ULB

ECOLE
POLYTECHNIQUE
DE BRUXELLES

Leveraging environmental and inter-robot signaling in the automatic modular design of robot swarms

Communication, reaction to events, and sequential missions

Thesis presented by David GARZÓN RAMOS

in fulfilment of the requirements of the PhD Degree in Engineering Sciences and Technology ("Docteur en Sciences de l'ingénieur et technologie")

Année académique 2024-2025

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

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. Thomas STÜTZLE (secretary)

Research Director of the FRS-FNRS at IRIDIA, École polytechnique de Bruxelles, Université libre de Bruxelles, Belgium.

Prof. Sabine HAUERT

Professor of the School of Engineering Mathematics and Technology and head of Hauert Lab, Bristol Robotics Laboratory, University of Bristol, United Kingdom.

Prof. Carlo PINCIROLI

Associate Professor of the Robotics Engineering department and director of NEST Lab, Worcester Polytechnic Institute, United States of America.

Prof. Mauro BIRATTARI (supervisor)

Research Director of the FRS-FNRS at IRIDIA, École polytechnique de Bruxelles, Université libre de Bruxelles, Belgium.

Research aim

Signaling is a key capability in collective behaviors for robot swarms. Yet, the current literature on the automatic design of robot swarms does not provide an approach that effectively leverages signaling capabilities in a generally applicable way.

In this thesis, I propose that automatic modular design (AutoMoDe) is a suitable approach to addressing this gap. I aim to show that AutoMoDe can leverage environmental and inter-robot signaling to automatically generate the control software necessary for robot swarms to operate effectively.

I investigate this issue in the context of design problems where a robots must cooperate, react to events, and perform missions sequentially. The presented research shows that AutoMoDe can effectively leverage signaling to address a wide range of design problems, including designing spatially-organizing behaviors, stigmergy-based behaviors, shepherding behaviors, and the design of robot swarms by demonstration.

Summary

Robot swarms are self-organizing groups of robots that can cooperate to accomplish tasks beyond their individual capabilities. Swarm robotics offers a promising solution for coordinating large groups of robots and enabling future large-scale robotic services. However, designing robot swarms is inherently difficult because they cannot be programmed directly. At design time, one must produce control software to program the individual actions of the robots. At deployment time, the collective behavior of the swarm will emerge from the interactions between robots, and robots and their environment. The difficulty is that there is no generally applicable methodology to determine in advance how to program individual robots to achieve a specific collective behavior after deployment.

Automatic modular design (AutoMoDe) is an appealing approach to overcome the difficulty of realizing robot swarms. In this approach, the designer specifies a mission for the swarm, and an automatic method generates the control software that the robots require to collectively perform the mission. This automatic process is driven by an optimization algorithm that fine-tunes parametric software modules and assembles them into a modular control architecture.

In this dissertation, we investigate whether AutoMoDe can establish meaningful interactions between robots, and robots and their environment, through signal-based interactions. Our research shows that AutoMoDe can leverage signaling capabilities if these are provided by the parametric software modules on which AutoMoDe operates. We experiment with robots that can perceive, display, and respond to color cues that convey information about the environment and about the actions/state of other robots. In our studies, AutoMoDe proved capable of using signaling to establish interaction strategies for missions requiring the swarm to cooperate, collectively react to events, and perform tasks sequentially. Previous research on the automatic design of robot swarms has independently utilized

signaling to address related problems, without evaluating the general applicability of the proposed methods. In this thesis, we overcome this empirical limitation and show that AutoMoDe is a versatile approach, offering roboticists a reliable, optimization-driven method to design collective behaviors for robot swarms.

The thesis developed in this dissertation is supported by empirical evidence. We present diverse scenarios in which automatically designed robot swarms rely on mission-specific signaling to perform their mission. In these experiments, we consider missions on which the performance of the swarm is evaluated by a single or by concurrent performance measures. Notably, we apply these ideas to the automatic design of spatially-organizing behaviors, stigmergy-based behaviors, shepherding behaviors, and the design of robot swarms by demonstration. In these studies, we show that a simple single-bit signaling protocol embedded in a specialized module for AutoMoDe was sufficient to overcome previous limitations in designing spatially organizing behaviors for robot swarms. Furthermore, we show that AutoMoDe can leverage both direct communication capabilities and indirect communication using pheromone-based stigmergy. We illustrate how AutoMoDe's ability to leverage signaling not only facilitates communication within the swarm but also enables interactions with other active agents in the swarm's workspace. We also show that AutoMoDe can conduct the design process by learning from demonstrations of the desired collective behavior.

In our research, we addressed these diverse design problems using AutoMoDe methods that had no fundamental differences from each other. Through this thesis, we learned how to use, adapt, and extend AutoMoDe to tackle diverse design problems effectively. As a result, with this dissertation, our aim is also to provide a reference guide on how to use AutoMoDe and capitalize on its general applicability.

Contributions

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

Critical review of the swarm robotics literature: We provide a critical review of the state of the art in the automatic modular design of robot swarms, highlighting common practices in selecting robot capabilities and defining missions for the robots.

Demonstrations of automatically designed collective behaviors: We present the most diverse range of missions and applications of the automatic off-line design of robot swarms currently available in the literature, supported by simulations and experiments with physical robots.

`TuttiFrutti`: We introduce `TuttiFrutti`, a novel modular automatic design method that enables the design of collective behaviors for robots that can display and perceive color signals.

`Mandarina`: We introduce `Mandarina`, a novel modular automatic design method capable of handling concurrent design criteria during the design process.

`Mate`, `Habanero`, `Pistacchio`, `DTF-MO`: We build on the original ideas introduced with `Mandarina` and `TuttiFrutti` to demonstrate the versatility of the automatic modular design approach in four new specialized design methods.

Software implementations: We implemented and maintain the software for the methods presented in the thesis. The software produced is available as open source software on the public repository of the DEMIURGE project.

Hardware implementations: We developed new infrastructure to conduct experiments with swarms of e-pucks. Notably, `MoCA`, a RGB modular arena, and Tycho, a robust ROS-based tracking system to monitor robot swarms. The hardware produced is available as open source hardware on the public repository of the DEMIURGE project.

Dissemination and public engagement: We contributed to the vulgarization and dissemination of swarm robotics and the automatic design of robot swarms. We focused our efforts on producing engaging multimedia demonstrations that feature the research presented in this thesis.

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.

The *dagger symbol* (†) signifies that the authors contributed equally and should be regarded as co-first authors.

The ideas, motivations, and state of the art presented in Chapters 1 and 2 are partially based on:

- 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.

- **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.*

The definition and the analysis of the automatic design method `TuttiFrutti` presented in Chapter 3 are based on:

- **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.

The definition and the analysis of the automatic design method `Mandarina` presented in Chapter 4 are based on:

- **Garzón Ramos, D.**, Pagnozzi, F., Stützle, T., and Birattari, M. (2024). "Automatic design of robot swarms under concurrent design criteria: a study based on Iterated F-Race". In: *Advanced Intelligent Systems*, p. 2400332.

The experiments and methods presented in Chapter 5 are based on:

- 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.

- Salman, M.†, **Garzón Ramos, D.**†, and Birattari, M. (2024). "Automatic design of stigmergy-based behaviours for robot swarms". In: *Communications Engineering* 3, p. 30.

- **Garzón Ramos, D.** and Birattari, M. (2024). "Automatically designing robot swarms in environments populated by other robots: an experiment in robot shepherding". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 12240–12247.

- Szpirer, J., **Garzón Ramos, D.**, and Birattari, M. (2024). "Automatic design of robot swarms that perform composite missions: an approach based on inverse reinforcement learning". In: *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 5791–5798.

Implementations of software and hardware used throughout the research are released as free and open source, and are described in:

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

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

- **Garzón Ramos, D.**, Salman, M., Ubeda Arriaza, K., Hasselmann, K., and Birattari, M. (2022). *MoCA: a modular RGB color arena for swarm robotics experiments*. Tech. rep. TR/IRIDIA/2022-014. IRIDIA, Université Libre de Bruxelles.

- Legarda Herranz, G., **Garzón Ramos, D.**, Kuckling, J., Kegeleirs, M., and Birattari, M. (2022). *Tycho: a robust, ROS-based tracking system for robot swarms*. Tech. rep. TR/IRIDIA/2022-009. IRIDIA, Université Libre de Bruxelles.

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

- Kegeleirs, M., **Garzón Ramos, D.**, and Birattari, M. (2019). "Random walk exploration for swarm mapping". In: *Towards Autonomous Robotic Systems: 20th Annual Conference, TAROS 2019*. Vol. 11650. Lecture Notes in Computer Science. Springer, pp. 211–222.

- Spaey, G., Kegeleirs, M., **Garzón Ramos, D.**, and Birattari, M. (2019). "Comparison of different exploration schemes 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.

- 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.

- **Garzón Ramos, D.**, Bolaños, J. P., Diaz, J., Pachajoa, G., and Birattari, M. (2021). "Introduciendo la robótica de enjambres a entusiastas de la robótica: experiencias y resultados de una colaboración académica". In: *I Congreso Internacional de la Sociedad de Doctores e Investigadores de Colombia (SOPHIC 2021): la ciencia al servicio de la sociedad*. Editorial SoPhIC, pp. 46–48.

- Kegeleirs, M., Todesco, R., **Garzón Ramos, D.**, Legarda Herranz, G., and Birattari, M. (2022). *Mercator: hardware and software architecture for*

experiments in swarm SLAM. Tech. rep. TR/IRIDIA/2022-012. IRIDIA, Université Libre de Bruxelles.

- Gharbi, I., Kuckling, J., **Garzón Ramos, D.**, and Birattari, M. (2023). "Show me what you want: inverse reinforcement learning to automatically design robot swarms by demonstration". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 5063–5070.

- Kegeleirs, M.†, **Garzón Ramos, D.**†, Garattoni, L., Francesca, G., and Birattari, M. (2023). "Automatic off-line design of robot swarms: exploring the transferability of control software and design methods across different platforms". In: *ICRA 2023 Transferability in Robotics Workshop*. EU Horizon project euRobin.

- Kegeleirs, M.†, **Garzón Ramos, D.**†, Hasselmann, K., Garattoni, L., Francesca, G., and Birattari, M. (2024). "Transferability in the automatic off-line design of robot swarms: from sim-to-real to embodiment and design-method transfer across different platforms". In: *IEEE Robotics and Automation Letters* 9.3, pp. 2758–2765.

- Kegeleirs, M.†, **Garzón Ramos, D.**†, Legarda Herranz, G.†, Gharbi, I.†, Szpirer, J.†, Hasselmann, K., Garattoni, L., Francesca, G., and Birattari, M. (2024). "Leveraging swarm capabilities to assist other systems". In: *ICRA 2024 Breaking Swarm Stereotypes Workshop*. EU EIC project EMERGE.

- Kegeleirs, M., **Garzón Ramos, D.**, and Birattari, M. (2024). "DeimOS: a ROS-ready operating system for the e-puck". In: *Journal of Open Research Software*, (under review).

- Kegeleirs, M.†, **Garzón Ramos, D.**†, Legarda Herranz, G.†, Gharbi, I.†, Szpirer, J.†, Debeir, O., Garattoni, L., Francesca, G., and Birattari, M. (2024). "Collective perception for tracking people with a robot swarm". In: *40th Anniversary of the IEEE Conference on Robotics and Automation (ICRA@40)*, pp. 1292–1294.

Acknowledgments

The project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (DEMIURGE Project, grant agreement No 681872), from Belgium's Wallonia-Brussels Federation through the ARC Advanced project GbO–Guaranteed by Optimization, and from the Belgian Fonds de la Recherche Scientifique–FNRS through the crédit d'équipement SwarmSim and the crédit de recherche SwarmUp. I also acknowledge support from the Colombian Ministry of Science, Technology and Innovation–Minciencias.

I would like to express my deepest gratitude to Prof. Mauro Birattari for accompanying me on this journey. Mauro, you taught me that while conducting experiments leads to scientific discovery, it is clear and compelling communication that truly makes an impact in science. Thank you for being a patient mentor and for supporting me as I searched for my own path as a scientist.

I also want to devote words of gratitude to the researchers who I had the fortune to collaborate with in the development of this thesis. Prof. Thomas Stützle, Dr. Federico Pagnozzi, Dr. Fernando Mendiburu, Dr. Muhammad Salman, and Jeanne Szpirer—thank you for devoting your time and energy to materializing these ideas. Science is a team effort, and thanks to you, I was part of the best team ever.

I wish to also thank Prof. Marco Dorigo and Prof. Hugues Bersini for hosting my research at IRIDIA. You have put together an awesome lab, filled with the most brilliant and charismatic researchers. I cannot fully express how proud I am to be IRIDIAn, and how happy I am knowing that I will always have a place at Building C, fifth floor. This, of course, is thanks to all the IRIDIAns I crossed paths with, who became my family when I moved to Brussels: Alberto, Christian, Volker, Harry, Michael, Nina, MK, Garazi, Imene, Sarah, Marcolino, Leslie, DJ,

Nicolas, Alex, Moises, Cedric, Guillaume, Gio, Raina, Yating, Yunshuang, Gian, Lorenzo, Julian, Raphael, Miriam, Lluc, Thomas, Carlotta, and Muriel, and the many visiting researchers and people I might be missing here. Thanks Alberto for being a friend and for always being there when I really needed a beer.

Throughout my PhD, I also had the opportunity to work side by side with the DEMIURGErs: Jonas, Kenny, Fed, Antoine, Miquel, Salman, Guillermo, Fernando, Ilyes, Jeanne, and Darko. I truly believe we built something great together: Go AutoMoDe! Special thanks to Jonas and Miquel for putting up with all the random ideas I threw your way over the years—and I am sorry for making you work on Christmas Eve. You are the best. Jonas, there is no value in words to describe how much it meant to share with you the ups and downs of becoming a researcher.

During my time at IRIDIA, I had the great fortune to supervise many students. Thank you to all of you, especially Jazmin, Juan Pablo, Gabriel, and Ernesto, for working with me even from the other side of the world.

A few months ago, I moved to the UK to take a new step in my scientific career with Prof. Sabine Hauert. Thank you, Sabine, for being kind, supportive, and nurturing during these last months, and thank you to all the Bristol swarmies for being so welcoming. I would also like to give special thanks to my collaborators at Universidad de Nariño – Colombia, Escuela Superior Politécnica del Litoral – Ecuador, and FARI – Belgium.

Finally, to my family, you are the reason and the motivation behind all of this. To everyone I have not explicitly mentioned here but who became an intrinsic part of this journey—thank you!

Contents

List of Figures

List of Tables

1. Introduction

We are becoming increasingly familiar with robots that can perform tasks in a wide range of domains. Think, for example, of a lawn mower robot, an autonomous vacuum cleaner, or a flying drone for leisure photography. Today, these robots are mostly limited to operating as individual solutions. Soon, cooperation between robots will play a major role in transforming these solutions into large-scale robotics services.

The problem is that programming robots to work together remains a challenging task that demands the expertise of skilled designers. However, this process can be supported by automatic tools that help with the design, programming, and deployment of the robots. In this dissertation, we advance the development of automatic methods for producing control software for robots that must to operate cooperatively. Specifically, we focus on the design of groups of robots that use signaling strategies to coordinate with each other.

A robot swarm (Beni 2005; Şahin 2005) is a self-organized group of robots that, by working together, can collectively perform missions beyond the capabilities of individual robots. A particularity of robot swarms is that the robots operate autonomously without relying on a leader robot or on external infrastructure. The collective behavior of a robot swarm—and hence the swarm's ability to accomplish a particular mission—results solely from the interactions that the robots have with the environment and with their peers (Dorigo et al. 2014). The problem is that, as of today, conceiving and implementing a collective behavior for a robot swarm is challenging. The desired collective behavior for the robots is specified globally for the swarm, but this behavior cannot be programmed directly. At design time, one must produce control software to program the individual actions of the robots. At deployment time, the collective behavior of the swarm will emerge from the interactions between robots, and robots and their environment. The challenge is that no generally applicable method exists to tell what an individual robot should do so that the desired collective behavior is obtained in the swarm (Brambilla et al. 2013; Dorigo et al. 2020, 2021).

Swarm robotics (Dorigo et al. 2014; Hamann 2018) emerged therefore as the study of how to design robot swarms. The field's seminal work is often dated to 2005 (Şahin 2005; Beni 2005). Since then, swarm robotics has attracted attention in the scientific community, with numerous papers published in leading journals such as Nature, Science, Science Robotics, and similar venues (Rubenstein et al. 2014; Werfel et al. 2014; Garattoni and Birattari 2018; Slavkov et al. 2018; Yu et al. 2018; Li et al. 2019; Xie et al. 2019; Hasselmann et al. 2021; Talamali et al. 2021; Castelló Ferrer et al. 2021; Strobel et al. 2023; Salman et al. 2024; Zhu et al. 2024). Indeed, the design of robot swarms has been identified as one of the major robotics challenges to be addressed in the upcoming years (Yang et al. 2018). Recent discussions have foreseen the milestones that would drive the advance of swarm robotics (Dorigo et al. 2020, 2021): (i) the appearance of novel robot platforms that can operate in unstructured and dynamic environments (Berlinger et al. 2021); (ii) the development of new methodologies for the design of collective behaviors (Mathews et al. 2017); (iii) new opportunities to exploit emergence (Garattoni and Birattari 2018); and (iv) the shift of focus towards applications suited for large groups of coordinated robots (Hunt and Hauert 2020)—e.g., precision agriculture, ecological monitoring, and city cleaning. Although the future is promising, the reality is that at present most achievements in swarm robotics research still occur under controlled laboratory conditions (Hamann et al. 2020).

There is a need for design methodologies that will enable the transition from

laboratory experiments to real-world applications (Hunt and Hauert 2020; Jones et al. 2020). Today, researchers promote the adoption of engineering principles in the realization of robot swarms (Winfield et al. 2005; Brambilla et al. 2013; Bozhinoski and Birattari 2018; Birattari et al. 2019). Traditionally, the design process has an iterative nature and is based on manual trial and error: a human designer manually refines the control software of the individual robots until the desired collective behavior emerges—see, for example, St-Onge et al. (2020). This procedure is costly, time-consuming, and does not guarantee that the results are reproducible. Reproducibility is a fundamental attribute to be achieved in the design of robot swarms—as in all branches of engineering. It ensures that the outcomes of the design process are reliable and predictable, regardless of the designer's subjective decisions, level of expertise, or any random variations in the process. In research, reproducibility is a cornerstone of scientific validity (Baker 2016), as methods that produce reproducible results are more likely to gain acceptance and be further developed by the scientific community. A lack of reproducibility in the proposed design methodologies can limit the practical application of swarm robotics in real-world scenarios, where consistent and reliable performance is paramount.

