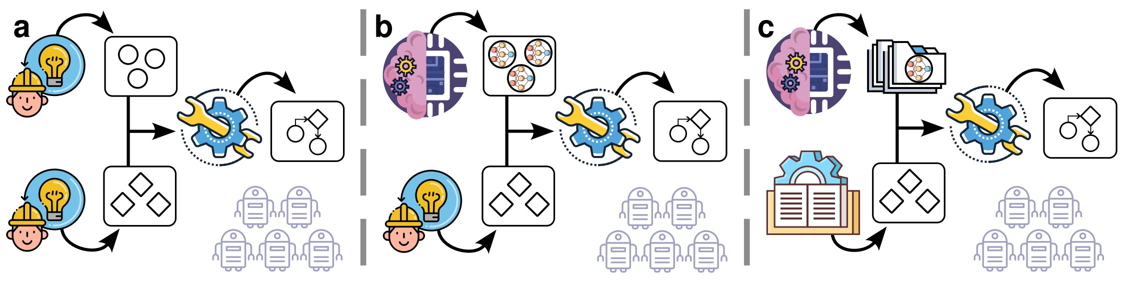


Figure 6.1: **Illustration of the modular design methods.** **a**, `Chocolate`, the behavior modules (black circles) and the condition modules (black diamonds) are hand-crafted, then, the modules are automatically assembled into probabilistic finite-state machines; **b**, `Arlequin`, the behavior modules (black circles) are pre-trained neural networks that are generated using behavior-specific objective functions; the condition modules (black diamonds) are hand-crafted, then, the modules are automatically assembled into probabilistic finite-state machines; **c**, `Nata`, a repertoire of behavior modules (black folder) in the form of neural networks is generated using a quality-diversity algorithm; the condition modules (black diamonds) are generated automatically via a set of rules, then, the modules are automatically assembled into probabilistic finite-state machines.

in Section 3.6. The differences between the three modular methods `Chocolate`, `Arlequin`, and `Nata` are illustrated in Figure 6.1.

6.2.4 Missions under analysis

The three missions (Figure 6.2) considered are classical swarm robotics missions; here we present their characteristics and objective functions. The XOR-Aggregation and the Foraging missions are identical to the ones with the same names presented in Chapter 4. For the convenience of the reader, we repeat their characteristics and objective functions hereafter.

XOR-Aggregation

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

$$F_{\mathrm{a}} = \sum_{t=1}^{T} \sum_{i=1}^{N} I_i(t); \tag{6.1}$$

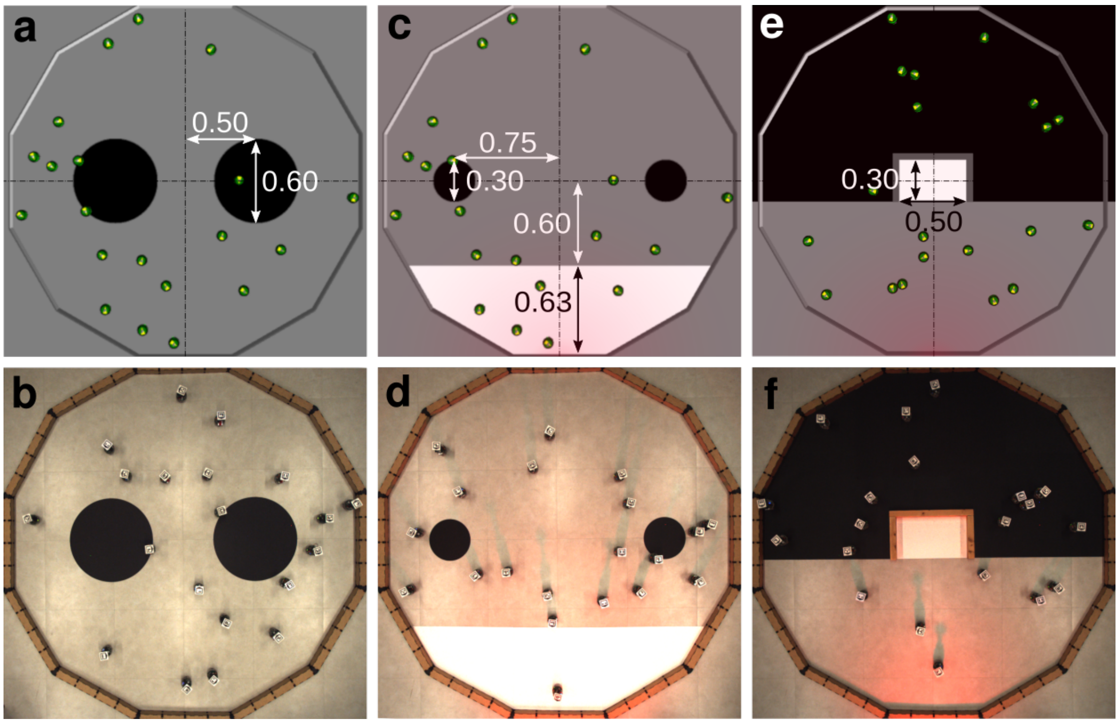


Figure 6.2: **Arenas for the three missions.** **a**, XOR-AGGREGATION, simulation; **b**, real robots. **c**, FORAGING, simulation; **d**, real robots. **e**, SHELTER W/CUES, simulation; **f**, real robots. The 20 robots operate in a dodecagonal arena of 4.91 m^2, the red glow in **c**, **d**, **e**, and **f** indicates the presence of a light source at the bottom side of the arena. Dimensions (in meters) of the elements present in the arenas are given in **a**, **c**, and **e**.

$$I_i(t) = \begin{cases} 1, & \text{if robot } i \text{ is in the area with a majority of robots;} \\ 0, & \text{otherwise.} \end{cases} \tag{6.2}$$

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

Foraging

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

$$F_\text{f} = K; \tag{6.3}$$

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

Shelter with cues from the environment

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

$$F_\text{s} = \sum_{t=1}^{T} \sum_{i=1}^{N} I_i(t); \tag{6.4}$$

$$I_i(t) = \begin{cases} 1, & \text{if robot } i \text{ is in the shelter;} \\ 0, & \text{otherwise.} \end{cases} \tag{6.5}$$

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

6.3 Repertoire analysis

Once the repertoire is generated, we analyze it to get some insight on the behavior modules that were selected by the algorithm. It is important to note that this analysis should not be considered as part of the procedure of the creation of the