Optimization-based design is an alternative approach to the design of collective behaviors for robot swarms (Birattari et al. 2020). In this approach, an optimization algorithm explores possible instances of control software for the robots and selects the one that maximizes performance on the specific mission at hand—according to a given performance measure. The computer-based nature of the approach improves the reproducibility achievable in the design process. Being performed by a machine, it reduces the influence of the designer's subjective decisions, level of expertise, and any biases that may be manually introduced into the design process. Optimization-based methods can be categorized with respect to different criteria. Common classifications divide them into (i) on-line and off-line methods, and into (ii) semi-automatic and (fully) automatic ones (Birattari et al. 2020). When the control software is produced or refined while the robot operates in the target environment, the method is referred to as on-line. When the control software is generated before deployment, typically in simulation, the method is referred to as off-line.[1] In semi-automatic methods, a human designer operates an optimization algorithm that serves as their primary design tool. On the contrary, automatic methods do not require any human intervention during the design process. Although

[1] This use of *on-line* and *off-line* is consistent with other literature in swarm robotics (Brambilla et al. 2013; Birattari et al. 2019, 2020; Kuckling 2023b). Other domains might use these terms in a different sense.

these classifications are not to be considered as strict—indeed, hybrids exist—they are convenient to appreciate the relative merits of different methods and to properly define expectations on their performance (Birattari et al. 2020).

Designers of robot swarms will benefit from developments on on-line/off-line and automatic/semi-automatic methods. However, it is our contention that automatic off-line methods will play a central role (Birattari et al. 2019). Semi-automatic methods are a useful and promising tool but are labor-intensive and require human intervention during the design process. On-line methods are appropriate to refine existing control software for robot swarms—limitations exists that restrict their general applicability. For example, they can explore a relatively small search space, could produce sub-optimal control software that could damage robots and environment (notably in the early phases of the design process), and are applicable only when the robots can assess their own collective performance. This last limitation is particularly significant. Swarms are typically unable to assess their performance because individual robots rely on local perception and lack awareness of the global state of the swarm. Without this information, individual robots cannot evaluate the swarm's performance. On-line methods are applicable in design problems where robots can obtain a reliable measure of their instantaneous collective performance—making on-line methods unsuitable in the general case. Off-line methods, by contrast, do not face this restriction. They are more generally applicable and have the potential to produce control software for robot swarms faster, with reduced human effort, while helping to ensure that the swarm's performance meets specifications and constraints. In this thesis, we make a contribution to the automatic design of robot swarms, and the research presented here can also support developments in the semi-automatic case. Although we believe that automatic off-line design addresses the problem of designing robot swarms in a more general way, in the long term, we expect that on-line, off-line, semi-automatic and automatic methods will coexist (Birattari et al. 2020; Xie et al. 2021)—hybrid methods could be particularly appealing and appropriate in many applications.

Whereas we deem it the most promising approach, automatic off-line design is not itself free from presenting challenges and open issues. The main problem faced in automatic off-line design (and also in semi-automatic off-line design) is the so-called reality gap (Jakobi et al. 1995), that is, the differences between reality and simulation models on which the off-line optimization is based. Due to the reality gap, control software designed off-line typically experiences an important performance drop when ported to the real robots. Even worse, the drop is method-dependent with some design methods being more intrinsically robust to the effects of the

reality gap than others. This has implications on how instances of control software should be assessed and eventually selected before being deployed in reality (Ligot and Birattari 2020, 2022).

Automatic off-line design is currently an early-stage technology that has been mostly demonstrated with laboratory experiments (Birattari et al. 2019). Indeed, automatic methods have rarely been used to produce control software for robot swarms in realistic deployments. Important scientific and engineering questions need to be addressed before reaching mature methods that are ready for real-world application. Key questions focus on the effectiveness of automatic methods in producing control software that enables robot swarms to meet the specifications of their mission. Can we design effective and reliable robot swarms via automatic off-line design? What are the components that influence the effectiveness of a method? How can we conceive a method that is effective? Given a class of missions, which is the most appropriate design method? Which features of a mission make it more or less hard to be tackled? To what extent a design method is robust to the reality gap? What can we do to improve the robustness of a method? How can we characterize and specify a mission or a class of missions that a swarm must perform? To what extent an automatic design method can be ported to other design problems, and vice versa?

The automatic offline design of robot swarms can be framed as a multi-agent reinforcement learning problem (Tan 1993; Matarić 1997; Albrecht et al. 2024). In this framework, a group of robots must learn behaviors without explicit examples, relying instead on delayed feedback (i.e., a performance metric) that reflects their collective performance. In reinforcement learning, the goal is to find a policy—a rule that maps the current state of the system to an action. This search is often driven by a value function, which evaluates how beneficial it is for an agent to be in a specific state or take a particular action, ultimately guiding the agent's decision-making process. Alterntively, some reinforcement learning approaches bypass the use of a value function in the learning process and perform a direct search in the policy space.

Typical multi-agent reinforcement learning approaches, those that rely on a value function, show significant promise for designing collective behaviors for groups of robots. However, related research often involves relaxing the desired characteristics and properties of robot swarms—i.e., injecting expert knowledge, such as reward shaping, or providing robots with global information to make the action-state space observable (Kuckling 2023b). Additionally, few demonstrations exist on their applicability with physical robots (Gharbi et al. 2023; Heuthe et al.

2024; Sadeghi Amjadi et al. 2024), and the current literature does not position new methods with respect to the state of the art on the automatic design of robot swarms (Hüttenrauch et al. 2019; Bloom et al. 2023; Sadeghi Amjadi et al. 2024).

Methods that conduct a direct search in the policy space are predominant in the state-of-the-art literature on the automatic design of robot swarms, with contributions that belong mainly in two approaches: (i) neuroevolution (Nolfi and Floreano 2000; Trianni 2008); and (ii) automatic modular design (AutoMoDe) (Francesca et al. 2014b; Birattari et al. 2021). Neuroevolution is the traditional approach to the automatic design of collective behaviors for robot swarms: each robot is controlled by an artificial neural network whose parameters (and possibly the architecture) are obtained via artificial evolution. This approach has proven to be flexible, enabling the design of diverse collective behaviors across various robot platforms. However, it is known to experience significant performance drops when the robots are deployed, compared to the performance observed in simulation during the design phase (Ligot and Birattari 2020; Hasselmann et al. 2021). The AutoMoDe approach was proposed as an alternative to neuroevolution (Francesca et al. 2014b, 2015). In this approach, the control software of the robots is produced via an optimization-based process that fine-tunes pre-existing software modules and combines them into a modular architecture such as a probabilistic finite-state machine or a behavior tree. The software modules can be produced manually or with the assistance of optimization processes—for example, via evolutionary computation (Gomes and Christensen 2018; Hasselmann et al. 2023).

The seminal work of Francesca (2017) on the AutoMoDe approach demonstrated two key points: (i) in a range of missions, AutoMoDe was shown to produce control software for robot swarms that met mission specifications more effectively than those designed by human experts (Francesca et al. 2015); and (ii) AutoMoDe is less prone to the effects of the reality gap than neuroevolution (Francesca et al. 2014b). More precisely, AutoMoDe methods tend to produce control software that eventually performs better once ported from simulation to reality. This occurs because neuroevolution tends to overfit the control software to the simulator's characteristics during the design process. On the contrary, the modularity of AutoMoDe introduces bias into the design process and limits the variance of the resulting control software to what can be achieved with predefined software modules, ultimately reducing the risk of overfitting. This understanding of the bias-variance trade-off aligns with the interpretation commonly discussed in the machine learning literature (Geman et al. 1992).

Inspired by the early AutoMoDe studies, researchers have devoted significant

effort to explore and better understand the underlying capabilities and limitations of the AutoMoDe approach (Birattari et al. 2021). Notably, Kuckling (2023a) studied the impact of adopting various optimization strategies and control architectures within the design process. Ligot (2023) conceived tools and protocols to assess and predict the performance of automatic design methods. Hasselmann (2023) explored novel ways to produce software modules for instances of AutoMoDe. Salman (2024) investigated original ways to exploit stigmergy in AutoMoDe. These investigations, along with a growing body of literature (Salman et al. 2019; Spaey et al. 2020; Pagnozzi and Birattari 2021; Bozhinoski and Birattari 2022; Cambier and Ferrante 2022; Gharbi et al. 2023; Endo et al. 2023; Kegeleirs et al. 2024b), have shown evidence that AutoMoDe is a general framework that can be adapted to study different aspects of the automatic design of robot swarms.

The research we present here also builds on the work of Francesca (2017) and draws inspiration and ideas from the research that followed (Birattari et al. 2019; Garzón Ramos et al. 2021b). Particularly, we focused on investigating whether AutoMoDe methods could leverage environmental and inter-robot signaling to design collective behaviors for robot swarms. Signaling is a key mechanism to enable self-organizing behaviors in robot swarms. Robots can employ various signaling mechanisms to communicate with one another and interpret environmental signals, enabling them to adapt to changing conditions (Trianni et al. 2004; Trianni and Dorigo 2006). Indeed, since the early studies in swarm robotics, it has been observed that designing effective communication strategies positively influence the performance of the robots (Trianni and Dorigo 2006). Yet, the current literature on the automatic design of robot swarms does not provide an approach that effectively leverages signaling capabilities in a generally applicable way.

Little work exists on the fully automatic design of communication-based behaviors for robot swarms (Hasselmann et al. 2023), let alone on developing a general framework applicable to diverse signaling mechanisms and design problems. We aim to bridge this gap in the current state of the art through the use of AutoMoDe. To do so, we diverged from the recurring experimental scenarios addressed by Francesca (2017), Kuckling (2023a), Ligot (2023), Hasselmann (2023), and other AutoMoDe literature. We explored two original avenues. On the one hand, we explored new experimental scenarios in which robots must rely on different forms of signaling to collectively perform their mission (Garzón Ramos and Birattari 2020, 2024; Mendiburu et al. 2022; Salman et al. 2024). On the other hand, we formulated design problems in which robot swarms must perform missions evaluated by concurrent performance measures (Garzón Ramos et al. 2024; Szpirer et al.

2024a). Exploring these two avenues helped define the boundaries of AutoMoDe's applicability, with scientific contributions showing that *automatic modular design methods can leverage environmental and inter-robot signaling to tackle missions where a robot swarm is required to communicate, react to events, and perform missions sequentially*—the thesis developed in this dissertation.

Materials and contributions

The scientific contributions of this thesis are grounded in the fundamental principles that we believe should guide the automatic off-line design of robot swarms (Birattari et al. 2019). First, the design methods presented here aim to address a class of missions within a design problem, rather than focusing on a single specific mission. Indeed, the methods do not require mission-specific modifications or adjustments to operate. Second, once a mission is specified and provided to the design methods, no human intervention is required at any stage of the design process. We applied these principles to both the AutoMoDe methods introduced in the thesis and the baseline methods we used to assess their performance. In our studies, we routinely used neuroevolution as a baseline (Garzón Ramos and Birattari 2020, 2024; Mendiburu et al. 2022; Garzón Ramos et al. 2024; Salman et al. 2024). This contributed to positioning our research with respect to the literature on the optimization-based design of robot swarms. Additionally, in some studies, we included control software manually developed by human designers (Garzón Ramos and Birattari 2024; Salman et al. 2024; Szpirer et al. 2024a). This other baseline, first introduced by Francesca (2017), helps apprise the difficulty involved in addressing a design problem. We used it in our research in a similar way.

In our research, we first investigated whether AutoMoDe methods could establish meaningful signal-based interactions both between robots and between robots and their environment. Specifically, we investigated whether AutoMoDe could leverage signals—detectable changes in the robots' behavior or environment—to design mission-specific coordination strategies for robot swarms. This was based on `TuttiFrutti` (Garzón Ramos and Birattari 2020), an original method specialized in generating control software for robots that can display and perceive color signals using RGB LEDs and a camera, respectively. We used colors as a signaling mechanism to operationalize AutoMoDe's ability to create signal-based interactions between physical robots—without relying on abstract or simulated communication. AutoMoDe would have to produce control software that enables the robots to utilize these colors effectively, resulting in collective behaviors that meet the mission's

specification. Historically, the diversity of missions that AutoMoDe methods could address had been limited by the restricted capabilities of the robot platform considered and the simplicity of the scenarios in which the robots operated—both constraints inherited from the first AutoMoDe methods. In `TuttiFrutti`, we considered a more capable robot platform to perform missions in which robots must act with respect to the colors displayed by objects in their environment or by their peers. To address the issue of simple experimental scenarios, we developed `MoCA` (Garzón Ramos et al. 2022), an open-source modular system to perform experiments with robot swarms. `MoCA` provides the tools to create, simulate, and physically deploy scenarios in which robots can react to the colors displayed by programmable RGB modules that surround their workspace. By introducing `TuttiFrutti` and `MoCA`, we significantly expanded the diversity of design problems and swarm robotics missions that can be studied with the AutoMoDe approach.

The key challenge for `TuttiFrutti` was that color signals are inherently meaningless or empty at the outset. During the design process, `TuttiFrutti` had to assign meaning to these signals, enabling their use in specific contexts to trigger particular behaviors. With `TuttiFrutti`, we showed how, and how well, AutoMoDe can automatically generate control software to perform missions without relying on mission-specific predefined communication protocols—as is typically done in the swarm robotics literature. `TuttiFrutti` provides the swarm robotics community with a generally applicable method for designing collective behaviors in which robots use color signals to coordinate—demonstrated through experiments with physical robots that must navigate their environment, react to events, and communicate relevant information to their peers. This is the first major scientific contribution of the thesis.

In the second part of our research, we used the capabilities introduced with `TuttiFrutti` and `MoCA` to explore whether AutoMoDe methods could address design problems that require handling multiple design criteria during the optimization process. This is a complex design problem that had rarely been addressed in the literature on the automatic design of robot swarms. Traditionally, research in the field had focused on missions specified by a single design criterion, using design methods based on single-objective optimization algorithms. This approach was initially adopted by Francesca (2017) and was inherited in the subsequent AutoMoDe literature. We relaxed this assumption in our research. We conceived `Mandarina` (Garzón Ramos et al. 2024), an original method that builds on `TuttiFrutti` and includes modifications to address design problems with multiple design criteria. We used `Mandarina` to generate control software, through a single

design process, for robot swarms to perform missions in sequence. In this problem, each mission in the sequence represents an independent design criterion that the automatic method had to handle during the optimization process. `Mandarina` designed robot swarms capable of performing missions sequentially by using environmental signals to determine which mission to perform and when to switch missions.

The key challenge for `Mandarina` was to produce control software capable of concurrently meeting the specifications of the missions in a given sequence—i.e., concurrently maximizing their associated performance metrics. In this process, `Mandarina` had to handle sequences of missions with conflicting goals between missions and performance measures defined on varying scales. Multi-criteria design problems in swarm robotics have been mostly addressed by manually applying expert knowledge to combine the design criteria into a single performance measure. With `Mandarina`, we showed how AutoMoDe can be adapted to manage such diverse design criteria during the design process without the need for manual intervention. `Mandarina` provides the swarm robotics community with an automatic method to produce control software for robot swarms in design problems that specify multiple performance metrics to be optimized. This is the second major scientific contribution of the thesis.

The results achieved with `TuttiFrutti` and `Mandarina` motivated us to apply their ideas to new problems in the automatic design of robot swarms. Therefore, we dedicated the final part of our research to explore their application in the automatic design of (i) spatially-organizing behaviors, (ii) stigmergy-based behaviors, (iii) shepherding behaviors, and (iv) the design of collective behaviors by demonstration. We based these studies on four original AutoMoDe methods: `Mate`, `Habanero`, `Pistacchio`, and DemoTuttiFrutti-`MO` (DTF-`MO`). First, experiments with `TuttiFrutti` showed that robots could use color signals to adjust their relative distance. We applied this idea in `Mate` to investigate whether a similar signaling approach could help meet predefined spatial distribution constraints for the swarm (Mendiburu et al. 2022). Second, experiments with `TuttiFrutti` demonstrated that robots could establish simple communication-based behaviors using direct signaling. We applied this idea in `Habanero` to explore whether robots could establish stigmergy-based communication using indirect signaling based on artificial pheromones (Salman et al. 2024). Third, experiments with `TuttiFrutti` showed that robots could effectively identify their peers and coordinate with them using color signals. We applied this idea in `Pistacchio` to investigate whether robots could use color signals to identify and coordinate with their peers and with

robots from another swarm in robot shepherding (Garzón Ramos and Birattari 2024). Fourth, the experiments with `Mandarina` demonstrated that it is possible to specify and address scenarios requiring the sequential execution of missions. We applied this idea in DTF-`MO` to explore whether the desired collective behaviors for a sequence of missions could be specified using demonstrations (Szpirer et al. 2024a).

The studies conducted with `Mate`, `Habanero`, `Pistacchio`, and DTF-`MO` addressed open problems in the automatic design of robot swarms. `Mate` tackled limitations of the seminal AutoMoDe methods by enabling the design of robot swarms that distribute themselves in an ordered manner within a space. `Habanero` showed that stigmergy-based behaviors could be automatically designed using AutoMoDe, providing an alternative to the manual approaches typically seen in the swarm robotics literature. `Pistacchio` illustrated how AutoMoDe is capable of designing coordination strategies for heterogeneous groups of robots, even when these robots operate with different control software architectures. DTF-`MO` showed that AutoMoDe is capable of conducting a multi-criteria design process with greater automation, removing the requirement for a mathematically formulated objective function. Most importantly for the purpose of this thesis, the experiments with `Mate`, `Habanero`, `Pistacchio` and DTF-`MO` demonstrated the versatility and wide applicability of the AutoMoDe approach, along with the ideas introduced in `Tutti-Frutti` and `Mandarina`. To the best of our knowledge, we provide here the most diverse compendium of collective behaviors achieved through the automatic design of robot swarms. This is the third major scientific contribution of the thesis.

In the course of our work, we collected and analyzed substantial empirical evidence that supports our thesis. The experiments and analyses were conducted within a consistent empirical framework, adapted from the original work of Francesca (2017). In each study, we proposed a set of swarm robotics missions that the automatic methods under consideration should address. These missions were based on well-defined collective behaviors for robot swarms, including various forms of aggregation, coverage, and decision-making (Brambilla et al. 2013; Schranz et al. 2020). All experiments were conducted using a swarm of e-puck robots (Mondada et al. 2009)—a small wheeled platform widely used in swarm robotics research (Dorigo et al. 2021). We employed both simulations and physical robots. In each study, we applied the design methods under consideration multiple times to produce sets of control software instances. Then, these sets were subjected to statistical analysis to determine the relative performance between methods. We consistently used this empirical framework across our studies to ensure the comparability of results,

minimize potential variability from different approaches, and maintain a clear and logical structure throughout the thesis.

The scientific output presented in this dissertation, along with the tools and materials developed, has been made available to the scientific community. These contributions, both scientific and technical, offer means to assist in the design, deployment, and assessment of robot swarms. Throughout the development of this thesis, we actively communicated our research to both the scientific community and the general public. In doing so, we notably expanded the body of literature on automatic modular design for robot swarms and helped to popularize swarm robotics research through outreach activities.