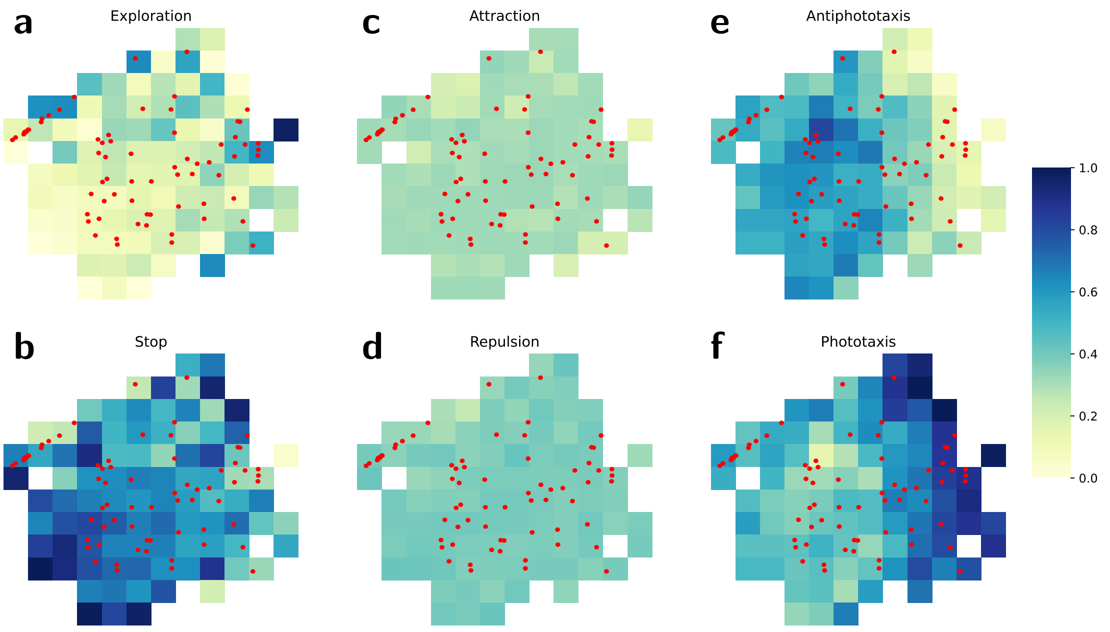


Figure 6.3: **Repertoire analysis.** Performance of the behaviors of the repertoire on the objective functions of `Arlequin`. We used principal component analysis (PCA) on behavior vectors space to reduce it to 2d (the 2 principal components account for 69.4% of the variance). For all behaviors in the repertoire we created, we evaluate them 10 times of each of the objective functions that were used to created the behaviors of `Arlequin` (see Section 5.1.2). For each behavior of the repertoire, the average performance is normalized, using min-max normalization, based on the performance of the behaviors of `Arlequin`. The 2d plane is discretised and the average performance of each region is mapped to the color of the heatmap. The Red dots represent the modules of the repertoire that are actually used in a probabilistic finite-state machine of the control software that were generated in this study. Performance of the repertoire on the objective function of `Arlequin` for: **a.** exploration; **b.** stop; **c.** attraction ; **d.** repulsion **e.** antiphototaxis; **f.** phototaxis.

repertoire. Indeed, the work involved in this analysis requires domain knowledge and expertise, which goes against our philosophy of reducing the expert knowledge needed for the implementation of the automatic design method. This analysis was performed *post factum*, after the automatic design phase and after the robot experiments, and was thus not used to fine-tune or steer the creation of the repertoire.

6.3.1 Analysis method

The repertoire of `Nata` is comprised of 672 behavior modules. We evaluate the performance of these modules on the six objective functions that we created for the design of the modules of `Arlequin` (see Section 5.1.2 for details on these objective functions). The six objective functions were created in `Arlequin` to train neural networks that would learn each of the six behaviors already present in `Chocolate`. In the case of this analysis, we only use these objective functions as performance metrics to evaluate the repertoire that was generated using the quality-diversity method described in Section 6.1. Each module of the repertoire is evaluated 10 times using each objective function and we record the performance over these 10 runs for each objective function. The average performance is then normalized using min-max normalization based on the performance for each objective function of the respective modules of `Arlequin`. We normalize using the performance of the modules of `Arlequin` to be able to assess whether satisfactory behaviors are present in the repertoire of `Nata`, compared to the modules of `Arlequin`.

In order to visualise the repertoire, we apply a principal component analysis (PCA), in order to reduce the dimensionality of the behavior characterization vector (described in Section 6.1.1) from 14 to 2 (here, the 2 principal components account for 69.4% of the variance). Each module is thus associated with a 2d space coordinate and a performance value. The 2d space is then discretised and an average in performed on the performance of the modules in the same spatial zone.

The resulting heatmap visualization is presented in Figure 6.3. A light yellow color indicates low performance; a dark blue color indicates high performance (close to the average one of `Arlequin` the corresponding module in `Arlequin`). Completely white areas represents areas in the behavior space that do not contain any module. The red dots printed on top of the heatmap represent the localization of the behavior modules that were actually selected during the automatic design phase in our study. They are included in the different probabilistic finite-state machines generated by `Nata` that we evaluated to report the results in Section 6.4.

6.3.2 Observations

We analyse the characteristics of the repertoire of `Nata` reported in Figure 6.3. For the exploration, stop, antiphototaxis, and phototaxis behaviors, modules with high performance are found in the repertoire, as indicated by the dark blue zones present in Figure 6.3a,b,e,f. In attraction and repulsion, the modules in the repertoire do not perform as well as the ones of `Arlequin`. The median quality of the repertoire is 0.99883, which indicates that, in simulation, the collisions between the robots are limited (see Section 6.1.1 for details on the quality score).

We observe that, as expected, pairs of modules that are dual behaviors (i.e. exploration/stop, attraction/repulsion and antiphototaxis/phototaxis) do indeed cover different zones in the behavior space. One module that perform well in exploration is expected to perform badly in stop, same goes for attraction/repulsion and antiphototaxis/phototaxis respectively. We also observe that compound behaviors, showing good performance in multiple objective functions, are found in the repertoire.

The positions of the red dots, over the heatmap, represent behaviors that are selected by the optimization algorithm in the design phase that we executed to obtain the results presented in Section 6.4. We observe that the modules in use are spread in the behavior space. We also observe that some high performing zones do not contain any used modules. This can be explained by the compound nature of the behaviors; some modules could be better suited to a specific mission, in conjunction with others. The view offered by this analysis is limited to the behaviors that were created for `Arlequin`, and their respective objective functions, which may not be the best possible ones to be assembled to tackle a given mission.

The spatial diversity of used modules, the high median quality score, and the satisfactory performance on the different objective functions suggest that the repertoire is mostly composed of potentially useful modules.

6.4 Results

Figure 6.4a shows the performance obtained by the four methods on the three missions, both in simulation and on physical robots. In simulation, for the FORAGING mission, `EvoStick` outperforms the other three modular methods, which perform similarly (the notches representing the 95% confidence interval on the boxplots overlap). For XOR-AGGREGATION, all methods perform similarly, whereas for SHELTER W/CUES, differences between methods are all significant: `EvoStick` performed best, followed by `Arlequin`, then `Chocolate` and finally `Nata`. Overall, when aggregating all results (see Figure 6.4b), `EvoStick` produced control software that performed best in simulation. Among the modular methods, `Arlequin` and `Chocolate` produced control software that perform similarly, and the one produced by `Nata` is outperformed by the one of `Chocolate`.

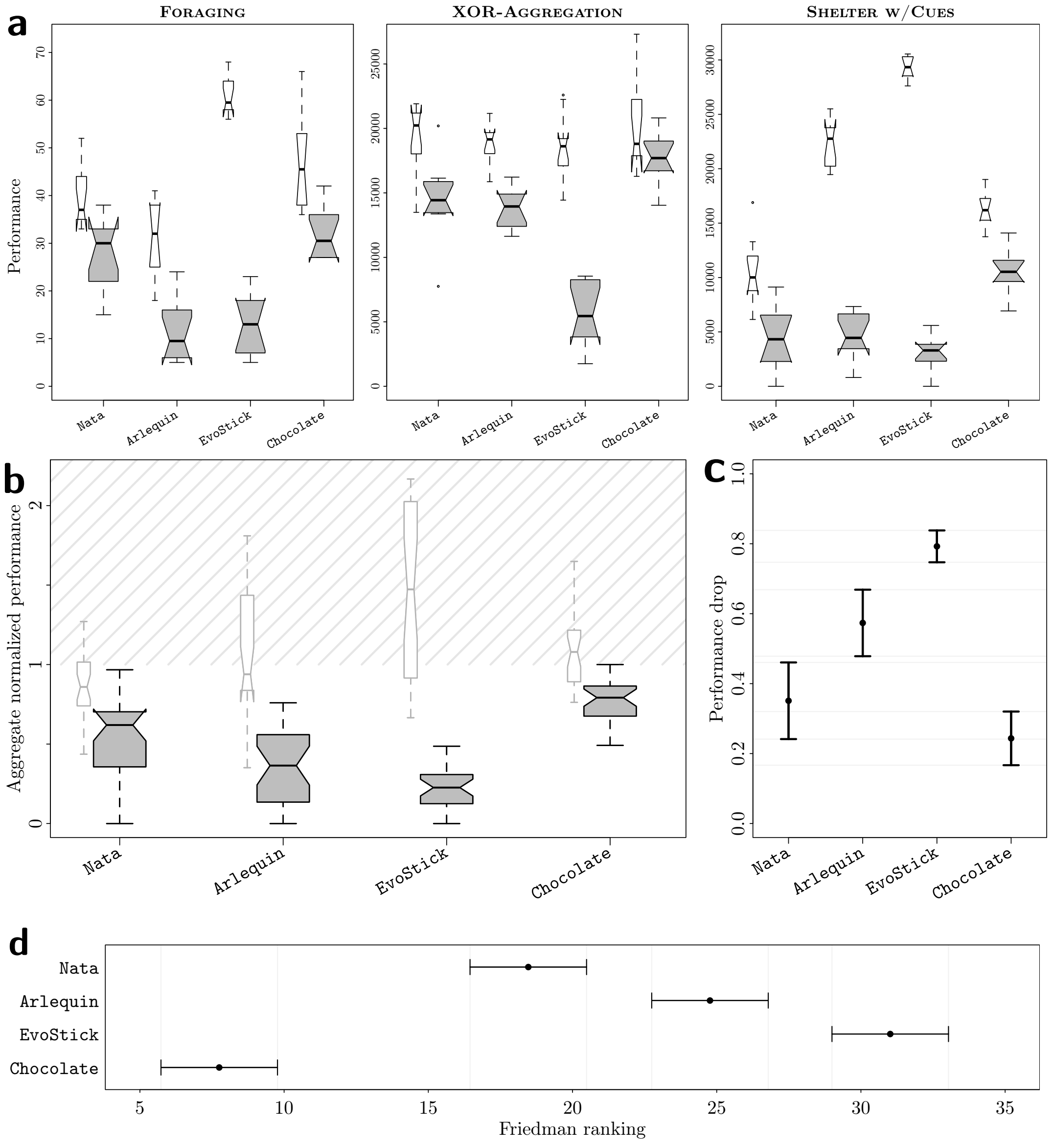


Figure 6.4: **Results. a.** Results per mission. Performance obtained in simulation (narrow white boxes) and in reality (thick grey boxes) in all three missions (the higher the better). **b.** Aggregated normalized results. Aggregated performance in simulation (narrow white boxes) and in reality (think grey boxes) across the three missions (the higher the better). Prior to aggregation, results are normalized between the lowest and the highest performance observed in reality by any method in a given mission. As a result, aggregated reality results range from 0 to 1, whereas aggregated simulation results might exceed 1 (shadowed area). This is observed because, in many cases, the performance in simulation exceeded the one in reality. **c.** Normalized performance drop experienced by the methods, aggregated across all missions, and 95% confidence interval. For a given instance of control software, the performance drop is computed as the difference between the performance assessed in simulation and the one observed in reality, and is normalized with the one assessed in simulation. **d.** Friedman test results. Friedman test on the aggregate results in reality of the three missions, the plot shows the average rank and the 95% confidence interval (the lower the better).

In reality, things are different. In fact, we observed several rank inversions between the methods. For FORAGING, `Nata` and `Chocolate` perform similarly, and outperform both `Arlequin` and `EvoStick`. For XOR-AGGREGATION, `Chocolate` is the best performing method. `Nata` and `Arlequin` perform similarly and slightly worse than `Chocolate`, but considerably better than `EvoStick`. For SHELTER W/CUES, the median performance of `Chocolate` is twice better than those of the other three methods, which perform similarly. Overall, when executed on a swarm of e-puck robots, the control software produced by `Chocolate` is the best performing one, followed by the one produced by `Nata`, then `Arlequin` and `EvoStick` (see Figure 6.4b). According to the Friedman rank sum test, the differences between these four methods are all significant with a confidence level of at least 95% (see Figure 6.4d).

Figure 6.4c shows the performance drop from simulation to reality, which gives an estimation of how much the methods are affected by the reality gap. `EvoStick` suffers from the largest overall performance drop and shows very poor performance in reality in all three missions, whereas `Chocolate` suffered from the lowest overall performance drop and is the method that is the best at crossing the reality gap. `Nata` suffers from a significantly lower performance drop than `EvoStick` and `Arlequin`, but larger than the one `Chocolate`, although not significantly.

Both in simulation and with physical robots, the results that we obtained using already existing methods, namely `Chocolate`, `EvoStick` and `Arlequin`, are consistent with previous studies (see Francesca and Birattari (2016), Ligot and Birattari (2020), and previous results presented in Chapters 4 and 5).

It is also worth mentioning that, as it can be observed in the Supplementary Videos N1 to N3 (Hasselmann and Birattari 2022), and as shown by the results in simulation, the different missions considered in the study can be accomplished by the design methods, and the allocated computing resources are sufficient to reach a satisfactory level of performance. All results obtained in simulation and real-robot experiments are available as supplementary data (Hasselmann and Birattari 2022).

6.5 Discussion

By introducing `Nata`, we addressed the drawbacks of both the neuroevolutionary robotics approach and the modular one: neuroevolutionary methods suffer from important performance drops due to the reality gap, whereas modular methods require human designers to meticulously implement behavior and condition modules. We presented `Nata`, a modular method that produces control software by automatically selecting and combining task-agnostic behaviors that were themselves automatically generated, via novelty search. `Nata` produced control software that is more robust to the reality gap than the one produced by the classical neuroevolutionary approach, whereas the process adopted to generate the modules to be combined requires less human expertise than the ones previously adopted by other modular methods. The results presented are to be considered as a proof of concept; future work should address the issue of further assessing the capabilities of `Nata` and extending them. In particular, one should focus on identifying the class of missions that `Nata` can tackle and on further improving its robustness to the reality gap.

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

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

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

As `Nata` produced control software that is more robust to the reality gap than the one produced by `Arlequin`, it appears that generating behaviors in the form of neural networks using novelty search with the aim of minimizing collisions is more effective than using direct evolution.

In this chapter, we reached the closest point we have ever been to automatically creating automatic design methods.

With `Nata`, we have gathered more evidence that the cornerstone of the creation of automatic design methods is the creation of the modules. The modules of `Chocolate` were crafted by hand and evaluations on physical robots during their implementation ensured that they crossed the reality gap satisfactorily. Evaluating the modules of the repertoire of `Nata` on robots with the objective of potentially pruning the repertoire to only keep modules that cross the reality gap satisfactorily would be extremely costly and time consuming. Indeed, a repertoire typically contains several hundreds of behavioral modules and each module should be evaluated in multiple environments so as to correctly assess its behavior characterization vector.

We foresee that the pruning of the repertoire could be done by leveraging the concept of pseudo-reality (Ligot and Birattari 2018). A pseudo-reality is a simulation model, different from the one used during the design, that is used to mimic the reality gap experienced when going from simulation to reality. Evaluations of behavioral modules in pseudo-reality, which are fast and inexpensive, could give an idea on their robustness to the reality gap and could subsequently be used to filter and select modules with the best potential to perform seamlessly in reality.

7. Conclusions

This thesis contributes to the progress of automatic modular design of robot swarms. Designing robot swarms is a hard problem; the relationship between the macroscopic behavior of the swarm, and the microscopic behavior of the individual robots is difficult to identify. This design problem is one of the major issues in swarm robotics; to this day, no general engineering methodology is available. Automatic design is foreseen as a promising way to create behaviors for robot swarms (Dorigo et al. 2020). However, the lack of proper benchmarking of automatic design methods is impeding the development of this research field.