Following this introductory chapter, the rest of the dissertation is organized as follows. Chapter 2 provides a review of the literature in the automatic modular design of robot swarms, including references to relevant swarm robotics literature and to the design problems explored in the thesis. Chapter 3 describes `TuttiFrutti` and our research on designing collective behaviors for robots that can display and perceive color signals. Chapter 4 presents `Mandarina` and our investigation into the design of robot swarms under concurrent design criteria, including the application of these concepts to creating robot swarms capable of performing sequences of missions. Chapter 5 provides details on `Mate`, `Habanero`, `Pistacchio`, and DTF-`MO`, providing an overview of each method, the experiments conducted, and the most significant findings of each study. We also briefly discuss our outreach efforts. Finally, Chapter 6 concludes the dissertation and outlines potential directions for future research.

2. Literature review

Automatic modular design (AutoMoDe) is a general approach to designing collective behaviors for robot swarms that must address a specific class of problems. Four principal elements characterize the design methods that are conceived within this approach: the class of problems on which they are intended to operate; the robot platform for which control software should be designed; the optimization strategy that drives the design process; and the control architecture and modules that are used to produce the control software of the robots.

In this chapter, we review and discuss literature on AutoMoDe in light of these four elements. To provide context, we briefly introduce the generalities of swarm robotics and highlight other approaches to the automatic off-line design of robot swarms. We conclude our literature review by organizing and revising research relevant to the design problems addressed in this dissertation.

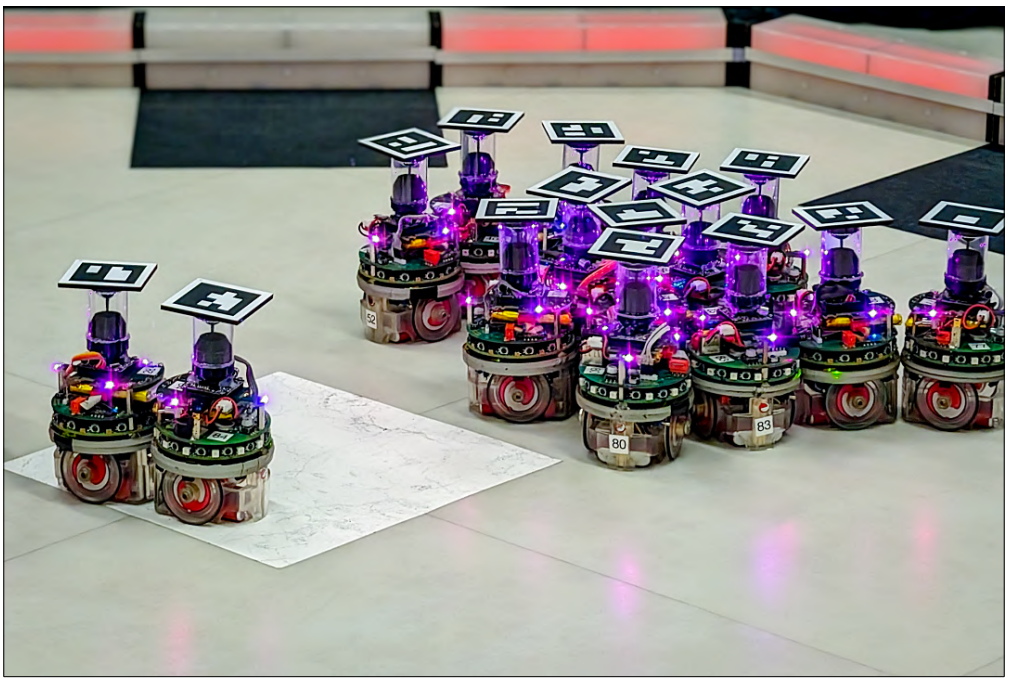


Figure 2.1: Swarm of e-puck robots. All studies presented in this dissertation are based on the e-puck.

2.1 Swarm robotics

A robot swarm is a redundant and self-organized group of robots capable of coordination and cooperation (Şahin 2005). Individually, the robots of a swarm are usually simple and have limited capabilities. However, through the collective actions of the group, the swarm can overcome the limitations of individual robots and perform missions that a single robot could not perform alone. Robot swarms are the embodiment of the ideas and concepts of swarm intelligence (Dorigo and Birattari 2007). This characterization has remained consistent since the early work that formally introduced the systems. For example, Beni (2005) described robot swarms as groups of non-intelligent robots that, when combined, function as a single intelligent entity. Indeed, the robots of a swarm are mostly considered to operate with reactive control and limited information-processing—missing the inference or planning abilities typical of other robotic systems. However, despite these limitations, they can achieve complex collective behaviors by relying solely on local interactions between robots and between robots and the environment. Figure 2.1 shows a swarm of e-puck robots (Mondada et al. 2009; Garattoni et al. 2015)—the robot platform used during the development of this thesis.

The emergence of collective behavior from local interactions can be described

and analyzed from two perspectives: the microscopic level and the macroscopic one (Garattoni and Birattari 2016; Hamann 2018). At the microscopic level, the focus is on understanding how the individual actions of the robots contribute to the collective behavior of the swarm. Microscopic analyses study the behavior rules on which individual robots operate. The discussion centers on how a robot perceives its environment and how it responds to the local information it has available. In these analyses, the behavior of a robot is usually described by microscopic models that describe its control logic. Common approaches consider monolithic principled methods, such as virtual-physics models, or structured control architectures like finite-state machines. For example, virtual-physics models have been used to describe spatially-organizing behaviors such as pattern formation (Spears et al. 2004), chain formation (Maxim et al. 2009), and collective exploration (Howard et al. 2002). Probabilistic finite-state machines have been used to describe collective behaviors such as aggregation (Soysal and Şahin 2005), chain formation (Nouyan et al. 2008), and also division of labor and task allocation (Labella et al. 2006; Liu et al. 2007). The microscopic perspective enables the application of bottom-up approaches to the design of robot swarms.

In contrast, the focus at the macroscopic level is on understanding the behavior of the swarm as a whole and describing how it evolves over time. These analyses aim to characterize the overall functioning of the collective behavior of the robots and its related properties. The traditional approach to conducting macroscopic studies is to apply mathematical analysis and modeling. Hamann (2012), for example, promoted the development of generally applicable swarm models and formalisms to provide insight into the behavior and properties of robot swarms. Differential equations are commonly used for this purpose in the literature. Examples include models for object clustering (Martinoli et al. 1999), foraging (Lerman and Galstyan 2002), flocking (Winfield et al. 2008), and stick pulling (Lerman et al. 2001). Recently, data-driven approaches have gained notable attention as an alternative for characterizing and analyzing collective behaviors. Data-driven approaches typically rely on the application of performance measures that specify the collective behavior under study. For instance, Milner et al. (2023) introduced the concept of swarm performance indicators—measurements derived from empirical data that describe the degree of robustness, scalability, and flexibility of a robot swarm. Similarly, Ligot and Birattari (2022) proposed a data-driven protocol to predict the real-world performance of robot swarms by analyzing their statistical performance in simulation. Notably, Kuckling et al. (2024) combined traditional mathematical modeling with data-driven analysis to shed light on misunderstood aspects of the

scalability of robot swarms.

The increasing interest in the design and realization of robot swarms gave rise to swarm robotics: a research field devoted to the study, development, validation and application of groups of robots that coordinate via swarm intelligence (Dorigo et al. 2014). Originally, swarm robotics emerged from the need to empirically validate theoretical models of social animal behavior (Dorigo et al. 2021)—primarily driven by research in biology. However, in recent years, the field has gradually shifted its focus. Although there is still interest in applying biological principles to the design of robot swarms, an increasing amount of research is now focused on developing engineering approaches that stand on their own. There is a growing body of literature that emphasizes the need for systematic design methods that guarantee system properties and performance levels in the operation of robot swarms (Brambilla et al. 2013; Hunt and Hauert 2020).

For a comprehensive introduction to swarm robotics, we recommend the textbook by Hamann (2018). The evolution of the field is thoroughly discussed by Dorigo et al. (2020, 2021). For an in-depth review of the field, we suggest the literature compiled and organized by Brambilla et al. (2013), Garattoni and Birattari (2016), Nedjah and Silva Junior (2019), and Schranz et al. (2020).

2.1.1 Characteristics of a robot swarm

Robot swarms share the typical characteristics of swarm intelligence systems (Dorigo et al. 2014). As discussed before, the behavior of a robot swarm emerges from local interactions between individuals and between the individuals and their environment. Despite being limited to local perception and communication, the robots can coordinate to perform missions that are relatively complex given their individual capabilities. Robot swarms operate in large groups without the need for centralized control or external infrastructure to guide their actions, or to assist with their localization and communication. A swarm is capable of self-organization and parallelization. If required by the mission, it can autonomously define roles and distribute tasks among its members. For example, Ferrante et al. (2015) investigated how roles emerge in a robot swarm during a complex foraging mission. In the experiments, the task was autonomously divided into simpler sub-tasks and distributed among the robots. Similarly, Garattoni and Birattari (2018) demonstrated that a group of robots can adopt specific roles to collectively sequence and plan the execution of sub-tasks.

Robot swarms are often characterized as redundant homogeneous systems. In

the literature, the most common swarm configuration is a group of robots that share the same physical design, are built with identical hardware, and run the same control software. This homogeneity, originally adopted from other disciplines in swarm intelligence, has facilitated the study of self-organization with groups of robots in simple scenarios. However, it is at the same time a restrictive working hypothesis. Recent discussions have emphasized the need to inject a degree of heterogeneity into the system to tackle more complex missions (Dorigo et al. 2021). At a basic level, this has been achieved by considering quasi-homogeneous robot swarms. These configurations rely on robots that have similar, but not identical, physical capabilities and/or operate with different control software. This approach was adopted by Strobel et al. (2018) to study collective decision-making in swarms that comprise two types of robots: those susceptible to change their opinion and those that exhibit stubborn behavior, remaining fixed on a single opinion. In a different context, Jones et al. (2019) experimented with the automatic design of collective behaviors by allowing each robot to independently develop its own control software.

Studies on fully operational heterogeneous robot swarms are rare and are mainly conducted in simulation. For example, Aswale and Pinciroli (2023) conducted simulations of multi-skilled groups of robots that must coordinate to perform a series of tasks. In their experiments, robots dynamically form coalitions to complete tasks that require the combined skills of different robots in the group. The most notable example of a heterogeneous robot swarm using physical robots was achieved by Dorigo et al. (2013) in the Swarmanoid project. In this project, the researchers created a swarm of aerial, wheeled, and grasping robots that could self-organize and cooperate to perform an object retrieval mission.

Distinction from other multi-robot systems

Swarm robotics is a specific approach to the coordination of multi-robot systems that can be distinguished from other more general approaches. In overall, other multi-robot systems typically incorporate advanced capabilities that are excluded from robot swarms, such as: (i) global localization and information about their environment; (ii) complex planning and interaction rules between robots; (iii) sophisticated communication protocols with guaranteed connectivity; (iv) explicit assignment of roles and identities; (v) precise knowledge of the number of operating robots; (vi) an explicit definition of the mission to be performed; and (vii) centralized control or coordination. It is worth noting that this is not a strict

boundary between swarm robotics and other forms of multi-robot systems. In fact, many systems discussed in this review of the literature may incorporate some of these capabilities. The distinction between robot swarms and other multi-robot systems is meant to provide a useful framework for organizing the literature. It allows for fair comparisons between approaches, sets appropriate expectations for the collective capabilities of the systems, and helps to give context to the design challenges addressed. For a broader overview of swarm robotics in the context of other multi-robot systems, we refer the reader to Parker et al. (2016).

2.1.2 Desirable properties of a robot swarm

The main characteristics of a robot swarm—self-organization, redundancy, and locality—enable the group of robots to operate with varying degrees of robustness, scalability, and flexibility (Dorigo et al. 2014). These are desirable system properties that have traditionally attracted attention to the realization of robot swarms.

The *robustness* of a robot swarm refers to the group's ability to tolerate individual robot failures. As discussed before, a robot swarm is a redundant system composed of a large number of individuals. If a few robots in the swarm fail, the overall operation is not significantly impacted as other robots continue to operate. Furthermore, because the swarm operates autonomously without a leader or external control infrastructure, the system has no single point of failure. In this topic, Christensen et al. (2009) conducted studies on the autonomous detection and repair of failing robots—thus improving the overall resilience of the system. In their experiments, the failure of a robot could be collectively detected by its peers, which could then take action to make it operative again. More recently, Lee et al. (2022) investigated methods to quantify the severity of individual robot failures and their impact on the overall performance of the swarm. The study aimed to support autonomous decision-making processes that could identify when interventions are needed.

The *scalability* of a robot swarm refers to the possibility of adding or removing robots to the group without having to redefine their behavior rules. More precisely, it refers to the ability of the swarm to remain unaffected by changes in the number of robots. The scalability of a swarm is closely related to the locality of the information with which robots operate. Each robot interacts only with neighboring peers, which makes it unaffected by the actions (or inaction) of robots outside its perception range. As a result, a robot is not severely affected by the appearance or disappearance of robots in portions of the swarm that are not directly perceivable.

A remarkable demonstration of the scalability of robot swarms was presented by Rubenstein et al. (2014) with the successful deployment of one thousand coin-sized legged robots named Kilobots. It is important to note that the environment in which the robots operate affects the scalability of the swarm. A significant change in the number of robots can drastically affect the density of the swarm, which in turn can greatly impact its operation. The scalability is therefore tied to the density of interactions between robots (Hamann 2018). Hamann and Reina (2021) studied this phenomenon in depth and proposed a general model to predict the potential for scalability in robot swarms and other parallelized systems.

The *flexibility* of a robot swarm refers to the group's ability to adapt to a wide range of tasks and/or potential changes in its environment. As already mentioned, the robots of a swarm are typically homogeneous, unspecialized, and deployed without predefined roles. Through self-organization, they can adapt to the specific requirements of the mission at hand. This flexibility allows the swarm to be easily reconfigured, display a variety of collective behaviors, and coordinate in different ways to perform its mission. The adaptability of robot swarms is well demonstrated by the evolution of the field itself. Much of the existing swarm robotics literature reports results obtained with generic simple robots endowed with similar functional capabilities (Dorigo et al. 2021)—for example, the foot-bot (Dorigo et al. 2013) and the e-puck (Mondada et al. 2009). Relying on similar functional capabilities, swarms of foot-bots and e-pucks have been shown to be capable of addressing problems that require abilities as diverse as the emergence of shape (Mathews et al. 2017) and planning (Garattoni and Birattari 2018). Moreover, foot-bots, e-pucks, and similar robots are the common base of numerous studies on typical problems like aggregation, foraging, and collective decision-making—as detailed in the reviews by Brambilla et al. (2013), Nedjah and Silva Junior (2019), and Schranz et al. (2020).

In the literature, robot swarms are often described as inherently robust, scalable, and flexible systems. However, it has been noted that these properties cannot be taken for granted and may require careful system design to achieve them to a certain degree (Dorigo et al. 2021; Milner et al. 2023).

2.2 Automatic off-line design of robot swarms

Researchers commonly design robot swarms via an iterative manual process. In this approach, a designer manually produces and refines the control software for

individual robots until the desired collective behavior emerges. A few principled methods have been proposed to aid in the manual design process for specific problems and classes of missions (Spears et al. 2004; Kazadi 2009; Schmickl and Hamann 2011; Berman et al. 2011; Beal et al. 2012; Lopes et al. 2014; Brambilla et al. 2014; Reina et al. 2015b,a; Lopes et al. 2016; Pinciroli and Beltrame 2016). However, the underlying assumptions of these methods prevent them from offering a single universally applicable solution (Birattari et al. 2020). That is, they cannot offer on their own a single approach to design all types of collective behaviors. When these principled methods fall short or their underlying hypotheses are too restrictive for the problem at hand, manual trial-and-error remains the dominant approach for producing control software for robot swarms.

The manual approach has served to demonstrate the feasibility of a wide variety of collective behaviors (Brambilla et al. 2013; Schranz et al. 2020). However, it is labor-intensive and challenging to accurately evaluate or characterize its performance. Manual design heavily depends on the designer's expertise, which is difficult to transfer and makes the process challenging to reproduce by designers with different skill sets. We argue that for swarm robotics to scale current robotics solutions into large-scale services, the common approach to producing control software for robot swarms must undergo a significant change. It must shift from a labor-intensive, ad hoc approach to a systematic engineering practice capable of producing robot swarms that are ready to operate out of the box. In this context, adopting automated practices can offer swarm designers a reproducible design process, enabling the production of control software with clearly defined performance guarantees. This is particularly relevant for application scenarios where the swarm must be repeatedly deployed and adapted to ever-changing environments (Birattari et al. 2019)—where lengthy manual development is not feasible.

As an alternative to manual design, optimization-based methods can reduce the need for human intervention in the design process. In this approach, the design problem is restated as an optimization problem: an optimization algorithm explores a space of possible instances of control software for the individual robots, and selects the one that maximizes the collective performance of the swarm—according to a mission-specific performance measure. This approach can potentially allow the design problem to be addressed in a systematic and fully automatic way (Birattari et al. 2020). Common classifications of optimization-based design methods divide them into on-line and off-line methods, and into semi-automatic and (fully) automatic methods (Birattari et al. 2020). This was discussed in further

detail in Chapter 1. The methods presented in this dissertation belong to the the automatic off-line design of robot swarms (Birattari et al. 2019). We therefore limit our discussion to this approach.

Automatic off-line design methods produce the control software of the robots before the swarm is deployed. Typically, the design process is conducted first in simulation, and the control software obtained is then ported to the robots. The largest body of literature on the automatic offline design of robot swarms belongs to neuroevolution (Nolfi and Floreano 2000; Trianni 2008; Doncieux et al. 2015)—both in the semi-automatic and fully automatic cases. In recent years, AutoMoDe (Francesca et al. 2014b; Birattari et al. 2021) has also received notable attention. In the remainder of the section, we provide a brief overview of notable studies in the automatic off-line design of robot swarms. We focus here on optimization-based methods other than those belonging to the AutoMoDe family, which are discussed in detail in Section 2.3.

For a more general overview of the approaches to the optimization-based design of robot swarms, we recommend the literature reviewed and organized by Francesca and Birattari (2016), Bredeche et al. (2018), Birattari et al. (2019, 2020), and Kuckling (2023b).

2.2.1 Notable research

We highlight design problems, robot platforms, optimization strategies, and control architectures considered in notable research on the automatic off-line design of robot swarms.

Quinn et al. (2003) used neuroevolution to generate control software for three medium-sized wheeled robots. The robots used infrared transceivers to detect each other by continuously emitting signals. The same sensors were also used to detect the proximity of obstacles. Each robot sent its sensory data to a central host computer, which ran distributed instances of control software for the swarm. The neural network mapped sensory inputs to discrete movements of the wheels (forward, still, backward). The evolutionary process optimized both the neural network's architecture and its weights. This method was demonstrated using physical robots on missions that required coordinated motion. Notably, they showed that small adjustments to a performance measure could significantly alter the evolved behavior, even giving the robots obstacle-avoidance capabilities.