As a first contribution, we provided a critical review of the state of the art, in order to establish the current practices in automatic design. We reviewed the most influential publications in the domain and especially focused our attention on offline automatic design and automatic modular design.

As a second contribution, we proposed a novel categorization of automatic design methods. Our categorization adds a new dimension to the already existing categorization between online and offline design methods. Online design brings the optimization process onboard the robots in the production environment, whereas, in offline design, the optimization is done before-hand, in a simulated environment, before deployment. In our orthogonal categorization, we disambiguate between semi- and fully-automatic design. In semi-automatic design, human intervention is allowed during the design process, whereas in (fully-)automatic design, once the design method has been defined, it operates without any intervention. This novel categorization helps to better understand the purpose and possible application of different automatic design methods. By establishing a clear difference between methods that operate on a class of missions compared to ones that are tailored to one specific mission, we believe this categorization will also allow fairer comparisons between design methods.

One of the objectives of this field is also to bring robot swarms into the physical world. Comparisons between automatic design methods should, therefore, also include analyses that indicate how robust to the reality gap one method is. This difference between the simulated environment (or more generally the environment used during the design phase), and the real world (the production environment) can induce significant differences in the behavior of the robot swarm. The reality gap is a major problem in offline design of robot swarms and the robustness to the reality gap, of a given method, is an important criterion of the success of an automatic design method. As our third contribution, we re-implemented, tested, and analysed the most popular automatic design methods using standard optimization algorithms. This analysis is the most extensive comparison of automatic design methods currently available in the literature that is supported by simulation and real robot experiments. This study shows the importance of the reality gap problem faced by many automatic design methods and specifically by neuroevolutionary design methods.

This study also further stressed the relevance of modular methods as methods that are robust to the reality gap. The conjecture, made by Francesca et al. (2014b), is that the reality gap problem is tightly linked to the *bias-variance* tradeoff. Control software architectures with high variance and low bias, such as neural networks, showcase high representational power. This can be a source of *overfitting* to the design environment and lead to decreased performance in the production environment. By introducing modules, we inject bias in the control software architecture with the goal of reducing the variance and therefore increasing its overall robustness to the reality gap. This work registers itself within the framework of the DEMIURGE project, whose aim is to create an end-to-end system to define and create swarm robotic systems by leveraging modular methods. Modular methods, such as most of the ones of the AutoMoDe family, require expert knowledge to be implemented, especially for the creation of the modules that are assembled and tuned. In contrast, neuroevolutionary methods require little knowledge and are easier to implement. Our fourth contribution is the creation of a modular automatic design method that joins the best of the two worlds: the ease of implementation of neural networks and the robustness to the reality gap of modular ones. We presented `Arlequin`, a method that uses pre-trained neural networks as modules. The creation and study of `Arlequin` showed that overfitting was reduced by the use of neural networks as modules and suggested that using neuroevolution in conjunction with modular methods is a promising approach. We have shown that the reality gap experienced by AutoMoDe methods comes mainly from the design of modules.

Our fifth contribution comes in line with the previous one. We created another modular automatic design method with the objective of further reducing the expert knowledge in swarm robotics needed to implement it, while also using neural networks as modules. Leveraging the idea of using novelty search to create a repertoire of behaviors, we presented `Nata`, a modular method that uses an automatically generated repertoire of behavioral modules and rule-based automatically designed conditional modules. We showed that meaningful behaviors where successfully obtained using this technique. The control software that we obtained still remained more robust to the reality gap than traditional neuroevolutionary methods. Our results suggest that neuroevolution is a viable approach to the creation of modules. We stressed that the cornerstone of the creation of automatic modular design methods is the definition and implementation of the modules.

We supported all studies presented in this thesis with real robot experiments using hardware and software that were partially or entirely developed in the framework of the DEMIURGE project. The extensive experimental work and implementation represent our last two contributions in this thesis. These are transversal contributions that supported the realization of the studies presented above. We believe that open research, concerning results as well as both software and hardware implementations also contributes to the development of the field of automatic design as a whole. Our implementations are available online as open-source software.

Future work should be dedicated to addressing the overfitting occurring at the module level. One should explore different ways of generating and selecting the behavioral modules that will then be part of the repertoire of modules to be combined into probabilistic finite state machines.

As a first step, one could explore different parameters of the neuroevolutionary process used to generate the modules, and select them on the basis of their performance assessed on physical robots. Since repertoires of behaviors can potentially be large, and to reduce the costs and time associated with performing tests in reality, one could also explore the use of *pseudo-reality* (Ligot and Birattari 2018, 2020, 2022)—that is, another simulation model, different from the one used during the design—to assess and select the modules. A different approach to generate the behavioral modules we wish to explore is the *transferability approach* of Koos et al. (2013). This approach is based on the occasional evaluation of the performance of instances of control software found during the design process, and based on these evaluations, steering the optimization algorithm to solutions that cross the reality gap satisfactorily, compared to direct neuroevolutionary methods. Also in this case, pseudo-reality could replace reality to reduce the costs and time of evaluations required during the design process.

One could also explore different control software architectures, such as trees or fixed sized small neural networks. Different architectures may lead to different constraints and potentially different types of modules.

For now, all experiments were done using the e-puck robots. Another important step in validating our approach should be to assess our automatic design methods on different robotic platforms. Diversity of robotic platforms, simulation environments and models, will also help validate the approach in a more systematic way.

It will also be important to refine and clarify the class of missions that a given robotic platform can tackle given its reference model. One could also study if and how the definition and implementation of the modules and the control software architecture affect this class of missions.

Bibliography

Ampatzis, C., Tuci, E., Trianni, V., Christensen, A. L., and Dorigo, M. (2009). "Evolving self-assembly in autonomous homogeneous robots: experiments with two physical robots". In: *Artificial Life* 15.4, pp. 465–484.

Auger, A. and Hansen, N. (2005). "A restart CMA evolution strategy with increasing population size". In: *2005 IEEE Congress on Evolutionary Computation*. Vol. 2. IEEE, pp. 1769–1776.

Beal, J., Dulman, S., Usbeck, K., Viroli, M., and Correll, N. (2012). "Organizing the aggregate: languages for spatial computing". In: *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments*. IGI Global, pp. 436–501.

Beni, G. (2005). "From swarm intelligence to swarm robotics". In: *Swarm Robotics: SAB 2004 International Workshop*. Vol. 3342. Lecture Notes in Computer Science. Springer, pp. 1–9.

Berlinger, F., Gauci, M., and Nagpal, R. (2021). "Implicit coordination for 3D underwater collective behaviors in a fish-inspired robot swarm". In: *Science Robotics* 6.50, eabd8668.

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

Bianco, R. and Nolfi, S. (2004). "Toward open-ended evolutionary robotics: evolving elementary robotic units able to self-assemble and self-reproduce". In: *Connection Science* 16.4, pp. 227–248.

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.

Birattari, M. (2009). *Tuning Metaheuristics: A Machine Learning Perspective.* Springer.

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.

Birattari, M., Delhaisse, B., Francesca, G., and Kerdoncuff, Y. (2016). "Observing the effects of overdesign in the automatic design of control software for robot swarms". In: *Swarm Intelligence: 10th International Conference, ANTS 2016.* Vol. 9882. Lecture Notes in Computer Science. Springer, pp. 45–57.

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., and Stützle, T. (2019). "Automatic off-line design of robot swarms: a manifesto". In: *Frontiers in Robotics and AI* 6, p. 59.

Birattari, M., Ligot, A., and Francesca, G. (2021). "AutoMoDe: a modular approach to the automatic off-line design and fine-tuning of control software for robot swarms". In: *Automated Design of Machine Learning and Search Algorithms.* Natural Computing Series. Springer, pp. 73–90.

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

Birattari, M., Stützle, T., Paquete, L., and Varrentrapp, K. (2002). "A racing algorithm for configuring metaheuristics". In: *GECCO'02: Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation.* Morgan Kaufmann Publishers, pp. 11–18.

Bongard, J. C. (2013). "Evolutionary robotics". In: *Communication ACM* 56.8, pp. 74–83.

Bongard, J. C. and Lipson, H. (2004). "Once more unto the breach: co-evolving a robot and its simulator". In: *Artificial Life IX: Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems.* MIT Press, pp. 57–62.

Bossens, D. M. and Tarapore, D. (2021). "QED: Using Quality-Environment-Diversity to Evolve Resilient Robot Swarms". In: *IEEE Transactions on Evolutionary Computation* 25.2, pp. 346–357.

Boudet, J. F., Lintuvuori, J., Lacouture, C., Barois, T., Deblais, A., Xie, K., Cassagnere, S., Tregon, B., Brückner, D., Baret, J. C., and Kellay, H. (2021). "From collections of independent, mindless robots to flexible, mobile, and directional superstructures". In: *Science Robotics* 6.56, eabd0272.

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

Brambilla, M., Ferrante, E., Birattari, M., and Dorigo, M. (2013). "Swarm robotics: a review from the swarm engineering perspective". In: *Swarm Intelligence* 7.1, pp. 1–41.

Bredeche, N., Haasdijk, E., and Prieto, A. (2018). "Embodied evolution in collective robotics: a review". In: *Frontiers in Robotics and AI* 5, p. 12.

Bredeche, N., Montanier, J.-M., Liu, W., and Winfield, A. (2012). "Environment-driven distributed evolutionary adaptation in a population of autonomous robotic agents". In: *Mathematical and Computer Modelling of Dynamical Systems* 18.1, pp. 101–129.

Brooks, R. A. (1992). "Artificial life and real robots". In: *Towards a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*. MIT Press, pp. 3–10.

Cambier, N. and Ferrante, E. (2022). "AutoMoDe-Pomodoro: an evolutionary class of modular Designs". In: *GECCO'22: Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, pp. 100–103.

Caruana, R., Lawrence, S., and Giles, C. L. (2000). "Overfitting in neural nets: backpropagation, conjugate gradient, and early stopping". In: *NIPS'00: Proceedings of the 13th International Conference on Neural Information Processing Systems*. MIT Press, pp. 402–408.

Chambers, J. M., Cleveland, W. S., Kleiner, B., and Tukey, P. A. (1983). *Graphical Methods For Data Analysis*. CRC Press.

Champandard, A. J. (2007). *Understanding Behavior Trees*. http://aigamedev.com/open/articles/bt-overview/.

Christensen, A. L. and Dorigo, M. (2006). "Evolving an integrated phototaxis and hole-avoidance behavior for a swarm-bot". In: *Artificial Life X: Proceedings of the Tenth International Conference on the Simulation and Synthesis of Living Systems*. MIT Press, pp. 248–254.

Conover, W. J. (1999). *Practical Nonparametric Statistics*. Wiley Series in Probability and Statistics. John Wiley & Sons.

Cotorruelo, A., Ligot, A., Garone, E., and 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.

Cully, A. and Demiris, Y. (2018). “Quality and Diversity Optimization: A Unifying Modular Framework”. In: *IEEE Transactions on Evolutionary Computation* 22.2, pp. 245–259.

Cully, A. and Mouret, J.-B. (2016). “Evolving a Behavioral Repertoire for a Walking Robot”. In: *Evolutionary Computation* 24.1, pp. 59–88.

Doncieux, S. and Mouret, J.-B. (2014). “Beyond black-box optimization: a review of selective pressures for evolutionary robotics”. In: *Evolutionary Intelligence* 7.2, pp. 71–93.

Dorigo, M., Birattari, M., and Brambilla, M. (2014). “Swarm robotics”. In: *Scholarpedia* 9.1, p. 1463.

Dorigo, M., Theraulaz, G., and Trianni, V. (2020). “Reflections on the future of swarm robotics”. In: *Science Robotics* 5, eabe4385.

Dorigo, M., Theraulaz, G., and Trianni, V. (2021). “Swarm robotics: past, present, and future [point of view]”. In: *Proceedings of the IEEE* 109.7, pp. 1152–1165.

Dorigo, M., Trianni, V., Şahin, E., Groß, R., Labella Thomas, H., Baldassarre, G., Nolfi, S., Deneubourg, J.-L., Mondada, F., Floreano, D., and Gambardella, L. M. (2003). “Evolving self-organizing behaviors for a Swarm-bot”. In: *Autonomous Robots* 17, pp. 223–245.

Duarte, M., Costa, V., Gomes, J., Rodrigues, T., Silva, F., Oliveira, S. M., and Christensen, A. L. (2016a). “Evolution of collective behaviors for a real swarm of aquatic surface robots”. In: *PLOS ONE* 11.3, e0151834.

Duarte, M., Gomes, J., Oliveira, S. M., and Christensen, A. L. (2016b). “EvoRBC: evolutionary repertoire-based control for robots with arbitrary locomotion complexity”. In: *GECCO’16: Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, pp. 93–100.

Duarte, M., Gomes, J., Oliveira, S. M., and Christensen, A. L. (2018). “Evolution of repertoire-based control for robots with complex locomotor systems”. In: *IEEE Transactions on Evolutionary Computation* 22.2, pp. 314–328.

Duarte, M., Oliveira, S. M., and Christensen, A. L. (2014a). “Evolution of hierarchical controllers for multirobot systems”. In: *ALIFE 14: The Fourteenth International Conference on the Synthesis and Simulation of Living Systems*. MIT Press, pp. 657–664.

Duarte, M., Oliveira, S. M., and Christensen, A. L. (2014b). “Hybrid control for large swarms of aquatic drones”. In: *ALIFE 14: The Fourteenth International Conference on the Synthesis and Simulation of Living Systems*. MIT Press, pp. 785–792.

Engebråten, S. A., Moen, J., Yakimenko, O. A., and Glette, K. (2021). "Evolving a repertoire of controllers for a multi-function swarm". In: *Applications of Evolutionary Computation: 21st International Conference, EvoApplications 2018*. Vol. 10784. Lecture Notes in Computer Science. Springer, pp. 734–749.

Fernández Pérez, I., Boumaza, A., and Charpillet, F. (2017). "Learning collaborative foraging in a swarm of robots using embodied evolution". In: *ECAL 2017, the Fourteenth European Conference on Artificial Life*. MIT Press, pp. 162–169.

Ferrante, E., Duéñez-Guzmán, E. A., Turgut, A. E., and Wenseleers, T. (2013). "GESwarm: grammatical evolution for the automatic synthesis of collective behaviors in swarm robotics". In: *GECCO'13: Proceedings of the 15th annual conference on Genetic and evolutionary computation*. ACM, pp. 17–24.

Ferrante, E., Turgut, A. E., Duéñez-Guzmán, E. A., Dorigo, M., and Wenseleers, T. (2015). "Evolution of self-organized task specialization in robot swarms". In: *PLOS Computational Biology* 11.8, e1004273.

Floreano, D., Husbands, P., and Nolfi, S. (2008). "Evolutionary robotics". In: *Springer Handbook of Robotics*. Springer Handbooks. Springer, pp. 1423–1451.

Floreano, D. and Mondada, F. (1996). "Evolution of plastic neurocontrollers for situated agents". In: *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*. MIT Press, pp. 402–410.

Floreano, D. and Urzelai, J. (2000). "Evolutionary robots with on-line self-organization and behavioral fitness". In: *Neural Networks* 13, pp. 431–443.

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

Francesca, G., Brambilla, M., Brutschy, A., Garattoni, L., Miletitch, R., Podevijn, G., Reina, A., Soleymani, T., Salvaro, M., Pinciroli, C., Mascia, F., Trianni, V., and Birattari, M. (2015). "AutoMoDe-Chocolate: automatic design of control software for robot swarms". In: *Swarm Intelligence* 9.2–3, pp. 125–152.