Dorigo et al. (2003) used neuroevolution to generate control software for swarms of four and eight ground robots called *s-bots*. These robots could attach to one

another to form a larger robot, known as a *swarm-bot*. The authors explored scenarios where the robots could perceive each other through sound signals when separated, or via traction sensors when physically connected. The neural network mapped sensory inputs to the velocity of their differential treels (tracks and wheels). The study showed results on the evolution of control software for aggregation and coordinated motion. In aggregation, the robots relied on sound signals to approach each other. In coordinated motion, a group of pre-assembled s-bots used traction sensing to align their bodies and move as a unified group.

Pugh et al. (2005) used particle swarm optimization (PSO) and artificial neural networks to generate control software for two Khepera robots (Mondada et al. 1997). The robots could only detect each other through physical proximity. The neural network that mapped proximity sensor readings to wheel velocities. The authors compared the performance of PSO and a genetic algorithm to optimize the weights of the neural network. The experiments were conducted in simulation on a mission that required the robots to move quickly while avoiding obstacles.

Christensen and Dorigo (2006) used neuroevolution to generate control software for a swarm of three s-bots. The s-bots in this study were equipped with ground sensors, rotation sensors, light intensity sensors, traction sensors, and a sound sensor. The robots could detect each other by emitting sound signals and using traction sensors when physically connected. The authors investigated three architectures of multi-layered feed-forward neural networks to map sensory inputs to the robots' treels. They compared the effectiveness of three evolutionary processes in optimizing the weights of the neural network. Initial experiments were conducted in simulation, followed by physical demonstrations in a mission involving phototaxis with hole-avoidance. In this mission, black patches on the floor represented the holes that the robots should avoid. The robots were pre-programmed to emit a sound when detecting a hole, alerting the entire group. Notably, the authors detailed how they incorporated various design criteria into a single objective function to guide the evolutionary process toward the desired behavior.

Trianni and Dorigo (2006) used neuroevolution to generate control software for a swarm of four s-bots. The s-bots in this study were equipped with ground sensors, traction sensors, and sound sensors. The neural network mapped sensory inputs to the treels and speaker of the robot. The experiments were conducted in simulation on a hole-avoidance mission. The authors compared the evolution of control software under three different scenarios of inter-robot perception. In the first scenario, collective motion was evolved while restricting the robots to perceive each other only through the force detected by their traction sensors. In the second

scenario, collective movement was again evolved, but this time the robots could also emit a pre-programmed sound signal whenever one detected a hole. In the third scenario, both the movement and the protocol for triggering the sound signal were evolved simultaneously.

Trianni and Nolfi (2009) used neuroevolution to generate control software for swarms of s-bots. In this study, the s-bots were only equipped with ground sensors and sound sensors. The robots could detect each other through sound signals, which were audible to all robots in the arena, but did not allow robot identification. The neural network mapped sensory inputs to both the wheels and the speaker. The authors evolved the movement of the robots and the protocol for triggering the sound signal. The experiments were conducted in simulation on a synchronization mission with up to ninety-six s-bots, and the control software was later tested on groups of two and three physical robots. In this mission, the robots synchronized by exhibiting periodic movement patterns, partially coordinated through sound signaling.

Hauert et al. (2009) used neuroevolution to generate control software for a swarm of twenty flying drones. The drones were equipped with sensors to measure their global heading relative to Earth's magnetic field. They could also communicate with other robots by broadcasting predefined data within a local range. The neural network mapped the heading of the robot and connectivity ratio (i.e., number of peers in reach) to the turning rate of the robot while in flight. The robot's flight speed was not controlled by the neural network and remained constant at all times. The experiments were conducted in simulation on a networking mission in which the swarm must establish a connected network between two points. The drones used communication to estimate their connectivity within the network and, combined with their heading information, could maneuver effectively to maintain a constant connection.

Ampatzis et al. (2009) used neuroevolution to generate control software for two s-bots. Unlike previous studies, these s-bots were equipped with sensors on their grippers and cameras capable of detecting color signals. The robots could detect each other by continuously displaying a pre-programmed color signal, which was perceived through their cameras. The neural network mapped the color signals detected by the camera, as well as the state of the gripper (whether it was grasping an object or not), to the movement of the treels and the actuation of the gripper. The authors evolved control software for a self-assembly mission, where robots were rewarded for approaching each other, physically connecting, and avoiding collisions. They also compared the effects of using red, green, and blue signals in

the communication between s-bots.

Waibel et al. (2009) used neuroevolution to generate control software for a swarm of ten small wheeled robots called micro-robots. These robots could detect each other only through physical proximity using proximity sensors, but they could perceive their environment and other objects using two cameras. The neural network mapped readings from the proximity sensors and cameras to the movement of the robots' wheels. The authors compared four strategies for evolving the weights of the neural network: (i) homogeneous and heterogeneous team composition; (ii) individual and group performance assessment. The experiments were conducted with physical robots in three foraging scenarios in which the robots must collect objects of different sizes. The evolutionary process produced collective behaviors in which robots either acted individually to transport smaller objects or cooperated to move larger ones.

Hettiarachchi and Spears (2009) used artificial evolution to optimize the control software parameters for swarms of up to one hundred abstract robots. The authors assumed that the robots could perceive and interact with each other based on Newtonian and Lennard-Jones force laws, though no specific details about the required equipment for the robots were provided. The robots operated with control software produced on the basis of these two force models. The authors used artificial evolution to determine the optimal parameter values to perform a mission involving collective motion with obstacle avoidance. They compared the performance of the two force models in this context. The goal of the evolutionary process was to design behaviors that allowed the robots to maintain a uniform spatial organization while performing the mission.

Hecker et al. (2012) used artificial evolution to optimize the control software parameters for a swarm of three small wheeled robots. The robots detected objects in their environment using proximity sensors and interacted with each other via virtual pheromones stored in RFID devices. These pheromones were tracked and communicated to the robots through a central host computer. The control software of the robots was produced on the basis of an ant behavior model, with its parameters optimized using a genetic algorithm. The authors conducted experiments in which three robots foraged virtual objects stored in RFID tags that were scattered in their environment. Additional experiments were performed in simulation with up to one hundred robots.

Gomes et al. (2013) used neuroevolution to generate control software for swarms of five e-puck robots (Mondada et al. 2009). The robots were equipped with proximity sensors to detect objects and specialized sensors to estimate the relative

distance of other robots within a local perception range. Another sensor estimated the percentage of robots perceived relative to the swarm size. The neural network mapped sensor readings to the velocity of the wheels, with a dedicated output node to stop movement immediately. The authors combined novelty search (Lehman and Stanley 2011) with the evolutionary algorithm NEAT (Stanley and Miikkulainen 2002) in their approach. NEAT allowed the evolutionary process to optimize both the neural network's architecture and its parameters. The inclusion of novelty allowed for the creation of a population of control software with diverse properties, without the need to explicitly define an objective function to obtaining them. The authors used simulations to evaluate the performance of the generated control software in missions involving aggregation and coordinated resource sharing.

Gauci et al. (2014b) used a grid search to explore the space of possible control software configurations for a swarm of forty e-pucks. The robots were equipped with a directional camera that acted as a line-of-sight binary sensor to detect the presence of other robots. The robots operated without a structured control architecture. Instead, the binary sensor readings were directly mapped to wheel velocities using a lookup table. The grid search was used to characterize the control software generated by different mappings, based on its ability to perform an aggregation mission. The experiments were conducted with both physical robots and simulations, where the swarm size was scaled up to one thousand robots.

In a parallel study, Gauci et al. (2014a) integrated artificial evolution into their design method. The robots were also equipped with line-of-sight sensors that were emulated through their directional cameras. However, in this study, the robots could detect and distinguish between other robots and movable cylindrical objects. The robots operated with a lookup table that mapped the line-of-sight sensor readings to the velocity of their wheels. The evolutionary process searched for mappings that would enable the robots to cluster the cylindrical objects that were scattered in the environment. The experiments on this clustering mission were conducted in simulation with up to one thousand e-pucks, and its feasibility was demonstrated using five physical robots. In subsequent studies, this design process was applied to (i) shepherding missions (Özdemir et al. 2017) in which the robots clustered other robots, and to (ii) a combination of shepherding and clustering (Dosieah et al. 2022), which involved both robots and static objects.

Duarte et al. (2014) used neuroevolution to generate control software for three robots similar to the e-puck. The authors conducted experiments in which robots must push a series of buttons to open gates in a room and then transport objects within it. To perform these missions, the robots were equipped with proximity

and sound sensors, as well as specialized sensors to detect other robots, objects, gates, and corridors. The authors approached the complex mission by manually decomposing it into simpler sub-missions, then designing control software instances for each sub-mission, and finally assembling them into a hierarchical architecture. The hierarchical control architecture with which robots operated consisted of sub-controllers and arbitrators, with the sub-controllers evolved using neural networks. For each sub-controller, the neural network mapped the sensory inputs of the robot to its wheels. Some sub-missions proved too complex for neuroevolution alone, and the authors manually produced control software for those cases. The study was conducted in simulation, comparing the hierarchical approach to two non-hierarchical methods: a simple evolutionary algorithm and NEAT. Notably, the study demonstrated the feasibility of combining a modular approach with neuroevolution.

Ferrante et al. (2015) used artificial evolution to generate control software for a swarm of five foot-bot robots (Dorigo et al. 2013). The robots were equipped with proximity and ground sensors, along with a range-and-bearing system to detect nearby robots. This version of the foot-bot also featured a gripper for holding and transporting objects. The robots operated with two types of control software. The first was a monolithic approach, in which each robot was assigned a predefined role from a set of possible roles. The second was a modular architecture that combined predefined software modules that described generic low-level behaviors. The authors used these two types of control software in simulation experiments to explore how task partitioning can emerge in a foraging mission. In a first experiment, they used artificial evolution to determine the optimal allocation of roles in the swarm to maximize its performance. In a second experiment, they used grammatical evolution (O'Neill and Ryan 2003; Ferrante et al. 2013) to find the optimal combination of low-level behaviors to perform the mission. The results showed that the modular combination of low-level behaviors could effectively exploit environmental features, producing behaviors similar to those manually crafted on the predefined roles.

Trianni and López-Ibáñez (2015) used neuroevolution to generate control software for swarms of up to ten foot-bots. These foot-bots were equipped with proximity sensors, RGB LEDs, and a camera, allowing them to perceive each other by emitting and detecting color signals. The neural network mapped the color blobs detected by the camera and proximity readings to the velocity of the wheels and a single LED beacon. The researchers evolved neural networks to perform flocking and a variation of the stick-pulling mission (Ijspeert et al. 2001). The

neural network architecture and sensor configuration were independently selected for each mission to facilitate the evolution. Notably, this study explored the use of a multi-objective optimization approach to generate the control software for the robots. This work is further discussed in Section 2.4.

Duarte et al. (2016) used neuroevolution to generate control software for a swarm of ten aquatic surface robots. These robots were equipped with virtual sensors based on GPS, which provided information about the relative positions of other robots, points of interest, and their own heading and location The robots could perceive each other and share positional information via WiFi. The neural network mapped positional data from other robots and points of interest to the two motors controlling the movement of the robot. The researchers used NEAT to evolve control software for four missions: homing, aggregation, dispersion, and coverage. They then demonstrated that these four behaviors could be manually assembled into a modular architecture to perform a mission with sequential tasks. Notably, these experiments were conducted with physical robots operating in an outdoor environment, going beyond typical laboratory settings for swarm robotics research.

Jones et al. (2018) used automatic modular design to generate control software for a swarm of twenty-five Kilobot robots (Rubenstein et al. 2014). The robots were equipped with infrared transceivers for local communication and photoresistors to detect light signals projected into their environment. To detect each other, the robots used infrared messages, which also helped estimate local robot density and the distance to points of interest. The robots operated using behavior trees (Champandard 2007; Colledanchise and Ögren 2018). The authors used genetic programming (Koza 1992) to design the tree architecture by assembling predefined action and composition nodes—i.e., software modules. The action nodes defined both the transition conditions based on sensor readings and the movement behaviors of the robots. The composition nodes coordinated the execution of the action nodes within the behavior tree. Notably, this study showed how automatic modular design could generate effective control software even with the limited capabilities of the Kilobot. In a subsequent study, Jones et al. (2019) showed that behavior trees could also evolve online, with a distributed design process implemented in a swarm of e-pucks with enhanced computing capabilities. This was demonstrated in a mission where the robots were required to collectively push an object.

Kaiser and Hamann (2019) used neuroevolution to generate control software for a swarm of one hundred robots represented as discrete agents. The robots operated

in a grid-like environment and were equipped with binary sensors to detect other robots within their local perception range. The control software consisted of two neural networks that evolved together during the neuroevolution process. The first network controlled the robot's actions through a sensory-motor mapping, while the second network predicted the sensor readings based on the actions selected by the first network. The evolutionary process optimized the network weights to allow the robots to move while minimizing errors in the predicted sensor readings. This approach, known as *minimizing surprise* (Hamann 2014), enables the emergence of collective behaviors without the need for a mission-specific objective function. The authors conducted simulation experiments in which the robots demonstrated shape-formation behaviors. In a follow-up study, Kaiser and Hamann (2022) applied this method to a swarm of four Thymio II robots (Riedo et al. 2013), adapting the minimizing surprise concept to an online design approach.

Mason and Hauert (2023) used neuroevolution to generate control software for a swarm of ten simulated DOTs robots (Jones et al. 2022). The DOTs robots were only equipped with four proximity sensors, limiting their interactions to physical proximity. The neural network mapped proximity readings to rotation commands and a forward velocity. The experiments involved a mission where the robots had to alternate between two tasks, each defined by an independent design criterion. One task required the robots to maximize their movement speed, while the other required them to minimize their distance from the center of the environment. To facilitate task switching, the neural network included two input nodes that indicated to the robot which design criterion was prioritized at any given time. The authors used the evolutionary algorithm xNES (Glasmachers et al. 2010) to optimize the weights of the neural network. The experiments, conducted in simulation, showed that the robots could switch between tasks when external stimuli were applied to the dedicated input nodes.

Endo et al. (2023) used automatic modular design to generate control software for four Khepera IV robots. The robots were equipped with proximity sensors to detect other robots and obstacles, and a camera to identify points of interest. The method operated on parametric software modules that defined the actions the robots could take and the conditions to transition between those actions. The authors used particle swarm optimization to find the suitable combinations of modules to perform an exploration mission. In the experiments, robots were required to collectively locate, identify, and report points of interest in the environment. They also accounted for the possibility of robot failures, incorporating repair actions to keep the robots operational, similar to the approach introduced by Christensen et al.

(2009). Notably, the researchers applied a multi-level design approach to improve the efficiency of the design process. First, they used a low-fidelity simulator to select the optimal combination of modules. Then, they used a high-fidelity simulator to fine-tune their operational parameters. The experiments were conducted with physical robots, and the authors compared the performance of the automatic design process to that of a manual approach.

As discussed in this section, much of the literature on the automatic offline design of robot swarms has traditionally focused on neuroevolution. This approach has proven highly versatile, with similar neuroevolutionary methods being adapted to address various design problems. Neural networks, with their considerable representational power, have been particularly effective in fitting sensory-motor mappings that enable robots to display a wide range of collective behaviors. Artificial evolution algorithms have shown their effectiveness in harnessing this power. In addition to neuroevolution, automatic modular design has also been used to generate control software for robot swarms, though in fewer studies and often with ad hoc methods tailored to specific design problems.

2.3 The AutoMoDe family of methods

AutoMoDe, short for Automatic Modular Design (Francesca et al. 2014b), is an approach to the design of control software for robot swarms. AutoMoDe methods produce control software for robots by tuning and assembling predefined software modules into a modular control architecture. This process is driven by an optimization algorithm that uses mission-specific performance metrics to search for suitable control software for the robots.

At present, AutoMoDe researchers have developed a whole family of methods by adopting the practices introduced by Francesca et al. (2014b). These methods have been conceived while investigating the application of AutoMoDe to different classes of problems, robot platforms, control architectures, and optimization strategies (Birattari et al. 2021).

In the following, we discuss the most relevant methods developed within the AutoMoDe family. This review of the literature aims to position the methods presented in this dissertation with respect to other AutoMoDe research. First, we elaborate on the background that motivated the development of AutoMoDe and introduce the core ideas behind the approach. Next, we outline the key elements that characterize AutoMoDe methods. Following this, we present `Vanil-`

`la`, the seminal AutoMoDe method that established many of the key elements still commonly used in AutoMoDe research. We then highlight the advances made in the methods introduced after `Vanilla`.

2.3.1 Background and core ideas

The large body of swarm robotics literature on neuroevolution, discussed earlier in this chapter, helped highlight a major issue: the control software produced via neuroevolution is particularly susceptible to the effects of the *reality gap* (Jakobi 1997). That is, it is notably affected by the unavoidable differences between the simulation environment—where the design is conducted—and the real-world environment—where the robots are deployed. Due to the reality gap, the neural networks designed in simulation frequently underperform when transferred to physical robots, sometimes even exhibiting behaviors that differ from those seen during the design process. This problem is widely acknowledged in the field of evolutionary robotics (Floreano et al. 2008). To mitigate the effects of the reality gap, designers commonly intervene in the design process by manually adjusting objective functions, optimization algorithms, or robot platforms (Birattari et al. 2020). While these interventions can be helpful to some extent, it would be far more desirable for the entire design process to be intrinsically robust to the reality gap, without relying solely on manual adjustments. This way, even when manual adjustments are possible, they could improve a process that is already capable of handling the reality gap effectively.

Francesca et al. (2014b) introduced automatic modular design (AutoMoDe) as an alternative to neuroevolution, aiming to address the reality gap problem—see also Chapter 1. AutoMoDe was conceived on the original idea that the inability of a design method to overcome the reality gap should be understood in the context of the *bias/variance* trade-off in machine learning (Geman et al. 1992). In neural network training, a low-bias model typically has a complex architecture that gives the model the power to represent diverse input-output mappings (i.e., a high variance). Conversely, a high-bias model has a restricted architecture, which reduces its power to represent such diverse mappings (i.e., a low variance). Although a low-bias/high-variance model might seem always preferable, it has been shown that it also carries a higher risk of overfitting. The model can perform well on the training set but fails to generalize to new data in the test set. In light of this, Francesca et al. (2014b) conjectured that the performance loss due to the reality gap was a problem akin to the performance loss due to overfitting. The control

software can perform well in simulation (i.e., the training set) and fail to generalize to the physical world (i.e., the test set).

In machine learning, it is known that the risk of overfitting can be reduced by injecting bias into the model—for example, by restricting the model's architecture. Francesca et al. (2014b) adopted this principle in AutoMoDe. They conceived a design process that generates control software within a constrained modular control architecture. This injection of bias inherently limited the range of possible control software that AutoMoDe methods can produce, restricting the solutions to those achievable within the constrained architecture. However, at the same time, this approach gave AutoMoDe the potential to reduce the risk of overfitting and improved its ability to cross the reality gap satisfactorily. This property was first demonstrated with `Vanilla` (Francesca et al. 2014b), the initial implementation of a method within the AutoMoDe family. Since then, it has become the central idea upon which subsequent AutoMoDe methods have been developed.

2.3.2 Characteristic elements

AutoMoDe is most useful when applied within the principles that we believe should guide the automatic off-line design of robot swarms (Birattari et al. 2019)—see Chapter 1. AutoMoDe methods are designed to address a whole class of missions, rather than focusing on a single, specific mission. Once a mission is specified and provided to the design method, the entire design process is executed automatically, with no further human intervention required to produce the control software. In practice, developers of AutoMoDe methods must design and implement a method that can produce control software for the class of missions at hand. Once created, the method can be used repeatedly to automatically generate control software, without additional input from the designer. This independence from ongoing human involvement is what qualifies the design process conducted by AutoMoDe as fully automatic.

AutoMoDe is a general framework that must be specialized to design control software for robot swarms. With `Vanilla`, the first AutoMoDe method, Francesca et al. (2014b) provided a proof of concept for AutoMoDe and established a set of practices to characterize and uniquely identify AutoMoDe methods. A method within this family is characterized by four key elements. First, the class of problems that the method is intended to address, including general mission requirements and common environmental features across missions. Second, the robot platform for which the control software will be designed, comprising both the hardware of the

robot and the control interface available to operate it. Third, the control architecture used to produce the control software, which considers both the predefined software modules and the architecture into which they can be assembled. Fourth, the optimization strategy that drives the design process by selecting and refining control software instances with respect to a mission-specific performance measure. Once these four elements are defined and the method is implemented, it is customary to give each AutoMoDe method a unique name and maintain the method unmodified for further comparison.

2.3.3 Seminal work: AutoMoDe-`Vanilla`

`Vanilla` (Francesca et al. 2014b) is a method for the automatic design of swarms of e-puck robots. It generates control software by selecting, fine-tuning, and assembling preexisting parametric software modules into probabilistic finite-state machines. `Vanilla`'s software modules were designed in a mission-agnostic way. They describe general low-level behaviors and transition conditions that can be combined in various ways to design collective behaviors. `Vanilla` is intended to address classes of missions in which robots must react to environmental cues, such as black and white floor patches or light sources. The specific mission to be addressed must be specified at design time, including a description of the experimental arena where the robots will operate and the performance measure used to evaluate their degree of success.

In the following, we briefly describe the main aspects of `Vanilla`. For a complete description of the method, we refer the reader to the original work of Francesca et al. (2014b).

Robot platform

`Vanilla` produces control software for an extended version of the e-puck robot (Mondada et al. 2009; Garattoni et al. 2015)—see Figure 2.2. More precisely, `Vanilla` produces control software for e-pucks whose functional capabilities are formally defined by the reference model RM 1.1 (Hasselmann et al. 2018a). Francesca et al. (2014b) introduced the notion of a *reference model* in the context of swarm robotics to formally define the specifications of the platform for which a design method can produce control software. In practice, the reference model defines the inputs and outputs of the control interface and their relationship with the hardware of the robot.

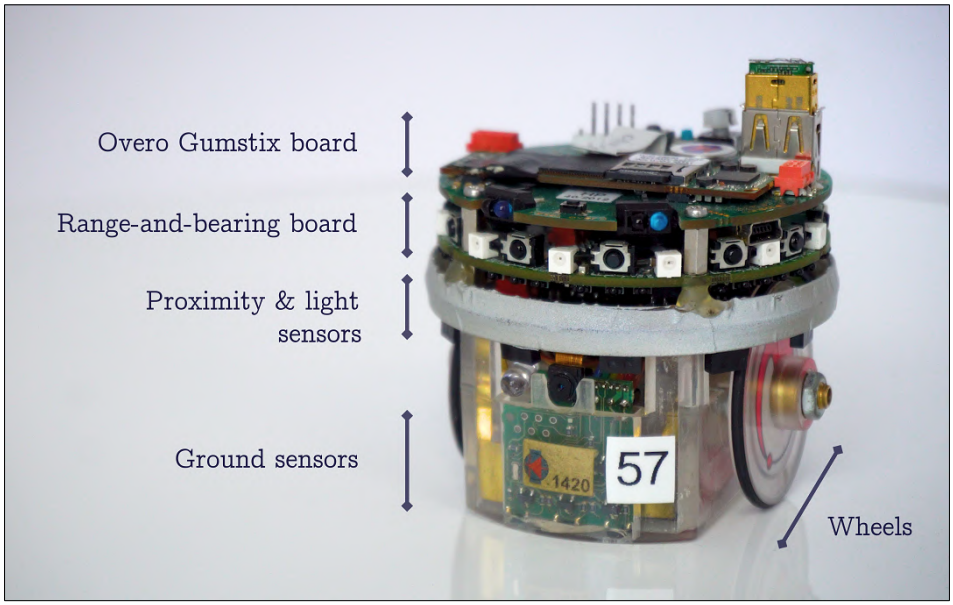


Figure 2.2: The e-puck robot. The picture indicates the set of sensors and actuators defined by the reference model RM 1.1, as introduced with `Vanilla`.

RM 1.1 describes an e-puck endowed with: 8 proximity sensors ($prox_i$) that can detect nearby obstacles; 8 light sensors ($light_i$) that can be used to compute the direction of a light source; 3 ground sensors (gnd_j) that differentiate between gray, black, and white floor; a range-and-bearing board (Gutiérrez et al. 2009) that estimates the number of neighboring peers (n) and their relative aggregate position (V_n); an Overo Gumstix extension board that runs a linux-based operating system; and left and right wheels (v_k), whose velocity can be set independently. The hardware and software of this version of e-puck are thoroughly described by Garattoni et al. (2015). Table 2.1 defines the possible input and output values of the reference model RM 1.1. For a further description of the reference model RM 1.1, see Hasselmann et al. (2018a).

Modular control architecture

`Vanilla` operates on a set of twelve parametric software modules. The set of modules comprises six low-level behaviors—the actions that a robot can execute, and seven transition conditions—the events that trigger the transition between low-level behaviors. `Vanilla` combines low-level behaviors and transition conditions to generate control software of the robots. The parameters of the software modules are automatically tuned during the design process conducted by `Vanilla`. Table 2.2 lists the set of modules on which `Vanilla` operates.

Table 2.1: The control interface for the e-puck according to the reference model RM 1.1 (Francesca et al. 2014b; Hasselmann et al. 2018a).

Input	**Value**	**Description**
$prox_{i\in\{1,\dots,8\}}$	$[0,1]$	reading of proximity sensor i
$light_{i\in\{1,\dots,8\}}$	$[0,1]$	reading of light intensity sensor i
$gnd_{j\in\{1,\dots,3\}}$	$\{black, gray, white\}$	reading of ground sensor j
n	$\{0,\dots,20\}$	number of neighboring robots detected
V_n	$([0.5,20];[0,2]\,\pi\,\mathrm{rad})$	their relative aggregate position
Output	**Value**	**Description**
$v_{k\in\{l,r\}}$	$[-0.16,0.16]\,\mathrm{m\,s^{-1}}$	target linear wheel velocity

Period of the control cycle: 0.1 s.

Table 2.2: `Vanilla`'s software modules. The modules are defined on the basis of the reference model RM 1.1, see Table 2.1.

Low-level behavior	**Parameter**	**Description**
EXPLORATION	$\{\tau\}$	movement by random walk
STOP	n.a.	standstill state
ATTRACTION	$\{\alpha\}$	physics-based attraction to neighboring robots
REPULSION	$\{\alpha\}$	physics-based repulsion from neighboring robots
PHOTOTAXIS	n.a.	physics-based attraction to a light source
ANTI-PHOTOTAXIS	n.a.	physics-based repulsion from a light source
Transition condition	**Parameter**	**Description**
BLACK-FLOOR	$\{\beta\}$	black floor beneath the robot
GRAY-FLOOR	$\{\beta\}$	gray floor beneath the robot
WHITE-FLOOR	$\{\beta\}$	white floor beneath the robot
NEIGHBOR-COUNT	$\{\xi,\eta\}$	number of neighboring robots greater than ξ
INV-NEIGHBOR-COUNT	$\{\xi,\eta\}$	number of neighboring robots less than ξ
FIXED-PROBABILITY	$\{\beta\}$	transition with a fixed probability

The six low-level behaviors are EXPLORATION, STOP, ATTRACTION, REPULSION, PHOTOTAXIS, and ANTI-PHOTOTAXIS. In EXPLORATION, the robot moves with ballistic motion and avoids obstacles by rotating in place if an obstacle is detected. The parameter τ regulates the number of control cycles during which the robot rotates. STOP sets the robot to a standstill behavior. ATTRACTION and REPULSION are physics-based behaviors that drive the robot according to artificial forces that originate at the position of neighboring peers. In both cases, the magnitude of the force is determined by the parameter α. If no other robot is perceived, the robot moves with ballistic motion. PHOTOTAXIS and ANTI-PHOTOTAXIS steadily drive the robot toward or away from light sources, respectively. If no light source is perceived, the robot moves with ballistic motion. EXPLORATION, ATTRACTION, REPULSION, PHOTOTAXIS, and ANTI-PHOTOTAXIS embed physics-based obstacle

avoidance (Borenstein and Koren 1989).

The six transition conditions are BLACK-FLOOR, GRAY-FLOOR, WHITE-FLOOR, NEIGHBOR-COUNT, INV-NEIGHBOR-COUNT, and FIXED-PROBABILITY. In BLACK-FLOOR, GRAY-FLOOR, WHITE-FLOOR, the robot transitions between low-level behaviors with a probability β if the robot steps into regions with black, gray, and white floor, respectively. NEIGHBOR-COUNT and INV-NEIGHBOR-COUNT are transitions that are triggered when the robot perceives a number of neighboring peers that is greater or less than ξ, respectively. FIXED-PROBABILITY is triggered with probability β without further condition.

Automatic design process

`Vanilla` designs control software using F-race (Birattari et al. 2002; Birattari 2009), a general purpose optimization algorithm originally conceived for tuning metaheuristics. In `Vanilla`, F-race explores and evaluates instances of control software according to a mission-specific performance metric—i.e, a measure of the degree of success of the swarm in the mission at hand. F-race starts the design process by randomly sampling the design space for candidate control software. More precisely, F-race samples candidate finite-state machines that result from assembling software modules and instantiating their parameters. `Vanilla` can assemble finite-state machines with up to four behaviors—each of which can have four outgoing transitions at most. During the optimization process, F-race evaluates each finite-state machine on a set of independent problem instances for the mission at hand. To that end, it performs a large number of simulations in which the initial position and orientation of the robots is selected at random. The performance of the swarm is calculated according to the mission-specific performance measure. This measure is used to rank the candidate solutions according to a series of Friedman tests (Conover 1999). The design process ends when F-race has exhausted the maximum number of simulation runs available to evaluate candidate solutions. After that, `Vanilla` returns the best instance of control software found, which is then ported to the physical robots without undergoing any modification.

Experimental setup

Francesca et al. (2014b) compared `Vanilla` with `EvoStick`—a straightforward implementation of the neuroevolutionary approach that operates on the reference model RM 1.1 of the e-puck. They evaluated the ability of the design methods to produce effective control software in two missions: AGGREGATION and FORAGING.

These missions were performed by a swarm of 20 e-pucks. For each mission, the authors conducted 20 automatic design processes to obtain 20 instances of control software—10 for each design method. They experimented with three different budgets of simulation runs to produce the instances of control software: 10 000, 50 000, and 200 000. After generating all the control software instances, they evaluated their performance in both simulation and with physical robots. Simulations during the design process and post-evaluation were performed using the ARGoS3 simulator along with the argos3-epuck library (Garattoni et al. 2015). ARGoS3 (Pinciroli et al. 2012) is a fast, parallel, multi-engine simulator specifically designed for multi-robot systems and is widely used in swarm robotics (Pitonakova et al. 2018).

In the experiments with physical robots, the authors used a tracking system (Stranieri et al. 2013) to monitor the robots in the experimental arena. For both simulation and physical experiments, the authors computed the performance of the swarm using ARGoS3. In the simulation runs, ARGoS3 estimated the position of the simulated robots at every time step and then calculated the swarm's performance based on the mission-specific measure. In the experiments with physical robots, the tracking system provided ARGoS3 with the positions of the robots at every time step. ARGoS3 then processed this information and returned a performance value. The computation made by ARGoS3 to estimate the performance of the swarm was consistent in both simulations and physical experiments. By maintaining the same evaluation procedure in the two settings, the authors could also study the effects of the reality gap on the automatically generated control software.

Results

In this first study with AutoMoDe, the results showed that `Vanilla` outperformed `EvoStick` in the automatic design of robot swarms. The control software generated by `EvoStick` outperformed `Vanilla` in simulations, but its performance significantly dropped when transferred to physical robots. Conversely, `Vanilla` maintained a similar performance both in simulation and in reality—outperforming `EvoStick` in the latter. This result supported the original idea behind AutoMoDe. By restricting the design space to a limited set of combinations of software modules, `Vanilla` could reduce the risk of overfitting the simulation environment. This, in turn, allowed `Vanilla` to cross the reality gap more effectively. On the other hand, `EvoStick` had the freedom to produce control software that was highly specialized

for the simulator—explaining its higher performance in simulation evaluations—but failed to generalize effectively to the physical robots. Notably, this study identified for the first time that the effects of the reality gap can be method-dependent. That is, the same reality gap can impact the performance of different automatic design methods in different ways. The relationship between the reality gap and machine learning's overfitting in `EvoStick` was further discussed by Birattari et al. (2016).

2.3.4 Research on the optimization process

In a follow-up study, Francesca et al. (2014a) further investigated the ability of `Vanilla` to generate control software compared to `EvoStick`, but also to human designers. They invited a group of swarm robotics practitioners to (i) propose five missions that `Vanilla` must address and (ii) produce themselves control software for the proposed missions. The missions were conceived for the capabilities of the e-puck defined by the reference model RM 1.1 and resulted in variations of aggregation, coverage, and decision-making. In the study, the authors devised two protocols for human designers to produce control software for a swarm of e-pucks: `U-Human` and `C-Human`. In `U-Human`, the designers were given unrestricted access to the robot's API. They could use it to program the robots directly, without any limitations on the software they could develop. In `C-Human`, however, the designers were restricted to using only the software modules and architecture on which `Vanilla` operates. The results showed that `Vanilla` outperformed `U-Human` and `EvoStick`, but the three were outperformed by `C-Human`. This outcome indicated that the human designers benefited from the reduced search space provided by `C-Human`. As `Vanilla` and `C-Human` operated on the same robot platform and software modules, the authors concluded that the optimization algorithm used in `Vanilla` was the element preventing it from outperforming the human designers.

Francesca et al. (2015) introduced `Chocolate`, an AutoMoDe method that improved upon `Vanilla` by replacing F-race with Iterated F-race (Balaprakash et al. 2007; Birattari et al. 2010; López-Ibáñez et al. 2016), a more sophisticated version of F-race. The only new element in `Chocolate` was the optimization algorithm; otherwise, it was identical to `Vanilla`. The authors extended their 2014 study to include `Chocolate`, comparing its performance against `Vanilla`, `EvoStick`, `U-Human`, and `C-Human`. They conducted experiments using the same missions defined by the human experts. The results showed that `Chocolate` outperformed all four alternative methods in producing control software for robot swarms. Although `Vanilla` was introduced first, the more powerful optimization

algorithm of Chocolate has made it the *de facto* standard method within the AutoMoDe family. Chocolate commonly serves as a base for the development of new AutoMoDe methods and is a recurrent baseline for assessing their performance.

Kuckling et al. (2019, 2020b) introduced IcePop, a variation of Chocolate in which the optimization algorithm is replaced by simulated annealing (Kirkpatrick et al. 1983)—a stochastic local search metaheuristic. The authors conducted simulation experiments on two missions previously studied with Vanilla and Chocolate, which involved foraging and aggregation. The results showed that simulated annealing was a viable algorithm for generating control software for robot swarms, even outperforming Chocolate in some cases. When the control software was evaluated against estimators for the impact of the reality gap (Ligot and Birattari 2020), IcePop appeared to be more susceptible to its effects than Chocolate.

In a follow-up study, Kuckling et al. (2020a) further explored the application of local search optimization as an alternative to Iterated F-race. They used iterated improvement (Gendreau and Potvin 2019) to generate control software for e-pucks that could operate with two control architectures: probabilistic finite state machines and behavior trees. In the two architectures, the software modules remained the same as in Chocolate. The performance of iterated improvement was compared with that of Iterated F-race and a genetic programming implementation. The experiments were conducted in simulation on missions similar to those used for Vanilla and Chocolate. The results showed that, as with simulated annealing in IcePop, iterated improvement is also a viable option to produce control software for robot swarms. The design process conducted with iterated improvement outperformed Chocolate when the design budgets were sufficiently large.

Hasselmann et al. (2021) compared Chocolate against several popular neuroevolutionary methods. The authors adapted versions of NEAT (Stanley and Miikkulainen 2002), CMA-ES (Hansen and Ostermeier 2001), and xNES (Glasmachers et al. 2010) to operate with the reference model RM 1.1 of the e-puck, and compared their performance against Chocolate and EvoStick. The experiments were conducted using missions defined for Vanilla and Chocolate. The results showed that, although they are more advanced methods, NEAT, CMA-ES, and xNES did not offer any clear advantage over EvoStick when used off the shelf. All neuroevolutionary methods performed similarly, but Chocolate outperformed them all. The study also showed that NEAT, CMA-ES, and xNES were highly susceptible to the reality gap, similar to EvoStick. Chocolate showed a better ability to cross the reality gap compared to the neuroevolutionary methods.

Cambier and Ferrante (2022) explored the application of evolutionary optimiza-

tion in the design process by introducing `Pomodoro`—a sub-family of AutoMoDe methods. `Pomodoro` consists of three methods that adapt `Chocolate` by replacing Iterated F-race with an alternative evolutionary algorithm. The first method, `BetterBoy`, uses a standard genetic algorithm. The second, `EarlyGirl`, uses grammatical evolution (Ferrante et al. 2013). The third, `BigRainbow`, uses a version of differential evolution (Tomczak et al. 2020). The authors conducted simulation experiments on missions previously studied with `Vanilla` and `Chocolate`, comparing the performance of these evolutionary-based methods against `Chocolate`. The results showed that evolutionary optimization strategies are a viable option for modular design. The three methods in `Pomodoro` generally performed similarly to `Chocolate`, with `Chocolate` outperforming them in some cases.

In AutoMoDe, the optimization process is driven by a performance measure that indicates how successfully the swarm performs its mission. Traditionally, this performance measure has been an objective function manually defined by the designer. The problem is that defining an objective function that accurately reflects the desired swarm behavior is a complex task that requires expert knowledge. This challenge has motivated new research into alternative approaches for specifying the performance measure that drives the optimization process. Bozhinoski and Birattari (2022) developed a simplified structured language that allows mission specification using common English text. This language operates on a dictionary that translates the text input into a mission specification compatible with AutoMoDe. The dictionary identifies keywords in the text and uses them to select from a set of parameterizable environmental features and predefined mission-specific objective functions. This approach allows the user to simply provide a textual description of the mission for the design process, without the need to formulate an objective function. The authors conducted experiments with `Chocolate`, showing that the design process could be successfully performed using text-based mission specifications.