Francesca, G., Brambilla, M., Brutschy, A., Garattoni, L., Miletitch, R., Podevijn, G., Reina, A., Soleymani, T., Salvaro, M., Pinciroli, C., Trianni, V., and Birattari, M. (2014a). "An experiment in automatic design of robot swarms: AutoMoDe-Vanilla, EvoStick, and human experts". In: *Swarm Intelligence: 9th International Conference, ANTS 2014*. Vol. 8667. Lecture Notes in Computer Science. Springer International Publishing, pp. 25–37.

Francesca, G., Brambilla, M., Brutschy, A., Trianni, V., and Birattari, M. (2014b). "AutoMoDe: a novel approach to the automatic design of control software for robot swarms". In: *Swarm Intelligence* 8.2, pp. 89–112.

Francesca, G., Brambilla, M., Trianni, V., Dorigo, M., and Birattari, M. (2012). "Analysing an evolved robotic behaviour using a biological model of collegial decision making". In: *From Animals to Animats 12: 12th International Conference on Simulation of Adaptive Behavior, SAB 2012*. Vol. 7426. Lecture Notes in Computer Science. Springer, pp. 381–390.

Garattoni, L. and Birattari, M. (2016). "Swarm robotics". In: *Wiley Encyclopedia of Electrical and Electronics Engineering*. John Wiley & Sons, pp. 1–19.

Garattoni, L. and Birattari, M. (2018). "Autonomous task sequencing in a robot swarm". In: *Science Robotics* 3.20, eaat0430.

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

Garzón Ramos, D. and Birattari, M. (2020). "Automatic design of collective behaviors for robots that can display and perceive colors". In: *Applied Sciences* 10.13, p. 4654.

Garzón Ramos, D., Bozhinoski, D., Francesca, G., Garattoni, L., Hasselmann, K., Kegeleirs, M., Kuckling, J., Ligot, A., Mendiburu, F. J., Pagnozzi, F., Salman, M., Stützle, T., and Birattari, M. (2021). "The automatic off-line design of robot swarms: recent advances and perspectives". In: *R2T2: Robotics Research for Tomorrow's Technology*.

Gauci, M., Chen, J., Li, W., Dodd, T. J., and Groß, R. (2014a). "Clustering objects with robots that do not compute". In: *AAMAS '14: Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), pp. 421–428.

Gauci, M., Chen, J., Li, W., Dodd, T. J., and Groß, R. (2014b). "Self-organized aggregation without computation". In: *The International Journal of Robotics Research* 33.8, pp. 1145–1161.

Geman, S., Bienenstock, E., and Doursat, R. (1992). "Neural networks and the bias/variance dilemma". In: *Neural Computation* 4.1, pp. 1–58.

Glasmachers, T., Schaul, T., Yi, S., Wierstra, D., and Schmidhuber, J. (2010). "Exponential natural evolution strategies". In: *GECCO'10: Proceedings of the 12th annual conference on Genetic and evolutionary computation*. ACM, pp. 393–400.

Gomes, J. and Christensen, A. L. (2018a). "Comparing approaches for evolving high-level robot control based on behaviour repertoires". In: *2018 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, pp. 1–6.

Gomes, J. and Christensen, A. L. (2018b). “Task-agnostic evolution of diverse repertoires of swarm behaviours”. In: *Swarm Intelligence: 11th International Conference, ANTS 2018*. Vol. 11172. Lecture Notes in Computer Science. Springer, pp. 225–238.

Gomes, J., Mariano, P., and Christensen, A. L. (2014). “Systematic Derivation of Behaviour Characterisations in Evolutionary Robotics”. In: *ALIFE 14: The Fourteenth International Conference on the Synthesis and Simulation of Living Systems*. MIT Press, pp. 212–219.

Gomes, J., Urbano, P., and Christensen, A. L. (2013). “Evolution of swarm robotics systems with novelty search”. In: *Swarm Intelligence* 7.2–3, pp. 115–144.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Gutiérrez, Á., Campo, A., Dorigo, M., Donate, J., Monasterio-Huelin, F., and Magdalena, L. (2009). “Open e-puck range & bearing miniaturized board for local communication in swarm robotics”. In: *2009 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3111–3116.

Haasdijk, E., Bredeche, N., and Eiben, A. (2014). “Combining environment-driven adaptation and task-driven optimisation in evolutionary robotics”. In: *PLOS ONE* 9.6, e98466.

Hamann, H. (2018). *Swarm robotics: a formal approach*. Springer.

Hansen, N. and Ostermeier, A. (2001). “Completely derandomized self-adaptation in evolution strategies”. In: *Evolutionary Computation* 9.2, pp. 159–195.

Hasselmann, K. and Birattari, M. (2020). “Modular automatic design of collective behaviors for robots endowed with local communication capabilities”. In: *PeerJ Computer Science* 6, e291.

Hasselmann, K. and Birattari, M. (2022). *Advances in the automatic modular design of control software for robot swarms: using neuroevolution to generate modules: supplementary material*. https://iridia.ulb.ac.be/supp/IridiaSupp2022-003.

Hasselmann, K., Ligot, A., Francesca, G., Garzón Ramos, D., Salman, M., Kuckling, J., Mendiburu, F. J., and Birattari, M. (2018a). *Reference models for AutoMoDe*. Tech. rep. TR/IRIDIA/2018-002. IRIDIA, Université Libre de Bruxelles.

Hasselmann, K., Ligot, A., Ruddick, J., and Birattari, M. (2021). “Empirical assessment and comparison of neuro-evolutionary methods for the automatic off-line design of robot swarms”. In: *Nature Communications* 12, p. 4345.

Hasselmann, K., Robert, F., and Birattari, M. (2018b). “Automatic design of communication-based behaviors for robot swarms”. In: *Swarm Intelligence: 11th*

International Conference, ANTS 2018. Vol. 11172. Lecture Notes in Computer Science. Springer, pp. 16–29.

Hasselmann, K., Robert, F., and Birattari, M. (2018c). *Automatic design of communication-based behaviors for robot swarms: supplementary material*. http://iridia.ulb.ac.be/supp/IridiaSupp2018-003/.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning: Data mining, Inference and Prediction*. Springer.

Hauert, S., Zufferey, J.-C., and Floreano, D. (2009). "Evolved swarming without positioning information: an application in aerial communication relay". In: *Autonomous Robots* 26.1, pp. 21–32.

Heaton, J. (2008). *Introduction to Neural Networks for Java*. Heaton Research.

Hecker, J. P., Letendre, K., Stolleis, K., Washington, D., and Moses, M. E. (2012). "Formica ex machina: ant swarm foraging from physical to virtual and back again". In: *Swarm Intelligence: 8th International Conference, ANTS 2012*. Vol. 7461. Lecture Notes in Computer Science. Springer, pp. 252–259.

Hettiarachchi, S. and Spears, W. M. (2009). "Distributed adaptive swarm for obstacle avoidance". In: *International Journal of Intelligent Computing and Cybernetics* 2.4, pp. 644–671.

Jakobi, N. (1997). "Evolutionary robotics and the radical envelope-of-noise hypothesis". In: *Adaptive Behavior* 6.2, pp. 325–368.

Jakobi, N., Husbands, P., and Harvey, I. (1995). "Noise and the reality gap: the use of simulation in evolutionary robotics". In: *Advances in Artificial Life: Third European Conference on Artificial Life*. Vol. 929. Lecture Notes in Artificial Intelligence. Springer, pp. 704–720.

Jones, S., Studley, M., Hauert, S., and Winfield, A. (2018). "Evolving behaviour trees for swarm robotics". In: *Distributed Autonomous Robotic Systems: The 13th International Symposium*. Vol. 6. Springer Proceedings in Advanced Robotics. Springer, pp. 487–501.