In a recent study (Gharbi et al. 2023), we also explored a more intuitive way to specify missions than providing an objective function. Instead of relying on a mathematical objective function or a textural description, we adapted `Chocolate` to operate over a set of user demonstrations of the desired swarm behavior. We introduced Demo-`Cho`, an advanced version of `Chocolate` that enables the automatic design of robot swarms starting from user demonstrations. To specify a mission, the user demonstrates the desired final spatial distribution of the robots at the end of the experimental run. Demo-`Cho` then generates control software that allows the swarm to achieve this spatial distribution. We evaluated Demo-`Cho`

and Chocolate on missions previously defined for AutoMoDe. For comparison, we evaluated the two methods using objective functions that had been used in earlier studies to produce similar spatial distributions. The results showed that Demo-Cho could design control software that performed on par with Chocolate, even without a mathematical objective function.

2.3.5 Research on the robot platform and its capabilities

The first to study the design of communication behaviors in AutoMoDe were Hasselmann et al. (2018b) and Hasselmann and Birattari (2020). The authors adapted Chocolate to operate with a new reference model for the e-puck, adding communication capabilities. They introduced the reference model RM 2, an extension of RM 1.1 (see Table 2.1), which enables the e-puck to broadcast and receive messages through its range-and-bearing board. The design method based on RM 2, named Gianduja, not only used a different robot platform but also modified Chocolate's low-level behaviors. The new modules included parameters to decide whether a message is broadcast during the execution of a module and to define the message content, with up to three bits per message. The authors conducted experiments on three missions in which communication-based coordination could provide the swarm with a performance advantage. The authors compared the performance of Gianduja with Chocolate and with versions of EvoStick that were adapted to use the capabilities defined in RM 2. The results showed that Gianduja designed robot swarms that relied on single-bit communication to perform their mission effectively. In a follow-up study, the authors conducted experiments on missions in which the swarm could potentially benefit from two- or three-bit communication. However, the more complex message structures did not show an advantage over the results already achieved with a single bit.

Salman et al. (2019) investigated the concurrent design of control software and hardware. They introduced Waffle, an extension of Chocolate that not only selects the software modules to be assembled into the finite state machine but also some of the hardware components for the e-puck. To design control software, Waffle can select from the software modules of Chocolate. To design the hardware of the robot, it can select from different range-and-bearing boards, each offering varying transmission/reception error rates, power consumption levels, and estimated market prices. The authors conducted simulation experiments using Waffle in aggregation and foraging missions previously studied with Chocolate. In addition to the standard objective function, they introduced economic constraints

into the optimization process. This required `Waffle` to balance the selection of fewer robots equipped with expensive high performance boards against the use of more robots with cost-effective alternatives. The results showed that `Waffle` could select the most suitable hardware for each mission while simultaneously designing the control software to operate them effectively.

Until recently, AutoMoDe had only been applied to designing collective behaviors for swarms of e-pucks, including the experiments presented in this dissertation. However, in a recent work (Kegeleirs et al. 2023, 2024b), we showed that `Chocolate` could be successfully transferred to a new robot platform called Mercator (Kegeleirs et al. 2022). Mercator is a small educational robot with a more advanced set of sensors and actuators compared to the e-puck. It is equipped with a LIDAR, an RGB ground sensor, an infrared depth camera, and a Raspberry Pi 4 computing board. Additionally, Mercator is nearly three times larger and can move three times faster than the e-puck. We were able to apply `Chocolate` directly to a swarm of Mercators by relying on the concept of the reference model. By reformatting Mercator's sensor data to match the control input requirements of the e-puck's reference model RM 1.1, we aligned the functional capabilities of the two robots. This adaptation allowed `Chocolate` to operate on the same software modules and control architecture for both e-puck and Mercator. We conducted experiments on three missions where `Chocolate` was tasked with designing collective behaviors for both e-pucks and Mercators, adjusting the arena size to fit the workspace needed for each robot. The results showed that `Chocolate` could effectively design control software for the Mercators, even though it was not originally developed for them.

2.3.6 Research on the modular control architecture

Research on the modular architecture of AutoMoDe has focused on exploring suitable control architectures, defining new software modules, or a combination of both. Here, we focus on methods that introduced new software modules and control architectures without introducing new hardware. Studies that introduced new modules and hardware were discussed earlier.

The software modules in AutoMoDe can be customized to develop methods specialized in addressing specific design problems. In Spaey et al. (2019, 2020), we investigated whether `Chocolate` could design more effective exploration behaviors by incorporating different random-walk behaviors into its modules. Originally, `Chocolate`'s low-level behaviors used a simple ballistic motion as the default random walk behavior. We relaxed this design choice by introducing `Coconut`, a method

that allows the design process to choose from various random walk behaviors and obstacle avoidance strategies. To achieve this, we modified `Chocolate`'s modules by adding parameters that enabled the selection of these alternative behaviors, previously adapted for the e-puck (Kegeleirs et al. 2019). We conducted experiments with `Coconut` and `Chocolate` on aggregation, foraging, and coverage missions in both bounded and unbounded environments—robots could potentially escape the arena in the latter. The results showed that `Coconut` successfully designed effective exploration strategies by selecting the most suitable random walk behavior for each mission. However, this did not provide a significant performance improvement over `Chocolate`.

Kuckling et al. (2018a) and Ligot et al. (2020b) investigated the use of behavior trees as a control architecture in AutoMoDe. They introduced `Maple`, a modified version of `Chocolate` that replaced the probabilistic finite-state machines with behavior trees. In other aspects, `Maple` and `Chocolate` remained the same. The authors conducted experiments on foraging and aggregation missions, comparing the performance of `Maple` with `Chocolate` and `EvoStick`. The results showed that `Maple` could design effective control software that performed similarly to `Chocolate`, with the two modular methods outperforming `EvoStick`. The control software produced by `Maple`, in the form of behavior trees, showed similar capabilities to cross the reality gap as that produced by `Chocolate`. This study contributed to support the idea that AutoMoDe's ability to cross the reality gap is closely linked to the modular nature of the design process, which can adopt architectures other than the probabilistic finite-state machine.

`Chocolate`'s modules proved to be functional for `Maple`, yet they missed some necessary properties to fully leverage the behavior tree formalism. In particular, the low-level behaviors missed success or failure conditions to drive the control cycle within the tree. This motivated Kuckling et al. (2021b, 2022) to develop `Cedrata`, an AutoMoDe method whose software modules were specifically designed for behavior trees. `Cedrata` operates on a modified version of the reference model RM 2 (Hasselmann et al. 2018b). The modules developed for `Cedrata` incorporated the required success/failure conditions and the communication capabilities endowed by RM 2. In the study, the authors considered three implementations of `Cedrata`, each using a different optimization algorithm: Iterated F-race, genetic programming, and grammatical evolution. The authors conducted experiments on missions in which the swarm was expected to benefit from communication-based coordination. As in the original studies on `Vanilla` and `Chocolate`, the implementations of `Cedrata` were compared to control software produced using the `C-Human` protocol.

The results showed that Cedrata could generate high-performing control software with the new modules, even outperforming human designers. However, Cedrata failed to design collective behaviors that effectively leveraged communication to perform the missions. Instead, the robots relied on simpler forms of coordination.

AutoMoDe is an automatic design process that operates without human intervention, but has traditionally relied on the manual development of the software modules it operates on. Ligot et al. (2020a) proposed an alternative approach to the development of AutoMoDe modules. They introduced Arlequin, a method in which low-level behaviors are generated through neuroevolution rather than manually programmed. In Arlequin, the authors used a neuroevolutionary strategy to replicate Chocolate's low-level behaviors. In other aspects, Arlequin operates similarly to Chocolate. Arlequin uses Iterated F-race to assemble the neuroevolution-based modules into probabilistic finite-state machines. The authors compared the performance of Arlequin with that of Chocolate and EvoStick. The results showed that producing modules via neuroevolution is a viable alternative to manually programming them, potentially increasing the degree of automation in AutoMoDe. Arlequin also showed a higher sensitivity to the reality gap than Chocolate, but less than EvoStick. This indicated that a degree of modularity could also provide an advantage to neuroevolutionary design processes.

Hasselmann et al. (2023) further investigated the application of neuroevolution to develop software modules in AutoMoDe. They introduced Nata, a design method that fully automates the creation of low-level behaviors and transition conditions. The authors applied a novelty search approach to discover a diverse set of task-agnostic low-level behaviors and transitions compatible with reference model RM 1.1. This process resulted in a repertoire of 670 low-level behaviors and 5 transition conditions that Nata could assemble into finite-state machines. Therefore, unlike Arlequin, Nata is not restricted to operate on low-level behaviors that mimic those of Chocolate. The authors compared the performance of Nata with that of Arlequin, Chocolate, and EvoStick. The results showed that the repertoire of modules generated by Nata enabled the method to produce control software that can perform missions previously studied with Chocolate. Although Nata did not outperform Chocolate, it successfully performed the same missions with less human intervention.

2.4 Design problems

In this section, we organize and comment on literature that provides context for the design problems to be addressed in the dissertation. Our aim is not to be exhaustive, but to highlight significant approaches and challenges in these topics. We first discuss the design of robot swarms that rely on color-based signaling to coordinate, a design problem studied with `TuttiFrutti` in Chapter 3. After, we discuss the realization of robot swarms under concurrent design criteria, a design problem studied with `Mandarina` in Chapter 4. We finish the chapter by dicussing relevant literature on the design of spatial organization behaviors, stigmergy-based behaviors, shepherding behaviors, and the design of robot swarms by demonstration—design problems studied with `Mate`, `Habanero`, `Pistacchio`, and DTF-`MO` in Chapter 5.

2.4.1 Design of robot swarms that coordinate via color signals

Robots that can display or perceive colors have been widely used to demonstrate collective behaviors in swarm robotics. The literature on robot swarms that use visual information is extensive. We focus here on studies in which robots can both display and perceive color signals to coordinate. We exclude from this discussion any system in which robots only perceive visual information but do not display it—such as in the work by Waibel et al. (2009), Gauci et al. (2014b), Chen et al. (2015), Lopes et al. (2016), and Jones et al. (2019).

Designers of robot swarms commonly use color lights to represent specific information that the robots must identify, process, and/or transmit—the nature of the information varies from one study to another and is used ad hoc to obtain a particular behavior. For example, Nouyan et al. (2008) designed a swarm that connects locations of interest by establishing a chain of robots that act as waypoints for their peers. They conducted two experiments in which robots use colors differently: in the first experiment, robots repeat a pattern of 3 colors along the chain to indicate the direction in which the peers should move; in the second one, robots use colors to inform their peers about a location of interest. Mathews et al. (2017) designed a swarm in which robots self-organize in mergeable structures. In their experiments, robots react to colored objects in their environment and display color signals that indicate their location. Garattoni and Birattari (2018) designed a robot swarm that autonomously identifies and performs sequences of tasks. In

their experiments, robots emit color signals to coordinate their collective action and associate objects that display a particular color with a task in the sequence.

In a more general sense, one can find a similar approach in swarms that exhibit self-assembly and morphogenesis (O'Grady et al. 2009, 2010; Mathews et al. 2017, 2019), collective fault detection (Christensen et al. 2009; Mathews et al. 2017), collective exploration (Nouyan et al. 2009; Ducatelle et al. 2011; Dorigo et al. 2013; Garattoni and Birattari 2018), collective transport (Nouyan et al. 2009; Dorigo et al. 2013), coordinated motion (Ferrante et al. 2010; Mathews et al. 2017, 2019), human-swarm interaction (Giusti et al. 2012; Podevijn et al. 2012), chain formation (Nouyan et al. 2008, 2009; Garattoni and Birattari 2018), group size regulation (Pinciroli et al. 2009), task allocation (Pini et al. 2011a,b, 2014; Brutschy et al. 2015; Garattoni and Birattari 2018), object clustering (Allwright et al. 2014; Pini et al. 2014), and foraging (Brambilla et al. 2014)—according to the taxonomy proposed by Brambilla et al. (2013). In these studies, designers manually established ad hoc relationships between the colors that a robot can perceive and the corresponding behavior that a robot must adopt when it perceives them. In other words, designers used mission-specific knowledge and expertise to define communication protocols that enable the robots to coordinate effectively. In this dissertation, conversely, we investigated whether AutoMoDe can establish similar relationships in a fully automatic way. This would allow designers of robot swarms to achieve robot coordination without requiring mission-specific expertise to design the communication strategy. Transitioning from ad hoc methods to a systematic and automated process is a technological step forward.[1]

Color lights are a signaling mechanism to operationalize the ability of an automatic design method to establish signal-based interactions between physical robots—without relying on abstract or simulated communication. Although other mechanisms have been used in the literature, e.g., infrared (Hasselmann and Birattari 2020) or sound (Trianni and Nolfi 2009), using color lights has additional advantages to deploying physical robot swarms. First, the capability of displaying and perceiving colors is platform independent and generalizes across different missions—robot platforms used in swarm robotics often include LEDs and cam-

[1] Automatically learning communication protocols for artificial agents is an area of interest in related fields, such as multi-agent reinforcement learning. However, popular literature on the topic remains constrained to simulated scenarios (Foerster et al. 2016; Das et al. 2019). In the context of swarm robotics, very recent advances in applying Large Language Models (LLMs) to robot swarms (Strobel et al. 2024) can open new possibilities for leveraging communication, enabling robot swarms to interact more effectively not only with each other but also with human operators.

eras (Nedjah and Silva Junior 2019). Second, colors facilitate the conception and implementation of complex missions—colored environments can be implemented in various manners (Mayet et al. 2010; Brutschy et al. 2012; Allwright et al. 2014; Brutschy et al. 2015; Soleymani et al. 2015). Finally, colors simplify the visualization and monitoring of robot swarms (Nedjah and Silva Junior 2019)—a property relevant to the human understandability of collective behaviors (Kolling et al. 2016).

The design of collective behaviors for robots that can display and perceive color signals has not been studied in a systematic way. We argue that evaluating automatic design methods across multiple missions—where robots communicate and respond to color-based information—provides an ideal benchmark. In an automatic design process, color signals do not inherently carry meaning. The challenge for any method is to assign meaning to these signals and operationalize them during the design process. The method must produce control software that uses color signals selectively in specific contexts, triggering appropriate robot behaviors that help the swarm meet the mission specifications. Here, we center our attention to methods framed within the principles of the automatic offline design of robot swarms (Birattari et al. 2019).

Some of the neuroevolution studies mentioned in Section 2.2 were conducted with robots that can display and perceive colors. For example, Floreano et al. (2007) evolved communication behaviors for a swarm that must perform a foraging mission. Ampatzis et al. (2009) evolved self-assembly behaviors with a team of two robots. Sperati et al. (2008, 2011) evolved coordinated motion behaviors with a group of three robots, and afterward developed a dynamic chain of robots to perform a foraging-like mission. Trianni and López-Ibáñez (2015) used multi-objective optimization to evolve flocking and a two-robot collaborative behavior. These studies belong in semiautomatic design rather than fully automatic design. The design methods under analysis were not tested for their ability to generate control software autonomously—i.e., without human intervention. The authors either focused on a single mission (Floreano et al. 2007; Ampatzis et al. 2009; Sperati et al. 2011) or modified the design methods and robot platforms to investigate multiple missions (Sperati et al. 2008; Trianni and López-Ibáñez 2015).

In related neuroevolutionary studies, mission-specific bias has been manually introduced into how robots display and perceive color signals. For example, the robots display colors that are manually defined at the beginning of the experiment (Ampatzis et al. 2009). Otherwise, they display color-based information that is manually encoded by the researchers (Sperati et al. 2011; Trianni and López-Ibáñez 2015). Alternatively, the perception capabilities of the robots are

adjusted to ease the design of specific behaviors on a per-mission basis (Sperati et al. 2008, 2011). These studies also did not fully explore the potential of using color-based information in the automatic design of collective behaviors. In fact, these previous works were limited to producing control software for robots that can display and perceive only one (Floreano et al. 2007; Sperati et al. 2008; Ampatzis et al. 2009) or at most two simultaneous colors (Sperati et al. 2011; Trianni and López-Ibáñez 2015).

Experimental environments

Swarm robotics research is conducted mainly under controlled laboratory conditions. Researchers design robot swarms to exhibit specific collective behaviors in simplified ad hoc scenarios. These scenarios typically involve convex and bounded spaces that are populated with objects that the robots can perceive, identify, and interact with. The characteristics of the environment where the robots operate partially condition the collective behaviors that the swarm can exhibit. Indeed, the scenario must provide the elements that enable the emergence of a desired collective behavior. For example, a robot swarm that navigates its environment using color cues requires colored objects to operate correctly.

As already described in this section, swarm robotics platforms often have cameras that give the robots the capability to perceive colored objects. Designers of robot swarms commonly conceive experiments in which these objects represent information that is relevant to the task that a swarm must perform. For example, objects that display different colors can indicate regions of the scenario in which robots should act differently. We identify two types of objects that are used to this purpose: non-programmable simple objects like prints, or wooden and plastic items (Francesca et al. 2014b, 2015; Castelló Ferrer et al. 2016; Kegeleirs et al. 2019; Jones et al. 2019); and programmable devices that can modify their characteristics in runtime and/or actively interact with the robots (Nouyan et al. 2009; Brutschy et al. 2015; Mathews et al. 2017; Allwright et al. 2019). On the one hand, simple objects are cost-effective and easy to fabricate. However, it is difficult to use them beyond the context of the specific experiment for which they are created. On the other hand, programmable devices require additional configuration steps in preparing an experimental setup. Yet, they are more versatile, reusable, and enable the design of experiments that consider time-varying scenarios.

In the swarm robotics literature, researchers typically focus on the robots and pay less attention to the components of the experimental arena where they operate.

Indeed, few systems have been formally released that enable the fast creation of experimental arenas for robots that perceive colored objects—we refer to systems that comprise programmable devices. Within these few systems, Allwright et al. (2019) developed a multi-robot construction system in which robots assemble small programmable RGB cubic blocks named SRoCS. The blocks in SRoCS are modules that embed a micro-controller and can communicate with robots by using Near-field Communication (NFC) modules, Xbee radio transceivers, and by changing the color of RGB LEDs. Similarly, Brutschy et al. (2015) introduced the Task Abstraction Module (TAM). The TAM is a programmable booth-shaped smart device with which robots can interact by stepping inside. The TAM can communicate with the robots using infrared transceivers and by displaying colors using RGB LEDs. It also embeds Xbee radio transceivers that enable the communication between TAMs.

Systems such as SRoCS and the TAM exemplify the benefits of using objects that display colors in swarm robotics experiments. Although these devices are limited to providing individual interaction points for the robots, they offer a flexible way to configure the workspace. This flexibility greatly facilitates the execution of complex and innovative experiments, such as those in Garattoni and Birattari (2018).

2.4.2 Design of robot swarms under concurrent design criteria

The concept of designing robot swarms under concurrent design criteria applies primarily to optimization-based design approaches. Traditional trial-and-error methods for manually designing collective behaviors cannot be easily analyzed within this framework. Manual design is largely driven by the designer's intuition and expertise, which are difficult to quantify and may not directly align with well-defined performance measures—i.e. the design criteria.

In optimization-based design, the performance measure is formally defined as part of the mission specification. The literature provides various examples in which, given a particular mission, the performance measure is formally defined as (i) a single function that measures the degree of success of the robots in the mission at hand or (ii) multiple functions that indicate whether the robots attain a set of objectives and satisfy a set of constraints. When multiple functions are considered, these are ultimately concurrent design criteria to be satisfied by the robot swarm. Examples of missions specified as a single function are provided by Francesca et al.

(2014b, 2015), Garzón Ramos and Birattari (2020), Hasselmann and Birattari (2020), Kuckling et al. (2021b), and Hasselmann et al. (2021). Otherwise, examples of missions specified through multiple functions are provided by Quinn et al. (2003), Christensen and Dorigo (2006), Marocco and Nolfi (2007), Ampatzis et al. (2009), Trianni and Nolfi (2009), Gomes et al. (2013), Duarte et al. (2014, 2016), Trianni and López-Ibáñez (2015), Francesca et al. (2015), Gomes and Christensen (2018), Jones et al. (2018, 2019), Garzón Ramos and Birattari (2020), Hasselmann and Birattari (2020), and Kuckling et al. (2021b).

Researchers in optimization-based design rarely adopt multi-criteria optimization methods to address problems with concurrent design criteria. The typical approach to address these problems is to aggregate the criteria into a single performance measure and then apply single-criterion optimization methods (Trianni and López-Ibáñez 2015). Previous studies in the automatic off-line design of robot swarms—both with neuroevolution and AutoMoDe—have indirectly addressed missions with concurrent design criteria in this way (Francesca et al. 2015; Garzón Ramos and Birattari 2020; Hasselmann and Birattari 2020). We believe that the current taxonomy of the possible approaches to optimization-based design (Birattari et al. 2020) should be extended further to include the single/multi-criteria dichotomy. Indeed, on-line and off-line methods, as well as semi-automatic and automatic ones, face the challenge of enabling the design of robot swarms that comply with concurrent requirements specified by a designer.

Trianni and López-Ibáñez (2015) described how designers often encode concurrent *mission-specific* and *mission-generic* design criteria into single objective functions—see also Doncieux and Mouret (2014). *Mission-specific* criteria are meant to express preferences on the desired outcome of the mission and/or on the behavior of the robots—for example, the number of objects collected in foraging (Francesca et al. 2014b; Jones et al. 2018, 2019) or the time needed by the robots to aggregate (Francesca et al. 2015; Gomes and Christensen 2018). On the other hand, *mission-generic* criteria are independent of the mission and are meant to express preferences on the design process itself—for example, the diversity and/or complexity of the control software that is produced (Gomes et al. 2013; Jones et al. 2018, 2019) or the financial cost of realizing a certain robot swarm (Salman et al. 2019). Many studies indirectly consider the optimization-based design of robot swarms under concurrent design criteria. However, although these studies frame multi-criteria design problems, their focus is not on investigating multi-objective optimization approaches to address them. The formulation of a multi-criteria problem is often only a convenient tool that favors the materialization of a desired

collective behavior.

Little research has been devoted to formally studying the advantages and limitations of addressing multi-criteria design with multi-objective optimization methods. Trianni and López-Ibáñez (2015) made a significant contribution to these ideas. As described in Section 2.2, they used simulations and neuroevolution to produce control software for robot swarms. They experimented in missions related to collective motion and an abstraction of the collaborative stick-pulling experiment (Ijspeert et al. 2001). In each mission, the performance was measured with respect to two objective functions—i.e., the design criteria. The study compared a weighted sum approach (Marler and Arora 2010) with a multi-objective approach based on the estimation of the Pareto set. In the weighted sum approach, the design criteria were mapped into terms that were subsequently aggregated into a single objective function. The authors explored various combinations of the weights associated with each term. In the multi-objective approach, the design criteria were mapped into terms that were used independently to compute the *hypervolume* (Zitzler and Thiele 1998)—i.e., the size of the design space that is dominated by the solutions obtained. In the two cases, the result of the design process was a set of solutions from which to manually select a desired one with respect to a performance preference.

Trianni and López-Ibáñez concluded that a weighted sum is appropriate when the designer is able to properly set the weights. Indeed, setting properly the weights could be possible in some cases due to the existence of prior knowledge or as a result of a trial-and-error process—for example, by testing various combinations of weights. The authors also concluded that a multi-objective method is more suitable when prior knowledge is not available or when it is not even possible to find a suitable combination of weights. For example, this is the case of criteria that vary in different proportion to each other and result in non-comparable or non-linearly related scales. The literature shows that the common approach to multi-criteria design remains the mapping of mission-specific and mission-generic design preferences into components of a single objective function—i.e., linear scalarization or weighted sum. Indeed, the design methods commonly require casting the multi-criteria design problem into a single-objective optimization one. This is the approach adopted in a large share of studies that belong in semi-automatic design and/or neuroevolution, but also in AutoMoDe. For semi-automatic design and/or neuroevolution, see the work of Quinn et al. (2003), Christensen and Dorigo (2006), Marocco and Nolfi (2007), Ampatzis et al. (2009), Trianni and Nolfi (2009), Duarte et al. (2014, 2016), Trianni and López-Ibáñez (2015), and Jones et al. (2018, 2019). For AutoMoDe, see the work of Francesca et al. (2015), Garzón Ramos and Birattari (2020), Hasselmann

and Birattari (2020), and Kuckling et al. (2021b).

In most studies that focus on neuroevolution, the authors do not describe the steps they followed to conceive the objective function of the multi-criteria design problem that is considered. An example of this is the recurrent use of functions that combine a mission-specific objective with a second term that penalizes collisions (Ampatzis et al. 2009; Duarte et al. 2014, 2016). Authors rarely state whether the objective function is (i) the formal specification of the mission in mathematical terms or it is rather (ii) a function engineered on the basis of prior/domain knowledge to guide the optimization process toward a desired solution.

In studies focusing on modular design, recent work considered missions that were specified as multi-criteria problems more explicitly by defining a set of sub-missions to be achieved (Francesca et al. 2015; Garzón Ramos and Birattari 2020; Hasselmann and Birattari 2020; Kuckling et al. 2021b). In these missions, the overall performance of the swarm is measured by a weighted sum of its performance in two sub-missions—which are executed either simultaneously (Francesca et al. 2015) or sequentially (Garzón Ramos and Birattari 2020; Hasselmann and Birattari 2020; Kuckling et al. 2021b). For example, Hasselmann and Birattari (2020) studied the design of a robot swarm that must change its behavior after finding a given marker in its environment. Initially, the robots must keep a steady motion; after a robot finds the marker, all robots must stop in place. In this mission, the two sub-missions must be performed sequentially and the overall performance of the swarm is measured by a weighted sum of its performance on the two.

Sequences of missions: a design problem with concurrent design criteria

Designing robot swarms that perform sequences of missions is in its own nature a design problem that requires meeting concurrent design criteria. If done manually, a user must (i) produce control software that performs well in all the sub-missions considered and (ii) devise efficient transitions from one to another. This is a more complex design problem than designing a robot swarm that performs a single mission. Little research exists on how to automatically design robot swarms that must perform sequences of missions, transitioning from one sub-mission to another. Previous related studies have assumed that the sub-missions can be addressed separately, and the transitions and/or order in the sequence are known beforehand. As noted in Section 2.2, Duarte et al. (2016) evolved individual neural networks to perform specific desired behaviors. They then assembled these neural networks into a single modular architecture capable of executing a mission composed by

a sequence of sub-missions. Garattoni and Birattari (2018) presented a robot swarm that is able to collectively sequence sub-missions at run-time, without the need to know a priori the order in which they should be executed. However, the behaviors needed to perform the sub-missions were manually programmed in advance. Further examples of robot swarms that perform sequences of missions are available in Krieger et al. (2000), Nouyan et al. (2009), Schmickl et al. (2011), and Castelló Ferrer et al. (2021).

2.4.3 Design of spatially-organizing collective behaviors

Spatially-organizing behaviors are a class of collective behaviors that focus on the organization and distribution of robots and objects in space (Brambilla et al. 2013). In this dissertation, we are mainly interested in the design of spatially-organizing behaviors that resemble those observed in previous studies on pattern formation (Lopes et al. 2014, 2016, 2017). Pattern-formation behaviors are commonly realized within the framework of virtual physics with methods based on artificial potential fields (Spears and Gordon 1999). In this approach, robots react to attractive and/or repulsive virtual forces that originate in the location of perceived peers. The artificial potential field approach was proposed by Khatib (1986), and then adapted to swarm robotics by Spears and Gordon (1999). This approach to realize spatially-organizing behaviors has been applied in the context of monitoring and surveillance (Spears et al. 2004; Shucker and Bennett 2007), distributed sensing and actuation (Lochmatter et al. 2013), coverage (Howard et al. 2002; Shucker and Bennett 2007), and collective motion (Spears et al. 2004; Turgut et al. 2008; Ferrante et al. 2012) among others. In this design problem, researchers also commonly produce control software for the robots by following a manual *ad hoc* process.

The literature on the optimization-based design of spatially-organizing behaviors belongs mainly in evolutionary robotics and neuroevolution. For example, as discussed in Section 2.2, Duarte et al. (2014, 2016) produced hybrid and hierarchical control software by evolving individual behaviors that are then executed via a behavior arbitrator. The authors used this method to design spatially-organizing behaviors for a swarm that detects intruders in its workspace—both in simulation and with physical robots. Similar approaches have been adopted in the design of spatially-organizing behaviors by incorporating virtual physics and artificial potential fields—mainly in the case of semi-automatic design. For example, Pinciroli et al. (2008b) produced control software for swarms of satellites that self-organize

in lattices. In a follow-up study, the authors used artificial evolution to fine-tune the parameters of the control software (Pinciroli et al. 2008a). Further examples of this approach are presented by Hettiarachchi and Spears (2005, 2006, 2009).

In AutoMoDe, Francesca et al. (2015) conducted experiments with `Chocolate` in three missions in which robots must operate under spatial distribution constrains. In these missions, the swarm was considered to perform well if the robots (i) uniformly cover the perimeter and surface of two target regions, (ii) uniformly cover the arena without entering in regions indicated as forbidden, and (iii) establish a network of robots that maximized the surface covered by the swarm. Although `Chocolate` could in principle design control software for this class of missions, the results were not completely satisfactory. In most cases, the robots addressed the mission without achieving any meaningful spatial organization strategy.

2.4.4 Design of stigmergy-based collective behaviors

Stigmergy is a coordination mechanism in which agents self-organize through indirect local communication that is mediated by the environment (Grassé 1959; Heylighen 2016a). In swarm robotics, a common way to study stigmergy is by implementing pheromone-based behaviors, which mimic the communication strategies of some social insects. For example, ants leave trails of chemical substances—i.e., pheromones—to which their peers can perceive and respond. Analogously, a robot swarm can use pheromone-based stigmergy to coordinate if robots are endowed with the means to emulate the process of releasing and perceiving pheromones in an artificial way (Salman et al. 2020).

Authors commonly use smart environments to implement pheromone-based stigmergy for robots. These include systems of robots or stationary devices, such as RFID tags or other electronic modules, distributed in the environment to store symbolic messages that represent virtual pheromones. Implementations of these virtual pheromones are presented by Payton et al. (2001), Campo et al. (2010), Khaliq et al. (2014), Antoun et al. (2016), and Alfeo et al. (2019). Alternatively, other authors have experimented with devices that display or project virtual pheromones on the ground, which robots can detect with their sensors. Examples of such systems are discussed by Garnier et al. (2013), Na et al. (2019, 2020), and Hunt et al. (2019). In other cases, mixed-reality environments have been used to immerse the robots in a space where they can release and sense virtual pheromones. The use of these mixed-reality systems is explored by Reina et al. (2017), Talamali et al. (2020), and Feola et al. (2023). Alternatively, other studies have focused

on mechanisms for the physical release of artificial pheromones using specialized onboard hardware that can dispense substances such as alcohol or wax, as presented by Russell (1997, 1999) and Fujisawa et al. (2014).

In recent work, we also developed a system for studying the design of pheromone-based stigmergy in swarm robotics. We named this system `Phormica` (Salman et al. 2020). It comprises a hardware module for the e-puck that projects UV light downward, laying an artificial pheromone trail on a ground floor previously coated with photochromic material. The ground exposed to UV light changes color from white to magenta, which fades back to white in about 50 s after the UV light is removed—mimicking the evaporation of chemical pheromones.

Designing stigmergy-based behaviors, whether using artificial pheromones or other forms of environmental modification, is still a challenging problem (Heylighen 2016b). There is an inherent complexity to the design problem as coordination strategies must be achieved relying on modifications to the environment. This makes the design process less intuitive compared to designing collective behaviors based on direct communication strategies—like those described for the coordination via color signals. Currently, no method exists to determine the conditions and quantities in which individuals should release pheromones, or react to them, to achieve a given desired behavior.

In the literature, pheromone-based stigmergy for robot swarms has mostly been achieved through manual design and while being tailored to specific missions in each case—as seen with the systems discussed above. The only exception to manual design is a study by Na et al. (2022) in which deep reinforcement learning was used to develop a collision avoidance behavior based on virtual pheromones. Although limited to simulation, this study showed that control software produced via a learning process could outperform that produced with manual design. The approach relied on a centralized infrastructure to store and distribute global pheromone information to the robots. This design choice ultimately limited its applicability in scenarios where robots are expected to autonomously release and detect artificial pheromones in their physical environment.

2.4.5 Design of robot shepherding behaviors

Research on swarm robotics often overlooks a critical aspect for the future deployment of robot swarms in real-world settings: the swarm must be endowed with the ability to operate in highly dynamic environments populated by other machines, robots, or living beings (King et al. 2023). This is also a topic that has rarely been

studied in the automatic design of robot swarms. Indeed, little research has been devoted so far to investigating how effectively automatic design methods can tackle missions that occur in such populated environments.

In this dissertation, we investigated the design of robot swarms that coordinate with other robots that populate their environment. We framed this problem as a robot shepherding problem (Lien et al. 2004). In robot shepherding, it is assumed that two groups of robots of different kind operate in the same environment—the shepherds and the sheep. Shepherds and sheep influence each other's behavior and constitute a heterogeneous system that must coordinate in a shared environment. Robot shepherding has been investigated mostly within the framework of collective motion. In most cases, the behavior of the sheep is inspired by flocking—see Brambilla et al. (2013). Typically, a designer defines a model that describes the desired behavior for the sheep and shepherds, and then manually produces control software for the two. The designer uses their knowledge and expertise to tailor the behavior of the shepherds to the behavior of the sheep, which is assumed to be known at design time. The purpose of these studies is often restricted to verify whether the models are suitable for creating shepherding behaviors.

Recent studies have shown that certain models offer a viable, principled approach to manually producing specific shepherding behaviors. These studies have shown that it is possible to coordinate the sheep with a single shepherd (Genter and Stone 2014, 2016; Licitra et al. 2019) or with a group of shepherds that act cooperatively (Özdemir et al. 2017; Pierson and Schwager 2018; Hu et al. 2020; Dosieah et al. 2022; Sebastián et al. 2022). Some of these studies also showed the benefits of using optimization processes to fine-tune the parameters of the control software (Özdemir et al. 2017; Dosieah et al. 2022). The limitation of the existing methods is that they have not been conceived to be of general applicability. They have not been tested to operate on a class of missions that involve varied interactions between shepherds and sheep. Indeed, these methods cannot be easily transferred from one problem to another if the behavior of the sheep differs significantly. For example, a model designed for sheep that move away from shepherds cannot be applied to scenarios where the sheep move toward them.

2.4.6 Design of robot swarms by demonstration

As discussed in Section 2.3, using demonstrations for the automatic design of robot swarms avoids the need to manually define an objective function to drive the design process. Instead of relying on a predefined objective function, this

approach learns a suitable performance measure by operating on demonstrations of the desired behavior. The design process therefore searches for control software that can replicate the demonstrations.

The idea of using demonstrations to program robot swarms builds on recent advances in *imitation learning*. Imitation learning is a type of reinforcement learning where the reward function is not explicitly defined (Osa et al. 2018). Instead, the learning process relies on demonstrations of the desired behavior, with agents attempting to learn a policy that replicates or closely approximates the demonstrated behavior. The most common imitation learning methods primarily focus on learning trajectories for the movement of individual robots, where a robot's trajectory is typically represented by the states of its joints and links. The learning algorithm then associates perceived stimuli with actions to achieve these states—see, for example, Pérez-Dattari et al. (2024). However, applying these methods to program robot swarms presents significant challenges. There is no standardized numerical or feature-based representation to describe the behavior of multiple interacting robots, as exists for individual trajectories. Additionally, without mission-specific expert knowledge, there is no established way to quantitatively measure the similarity between two collective behaviors. Therefore, common imitation learning methods must be revisited and adapted to the particularities of the problem of designing robot swarms. For a comprehensive review of imitation learning approaches, we recommend the work of Hussein et al. (2017). For an overview of its application to swarm robotics, see the review by Kuckling (2023b).

The the adoption of imitation learning in the automatic design of robot swarms is relatively new and little research has been devoted to investigate it. In this context, Li et al. (2016) proposed *Turing Learning*. This method uses a discriminator to drive the design process by assessing how closely the swarm can match a set of demonstrated trajectories. Similarly, Alharthi et al. (2022) used video recordings of Kilobots to learn a behavior tree capable of producing collective behaviors similar to those demonstrated in the videos. Inverse reinforcement learning was first applied to robot swarms by Šošić et al. (2017). They presented a solution to learning a local reward function that explains and reproduces the desired global behavior of a swarm. In their study, the authors designed collective behaviors for a swarm of particles that perform collective motion and alignment.

As introduced in Section 2.3, we developed Demo-`Cho`: an AutoMoDe method that can design robot swarms on the basis of user demonstrations (Gharbi et al. 2023). Demo-`Cho` combines `Chocolate` with apprenticeship learning (Abbeel and Ng 2004), an implementation of the inverse reinforcement learning approach. Our

study with Demo-`Cho` focused on investigating the general applicability of inverse reinforcement learning in the automatic design of robot swarms. We therefore applied Demo-`Cho` to various missions previously studied with `Chocolate`. As in Šošic's work, we conducted experiments with a swarm that must achieve a desired spatial organization.

3. AutoMoDe-TuttiFrutti

The diversity of the missions that AutoMoDe can address has been so far constrained by the limited capabilities of the e-puck considered in the conception of `Vanilla` and `Chocolate`. We advanced the state of the art of the AutoMoDe family by developing `TuttiFrutti`: a method that generates control software for swarms of e-pucks that can display and perceive color signals using their RGB LEDs and omnidirectional camera. In this chapter, we show that these new capabilities enabled us to explore the automatic design of robot swarms capable of performing missions that involve communication, navigation, and the reaction to events.