Jones, S., Winfield, A., Hauert, S., and Studley, M. (2019). "Onboard evolution of understandable swarm behaviors". In: *Advanced Intelligent Systems* 1.6, p. 1900031.

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

Kim, S. and Doncieux, S. (2017). "Learning highly diverse robot throwing movements through quality diversity search". In: *GECCO'17: Proceedings of the Genetic and Evolutionary Computation Conference*. Morgan Kaufmann Publishers, pp. 1177–1178.

König, L., Mostaghim, S., and Schmeck, H. (2009). “Decentralized evolution of robotic behavior using finite state machines”. In: *International Journal of Intelligent Computing and Cybernetics* 2.4, pp. 695–723.

Koos, S., Mouret, J.-B., and Doncieux, S. (2013). “The transferability approach: crossing the reality gap in evolutionary robotics”. In: *IEEE Transactions on Evolutionary Computation* 17.1, pp. 122–145.

Kuckling, J., Hasselmann, K., Pelt, V. van, Kiere, C., and Birattari, M. (2021). *AutoMoDe Editor: a visualization tool for AutoMoDe.* Tech. rep. TR/IRIDIA/2021-009. IRIDIA, Université Libre de Bruxelles.

Kuckling, J., Ligot, A., Bozhinoski, D., and Birattari, M. (2018). “Behavior trees as a control architecture in the automatic modular design of robot swarms”. In: *Swarm Intelligence: 11th International Conference, ANTS 2018.* Vol. 11172. Lecture Notes in Computer Science. Springer, pp. 30–43.

Kuckling, J., Stützle, T., and Birattari, M. (2020a). “Iterative improvement in the automatic modular design of robot swarms”. In: *PeerJ Computer Science* 6, e322.

Kuckling, J., Ubeda Arriaza, K., and Birattari, M. (2019). “Simulated annealing as an optimization algorithm in the automatic modular design of robot swarms”. In: *Proceedings of the Reference AI & ML Conference for Belgium, Netherlands & Luxemburg, BNAIC/BENELEARN 2019.* Vol. 2491. CEUR Workshop Proceedings.

Kuckling, J., Ubeda Arriaza, K., and Birattari, M. (2020b). “AutoMoDe-IcePop: automatic modular design of control software for robot swarms using simulated annealing”. In: *Artificial Intelligence and Machine Learning: BNAIC 2019, BENELEARN 2019.* Vol. 1196. Communications in Computer and Information Science. Springer, pp. 3–17.

Lehman, J. and Stanley, K. O. (2011a). “Abandoning objectives: evolution through the search for novelty alone”. In: *Evolutionary Computation* 19.2, pp. 189–223.

Lehman, J. and Stanley, K. O. (2011b). “Evolving a diversity of virtual creatures through novelty search and local competition”. In: *GECCO’11: Proceedings of the 13th annual conference on Genetic and evolutionary computation.* ACM, pp. 211–218.

Li, S., Batra, R., Brown, D., Chang, H.-D., Ranganathan, N., Hoberman, C., Rus, D., and Lipson, H. (2019). “Particle robotics based on statistical mechanics of loosely coupled components”. In: *Nature* 567.7748, pp. 361–365.

Ligot, A. and Birattari, M. (2018). “On mimicking the effects of the reality gap with simulation-only experiments”. In: *Swarm Intelligence: 11th International Con-*

ference, ANTS 2018. Vol. 11172. Lecture Notes in Computer Science. Springer, pp. 109–122.

Ligot, A. and Birattari, M. (2020). “Simulation-only experiments to mimic the effects of the reality gap in the automatic design of robot swarms”. In: *Swarm Intelligence* 14, pp. 1–24.

Ligot, A. and Birattari, M. (2022). “On Using Simulation to Predict the Performance of Robot Swarms”. In: *Scientific Data* 9.788.

Ligot, A., Cotorruelo, A., Garone, E., and Birattari, M. (2022). “Towards an empirical practice in off-line fully-automatic design of robot swarms”. In: *IEEE Transactions on Evolutionary Computation*.

Ligot, A., Hasselmann, K., and Birattari, M. (2020a). “AutoMoDe-Arlequin: neural networks as behavioral modules for the automatic design of probabilistic finite state machines”. In: *Swarm Intelligence: 12th International Conference, ANTS 2020*. Vol. 12421. Lecture Notes in Computer Science. Springer, pp. 109–122.

Ligot, A., Hasselmann, K., Delhaisse, B., Garattoni, L., Francesca, G., and 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.

Ligot, A., Kuckling, J., Bozhinoski, D., and Birattari, M. (2020b). “Automatic modular design of robot swarms using behavior trees as a control architecture”. In: *PeerJ Computer Science* 6, e314.

Lopes, Y. K., Leal, A. B., Dodd, T. J., and Groß, R. (2014). “Application of supervisory control theory to swarms of e-puck and Kilobot robots”. In: *Swarm Intelligence: 9th International Conference, ANTS 2014*. Vol. 8667. Lecture Notes in Computer Science. Springer, pp. 62–73.

Lopes, Y. K., Trenkwalder, S. M., Leal, A. B., Dodd, T. J., and Groß, R. (2016). “Supervisory control theory applied to swarm robotics”. In: *Swarm Intelligence* 10.1, pp. 65–97.

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., and Stützle, T. (2016). “The irace package: iterated racing for automatic algorithm configuration”. In: *Operations Research Perspectives* 3, pp. 43–58.

Lunacek, M. and Whitley, L. D. (2006). “The dispersion metric and the CMA evolution strategy”. In: *GECCO’06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*. ACM, pp. 477–484.

Mayor, A. (2018). *Gods and Robots: Myths, Machines, and Ancient Dreams of Technology*. Princeton University Press.

Mendiburu, F. J., Garzón Ramos, D., Morais, M. R. A., Lima, A. M. N., and Birattari, M. (2022). “AutoMoDe-Mate: automatic off-line design of spatially-organizing behaviors for robot swarms”. In: *Swarm and Evolutionary Computation* 74, p. 101118.

Miglino, O., Lund, H. H., and Nolfi, S. (1995). “Evolving mobile robots in simulated and real environments”. In: *Artificial Life* 2.4, pp. 417–434.

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

Montanier, J.-M., Carrignon, S., and Bredeche, N. (2016). “Behavioural specialization in embodied evolutionary robotics: why so difficult?” In: *Frontiers in Robotics and AI* 3.38, pp. 1–11.

Morgan, N. and Bourlard, H. (1989). “Generalization and parameter estimation in feedforward nets: some experiments”. In: *NIPS’89: Proceedings of the 2nd International Conference on Neural Information Processing Systems*. Morgan Kaufmann Publishers, pp. 630–637.

Mouret, J.-B. and Clune, J. (2015). *Illuminating search spaces by mapping elites*. http://arxiv.org/abs/1504.04909.

Neupane, A. and Goodrich, M. (2019). “Learning swarm behaviors using grammatical evolution and behavior trees”. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. IJCAI Organization, pp. 513–520.

Nikolaev, A. G. and Jacobson, S. H. (2010). “Simulated annealing”. In: *Handbook of metaheuristics*. Vol. 146. International Series in Operations Research & Management Science. Springer, pp. 1–39.

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

Nolfi, S. and Floreano, D. (2000). *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press.

Nolfi, S., Floreano, D., Miglino, O., and Mondada, F. (1994). “How to evolve autonomous robots: different approaches in evolutionary robotics”. In: *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*. MIT Press, pp. 190–197.

O’Neill, M. and Ryan, C. (2003). *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Genetic Programming Series. Springer.

Pagliuca, P. and Nolfi, S. (2019). “Robust optimization through neuroevolution”. In: *PLOS ONE* 14.3, e0213193.

Pinciroli, C. and Beltrame, G. (2016). “Buzz: a programming language for robot swarms”. In: *IEEE Software* 33.4, pp. 97–100.

Pinciroli, C., Trianni, V., O’Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G. A., Ducatelle, F., Birattari, M., Gambardella, L. M., and Dorigo, M. (2012). “ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems”. In: *Swarm Intelligence* 6.4, pp. 271–295.

Prechelt, L. (2012). “Early stopping — but when?” In: *Neural Networks: Tricks of the Trade: Second Edition*. Vol. 7700. Lecture Notes in Computer Science. Springer, pp. 53–67.

Pugh, J. and Martinoli, A. (2009). “Distributed scalable multi-robot learning using particle swarm optimization”. In: *Swarm Intelligence* 3.3, pp. 203–222.

Pugh, J., Martinoli, A., and Zhang, Y. (2005). “Particle swarm optimization for unsupervised robotic learning”. In: *2005 IEEE Swarm Intelligence Symposium, SIS 2005*. IEEE, pp. 92–99.

Pugh, J. K., Soros, L. B., and Stanley, K. O. (2016). “Quality Diversity: a new frontier for evolutionary computation”. In: *Frontiers in Robotics and AI* 3, p. 40.

Quinn, M., Smith, L., Mayley, G., and Husbands, P. (2003). “Evolving controllers for a homogeneous system of physical robots: structured cooperation with minimal sensors”. In: *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 361.1811, pp. 2321–2343.

Raskutti, G., Wainwright, M. J., and Yu, B. (2014). “Early stopping and nonparametric regression: an optimal data-dependent stopping rule”. In: *Journal of Machine Learning Research* 15, pp. 335–366.

Reina, A., Miletitch, R., Dorigo, M., and Trianni, V. (2015a). “A quantitative micro–macro link for collective decisions: the shortest path discovery/selection example”. In: *Swarm Intelligence* 9.2–3, pp. 75–102.

Reina, A., Valentini, G., Fernández-Oto, C., Dorigo, M., and Trianni, V. (2015b). “A design pattern for decentralised decision making”. In: *PLOS ONE* 10.10, e0140950.

Rubenstein, M., Cornejo, A., and Nagpal, R. (2014). “Programmable self-assembly in a thousand-robot swarm”. In: *Science* 345.6198, pp. 795–799.

Russell, S. J. and Norvig, P. (2020). *Artificial Intelligence: a modern approach*. Pearson Education, Inc.

Schranz, M., Di Caro, G. A., Schmickl, T., Elmenreich, W., Arvin, F., Şekercioğlu, Y. A., and Sende, M. (2021). “Swarm intelligence and cyber-physical systems: concepts, challenges and future trends”. In: *Swarm and Evolutionary Computation* 60.14, p. 100762.

Schranz, M., Umlauft, M., Sende, M., and Elmenreich, W. (2020). “Swarm robotic behaviors and current applications”. In: *Frontiers in Robotics and AI* 7, p. 36.

Silva, F., Duarte, M., Correia, L., Oliveira, S. M., and Christensen, A. L. (2016). “Open issues in evolutionary robotics”. In: *Evolutionary Computation* 24.2, pp. 205–236.

Silva, F., Urbano, P., Correia, L., and Christensen, A. L. (2015). “odNEAT: an algorithm for decentralised online evolution of robotic controllers”. In: *Evolutionary Computation* 23.3, pp. 421–449.

Slavkov, I., Carrillo-Zapata, D., Carranza, N., Diego, X., Jansson, F., Kaandorp, J., Hauert, S., and Sharpe, J. (2018). “Morphogenesis in robot swarms”. In: *Science Robotics* 3.25, eaau9178.

Spaey, G., Kegeleirs, M., Garzón Ramos, D., and Birattari, M. (2020). “Evaluation of alternative exploration schemes in the automatic modular design of robot swarms”. In: *Artificial Intelligence and Machine Learning: BNAIC 2019, BENELEARN 2019*. Vol. 1196. Communications in Computer and Information Science. Springer, pp. 18–33.

Spears, W. M., Spears, D., Hamann, J. C., and Heil, R. (2004). “Distributed, physics-based control of swarms of vehicles”. In: *Autonomous Robots* 17.2, pp. 137–162.

Stanley, K. O. and Miikkulainen, R. (2002). “Evolving neural networks through augmenting topologies”. In: *Evolutionary Computation* 10.2, pp. 99–127.

Stranieri, A., Turgut, A. E., Salvaro, M., Garattoni, L., Francesca, G., Reina, A., Dorigo, M., and Birattari, M. (2013). *IRIDIA’s arena tracking system*. Tech. rep. TR/IRIDIA/2013-013. IRIDIA, Université Libre de Bruxelles.

Trianni, V. (2008). *Evolutionary Swarm Robotics*. Springer.

Trianni, V. and Dorigo, M. (2006). “Self-organisation and communication in groups of simulated and physical robots”. In: *Biological Cybernetics* 95, pp. 213–231.

Trianni, V. and Nolfi, S. (2009). “Self-organizing sync in a robotic swarm: a dynamical system view”. In: *IEEE Transactions on Evolutionary Computation* 13.4, pp. 722–741.

Usui, Y. and Arita, T. (2003). “Situated and embodied evolution in collective evolutionary robotics”. In: *Proceedings of the 8th International Symposium on*

Artificial Life and Robotics. International Society of Artificial Life and Robotics (ISAROB), pp. 212–215.

Vassiliades, V., Chatzilygeroudis, K., and Mouret, J.-B. (2018). “Using Centroidal Voronoi Tessellations to Scale Up the Multidimensional Archive of Phenotypic Elites Algorithm”. In: *IEEE Transactions on Evolutionary Computation* 22.4, pp. 623–630.

Waibel, M., Keller, L., and Floreano, D. (2009). “Genetic team composition and level of selection in the evolution of multi-agent systems”. In: *IEEE Transactions on Evolutionary Computation* 13.3, pp. 648–660.

Watson, R. A., Ficici, S. G., and Pollack, J. B. (1999). “Embodied evolution: embodying an evolutionary algorithm in a population of robots”. In: *1999 Congress on Evolutionary Computation, CEC99*. Vol. 1. IEEE, pp. 335–342.

Watson, R. A., Ficici, S. G., and Pollack, J. B. (2002). “Embodied evolution: distributing an evolutionary algorithm in a population of robots”. In: *Robotics and Autonomous Systems* 39.1, pp. 1–18.

Werfel, J., Petersen, K., and Nagpal, R. (2014). “Designing collective behavior in a termite-inspired robot construction team”. In: *Science* 343.6172, pp. 754–758.

Wolpert, D. (1997). “On bias plus variance”. In: *Neural Computation* 9, pp. 1211–1243.

Xie, H., Sun, M., Fan, X., Lin, Z., Chen, W., Wang, L., Dong, L., and He, Q. (2019). “Reconfigurable magnetic microrobot swarm: multimode transformation, locomotion, and manipulation”. In: *Science Robotics* 4.28, eaav8006.

Xin, Z., Xiangyong, W., Zhepei, W., Yuman, G., Haojia, L., Qianhao, W., Tiankai, Y., Haojian, L., Yanjun, C., Chao, X., and Fei, G. (2022). “Swarm of micro flying robots in the wild”. In: *Science Robotics* 7.66, eabm5954.

Yu, J., Wang, B., Du, X., Wang, Q., and Zhang, L. (2018). “Ultra-extensible ribbon-like magnetic microswarm”. In: *Nature Communications* 9.1, p. 3260.

Zagal, J. C. and Ruiz-del-Solar, J. (2007). “Combining simulation and reality in evolutionary robotics”. In: *Journal of Intelligent & Robotic Systems* 50.1, pp. 19–39.