Francesca and Birattari (2016) discussed how the capabilities of robot platforms can limit the variety of collective behaviors that automatic design methods can produce. As detailed in Chapter 2, most AutoMoDe methods build on `Chocolate` and have been limited to the class of missions defined for the e-puck with reference model RM 1.1—see Table 2.1. In these missions, the robots must position themselves relative to their peers or a few static environmental features, such as black or white floor patches, and rely on a single global reference for navigation, typically a strong ambient light source. Hasselmann et al. (2018b) showed that by expanding the capabilities of the e-puck in RM 2, they achieved a wider range of collective behaviors using `Gianduja`. Indeed, by allowing robots to broadcast and respond to messages, `Gianduja` could perform complex missions that `Chocolate` could not address—e.g., those requiring event-handling collective behaviors.

Our approach to developing `TuttiFrutti` borrows from the reasoning of Hasselmann et al. (2018b). We conceive `TuttiFrutti` as an AutoMoDe method that targets e-puck robots with the added capability of displaying and perceiving colors. `TuttiFrutti` therefore specializes in designing collective behaviors for robot swarms that use color signals to coordinate. By developing `TuttiFrutti`, we overcome the limitations of previous AutoMoDe methods from a two-fold perspective. First, e-pucks that can display and perceive colors allow for the design of swarms where individuals communicate through color signals. Second, these more capable swarms can perform missions in complex and time-varying environments. With the research conducted with `TuttiFrutti`, we significantly enlarged the variety of collective behaviors that can be designed with AutoMoDe—as will also be shown in Chapters 4 and 5.

`TuttiFrutti` can address missions that previous instances of AutoMoDe cannot not address. We used these missions to investigate various aspects of the design problem: (i) whether `TuttiFrutti` is capable of determining if a color displayed in the environment provides useful information to perform a mission; (ii) whether it can produce collective behaviors that demonstrate color-based communication between robots; (iii) whether the extended capabilities of the e-puck increase the difficulty of automatically designing control software for the robot swarm; (iv) and how these new resources can be used to create more complex missions.

The following sections provide a detailed description of `TuttiFrutti`, its characteristic elements, and the experiments we conducted with the method.

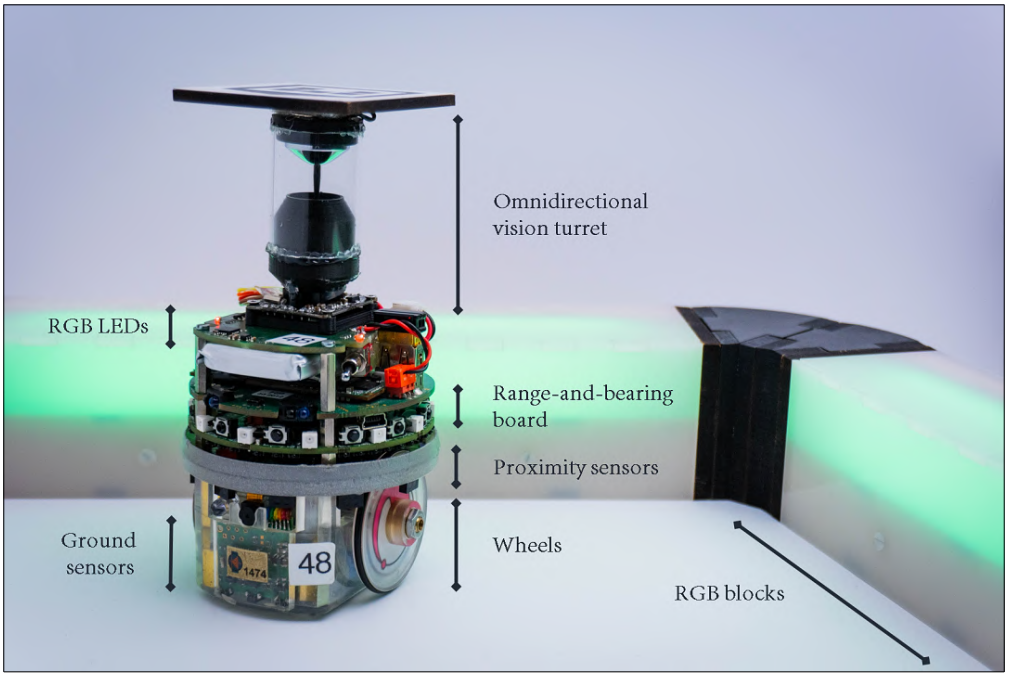


Figure 3.1: Extended version of the e-puck. The picture indicates the set of sensors and actuators defined by RM 3, as introduced with `TuttiFrutti`. Alongside, we also show the RGB blocks that we used in our experiments with `TuttiFrutti`.

3.1 Robot platform

`TuttiFrutti` produces control software for an extended version of the e-puck (Mondada et al. 2009; Garattoni et al. 2015)—see Figure 3.1. Like Francesca et al. (2014b) in `Vanilla`, we also adopt the concept of reference model to formally characterize the e-puck for which `TuttiFrutti` can produce control software. We consider a model of the e-puck endowed with a set of sensors and actuators defined by the reference model RM 3—see Table 3.1.

In the reference model RM 3, the e-puck operates with an Overo Gumstix extension board that runs a linux-based operating system. The e-puck can detect nearby obstacles by its eight proximity sensors ($prox_i$) distributed around its chassis. Three infrared ground sensors (gnd_j) allow the e-puck to differentiate between black, gray and white floor. A range-and-bearing board (Gutiérrez et al. 2009) allows the e-puck to estimate the number of neighboring peers (n) in a 0.5 m range. A vector (V_n) represents the relative aggregate position of the neighboring e-pucks. The omnidirectional vision turret allows the e-puck to perceive red, blue, green, cyan, magenta, and yellow lights (cam_c) in a 360° field of view and within a 0.5 m range. For each color perceived, a unit vector (V_c) represents the relative aggregate

Table 3.1: The control interface for the e-puck according to the reference model RM 3. Robots can perceive: red (R); green (G); blue (B); cyan (C); magenta (M); and yellow (Y). Robots can display no color ($\varnothing$); cyan (C); magenta (M); and yellow (Y). V_c is calculated likewise V_n—for each perceived color, the positions of color signals are aggregated into a unique vector.

Input	**Value**	**Description**
$prox_{i\in\{1,\dots,8\}}$	$[0,1]$	reading of proximity sensor i
$gnd_{j\in\{1,\dots,3\}}$	$\{black, gray, white\}$	reading of ground sensor j
n	$\{0,\dots,20\}$	number of neighboring robots detected
V_n	$([0.5,20];[0,2]\,\pi\,\mathrm{rad})$	their relative aggregate position
$cam_{c\in\{R,G,B,C,M,Y\}}$	$\{yes, no\}$	colors perceived
$V_{c\in\{R,G,B,C,M,Y\}}$	$(1.0;[0,2]\,\pi\,\mathrm{rad})$	their relative aggregate direction
Output	**Value**	**Description**
$v_{k\in\{l,r\}}$	$[-0.12,0.12]\,\mathrm{m\,s^{-1}}$	target linear wheel velocity
$LEDs$	$\{\varnothing, C, M, Y\}$	color displayed by the LEDs

Period of the control cycle: 0.1 s.

position of robots/objects that display the color. The control software of the robot can set the velocity of each wheel (v_k) between -0.12 and $0.12\,\mathrm{m\,s^{-1}}$. The control software can also set the three RGB LEDs placed on the top of the e-puck to display cyan, magenta or yellow.

RM 3 is the first reference model in the AutoMoDe family to include the omnidirectional vision turret and RGB LEDs of the e-puck. An important difference between `TuttiFrutti` and other instances of AutoMoDe is that in RM 3 we removed the capability of the e-puck for estimating the position of ambient light sources. Although present in the reference models RM 1.1 and RM 2 (Hasselmann et al. 2018a), this capability is incompatible with the omnidirectional vision turret and RGB LEDs we added in RM 3. The ambient light sources in RM 1.1 and RM 2 are in practice strong halogen lights, which unavoidably overexpose the camera of the e-puck and make it difficult for the vision system to detect the RGB LEDs of other robots.

3.2 Modular control architecture

`TuttiFrutti` assembles predefined software modules into probabilistic finite-state machines—like `Vanilla`, `Chocolate`, and other AutoMoDe methods. The set of modules in `TuttiFrutti` comprise six low-level behaviors—the actions that a robot can take, and seven transition conditions—the events that trigger the transition between low-level behaviors. These modules adapt and extend the

Table 3.2: TuttiFrutti's software modules. The modules are defined on the basis of reference model RM 3, see Table 3.1.

Low-level behavior*	**Parameter**	**Description**
EXPLORATION	$\{\tau, \gamma\}$	movement by random walk
STOP	$\{\gamma\}$	standstill state
ATTRACTION	$\{\alpha, \gamma\}$	physics-based attraction to neighboring robots
REPULSION	$\{\alpha, \gamma\}$	physics-based repulsion from neighboring robots
COLOR-FOLLOWING	$\{\delta, \gamma\}$	steady movement towards robots/objects of color δ
COLOR-ELUSION	$\{\delta, \gamma\}$	steady movement away from robots/objects of color δ
Transition condition	**Parameter**	**Description**
BLACK-FLOOR	$\{\beta\}$	black floor beneath the robot
GRAY-FLOOR	$\{\beta\}$	gray floor beneath the robot
WHITE-FLOOR	$\{\beta\}$	white floor beneath the robot
NEIGHBOR-COUNT	$\{\xi, \eta\}$	number of neighboring robots greater than ξ
INV-NEIGHBOR-COUNT	$\{\xi, \eta\}$	number of neighboring robots less than ξ
FIXED-PROBABILITY	$\{\beta\}$	transition with a fixed probability
COLOR-DETECTION	$\{\delta, \beta\}$	robots/objects of color δ perceived

* All low-level behaviors display a color $\gamma \in \{\varnothing, C, M, Y\}$ alongside the action described.

modules originally conceived for Vanilla—see Table 2.2. TuttiFrutti's modules have been designed to operate with RM 3 and provide the e-puck different ways of interacting with robots and objects that display colors. Table 3.2 lists TuttiFrutti's low-level behaviors and transition conditions.

3.2.1 Low-level behaviors

In EXPLORATION, the robot moves straight until it detects an obstacle in front ($prox_i$). It then rotates for a number of control cycles defined by the integer parameter τ, in a range of $\tau \in \{0, \ldots, 100\}$. STOP sets the robot to a standstill behavior. In ATTRACTION and REPULSION, the robot moves closer (V_d) or further from ($-V_d$) neighboring peers, respectively. In both cases, the velocity of the robot is a function of the number of robots detected (n) and the parameter $\alpha \in [0, 5]$. If the robot does not detect other robots, it moves straight. COLOR-FOLLOWING and COLOR-ELUSION move the robot with constant velocity towards (V_c) or away ($-V_c$) from robots or objects displaying specific colors (cam_c). The parameter $\delta \in \{R, G, B, C, M, Y\}$ determines the color to which the robots react. Robots can display the colors $\delta \in \{C, M, Y\}$, and other objects that might populate the environment can display the colors $\delta \in \{R, G, B\}$. If the robot does not perceive the color determined by δ, it moves straight. ATTRACTION, REPULSION, COLOR-FOLLOWING and COLOR-ELUSION incorporate physics-based obstacle avoidance (Borenstein

and Koren 1989). In all the low-level behaviors, the parameter $\gamma \in \{\varnothing, C, M, Y\}$ determines the color displayed by the RGB LEDs of the robot. The parameters τ, α, δ, and γ are tuned by the automatic design process.

3.2.2 Transition conditions

BLACK-FLOOR, GRAY-FLOOR and WHITE-FLOOR trigger a transition when the robot steps on a region of the floor (gnd_j) that is, respectively, black, gray or white. The parameter $\beta \in [0, 1]$ determines the probability of transitioning. NEIGHBOR-COUNT and INV-NEIGHBOR-COUNT are transition conditions that consider the number of neighboring robots (n). NEIGHBOR-COUNT triggers a transition with a probability $z(n) \in [0, 1]$, with $z(n) = \frac{1}{1+e^{\eta(\xi-n)}}$. Conversely, INV-NEIGHBOR-COUNT triggers a transition with a probability of $1 - z(n)$. The parameter $\xi \in [0, 20]$ determines the inflection point of the probability function $z(n)$, and the parameter $\eta \in \{0, \ldots, 10\}$ determines its steepness. FIXED-PROBABILITY triggers a transition with a fixed probability determined by $\beta \in [0, 1]$—no further condition is considered. COLOR-DETECTION is based on the colors perceived by the robot (cam_c). The parameter $\delta \in \{R, G, B, C, M, Y\}$ defines the color that triggers a transition with probability $\beta \in [0, 1]$. Robots in the swarm can display the colors $\delta \in \{C, M, Y\}$, and other objects that might populate the environment can display the colors $\delta \in \{R, G, B\}$. The parameters β, ξ, η, and δ are tuned by the automatic design process.

EXPLORATION, STOP, ATTRACTION and REPULSION are modified versions of `Vanilla`'s original low-level behaviors. In `TuttiFrutti`, we extended these modules by adding the ability to control the color displayed by the LEDs. All transition conditions, with the exception of COLOR-DETECTION, are implementations of `Vanilla`'s original modules. COLOR-FOLLOWING, COLOR-ELUSION, and COLOR-DETECTION are modules that we introduced for the first time with `TuttiFrutti`.

Figure 3.2 shows a simplified illustration of `TuttiFrutti`'s software modules assembled into a probabilistic finite-state machine and the resulting behavior on an e-puck.

3.3 Automatic design process

`TuttiFrutti` produces control software using the automatic design process introduced in `Chocolate`—see Chapter 2. It conducts an optimization process to find a combination of modules and parameters that, once deployed to the robots, maximize the performance of the swarm according to a mission-specific perfor-

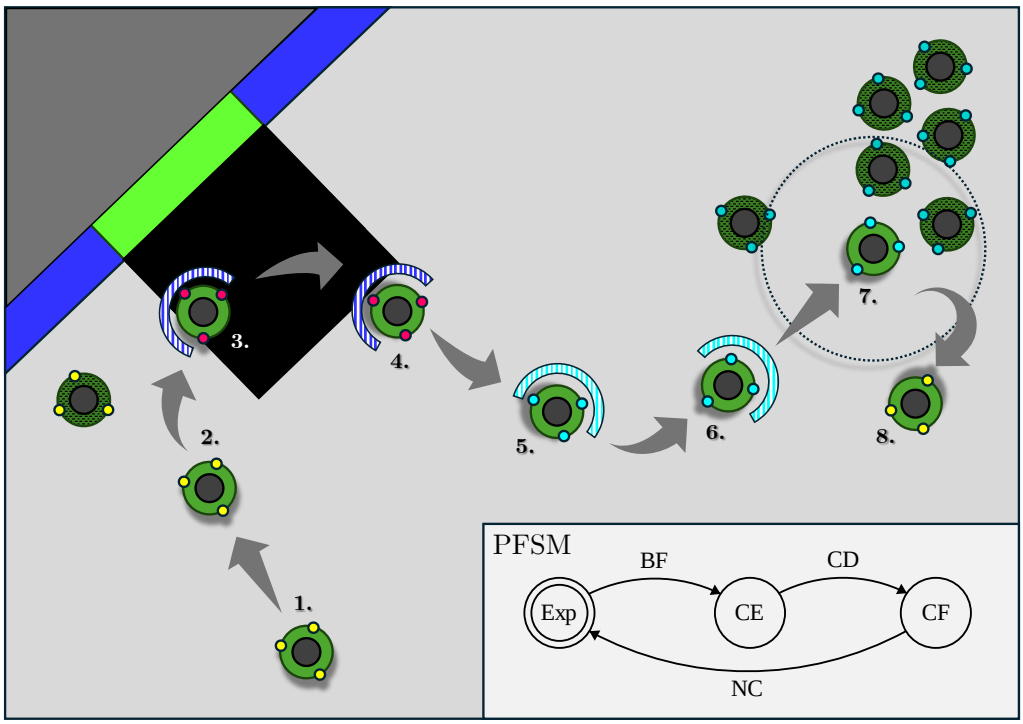


Figure 3.2: Simplified illustration of `TuttiFrutti`'s software modules assembled into a probabilistic finite-state machine (PFSM) and the resulting behavior on an e-puck. 1.) The finite-state machine starts with the behavior EXPLORATION (Exp), which in this case sets the e-puck's LEDs to display yellow while the robot moves randomly in the arena. 2.) The e-puck detects a robot on its left and turns right to avoid a collision. 3.) When the e-puck detects a region with a black floor, the transition BLACK-FLOOR (BF) is activated, and the e-puck switches to the behavior COLOR-ELUSION (CE). 4.) The e-puck executes COLOR-ELUSION (CE), driving the robot away from the blue walls and changing its LEDs to display magenta. 5.) The e-puck detects that other robots are displaying cyan with their LEDs, activating the transition COLOR-DETECTION (CD) and switching to the behavior COLOR-FOLLOWING (CF). 6.) The e-puck executes COLOR-FOLLOWING (CF), moving toward other robots displaying cyan and changing its own LEDs to cyan as well. 7.) The e-puck detects two neighboring robots within its perception range, activating the transition NEIGHBOR-COUNT (NC) and switching back to EXPLORATION (CE). 8.) The finite-state machine continues to operate until the mission ends. For more information on the modules and their parameters, see Table 3.2.

mance measure. In TuttiFrutti, the probabilistic finite-state machine is restricted to a maximum of four states—the low-level behaviors—and four outgoing edges per state—the transition conditions. Transitions always occur between different states, and self-transitions are not permitted. The modules and their parameters are selected off-line through an optimization process conducted with Iterated F-race (López-Ibáñez et al. 2016). Iterated F-race explores the design space to find control software configurations suited for the mission at hand. The performance of the configurations is estimated through simulations performed in ARGoS3 (Pinciroli et al. 2012), version beta 48, together with the argos3-epuck library (Garattoni et al. 2015). The duration of the optimization process is determined by a predefined simulations budget. Once the budget is exhausted, the design process ends and TuttiFrutti returns the best configuration found. This configuration is then uploaded to the physical robots and evaluated in the target environment.

3.4 Experimental setup

We investigate the ability of TuttiFrutti to address a class of missions in which the colors displayed by objects in the environment provide relevant information to the robots. To evaluate this design problem, we conduct experiments in simulation and with physical robots in three missions. The baseline for these experiments is an implementation of the neuroevolutionary approach.

3.4.1 Experimental environment

We developed a programmable RGB environment named MoCA to leverage the new capabilities introduced with TuttiFrutti. This system allowed us to explore more complex environments and missions than those previously studied with AutoMoDe. MoCA (Garzón Ramos et al. 2022) is an open source, modular platform that provides tools for creating, simulating, and deploying physical scenarios to experiment with robots that react to colored objects. At its core, MoCA consists of interconnected blocks equipped with RGB LEDs, which are controlled via a computer. The RGB blocks are 0.25 m in length and match the height of the e-puck. MoCA's blocks can display color patterns that can be reconfigured at run-time. The modularity of MoCA facilitates the deployment of experimental arenas of various sizes and shapes.

In our experiments, we use MoCA to create experimental scenarios by arranging the RGB blocks into polygonal structures that serve as robot arenas. In addition to the physical system, we developed a simulated version that is used in ARGoS3

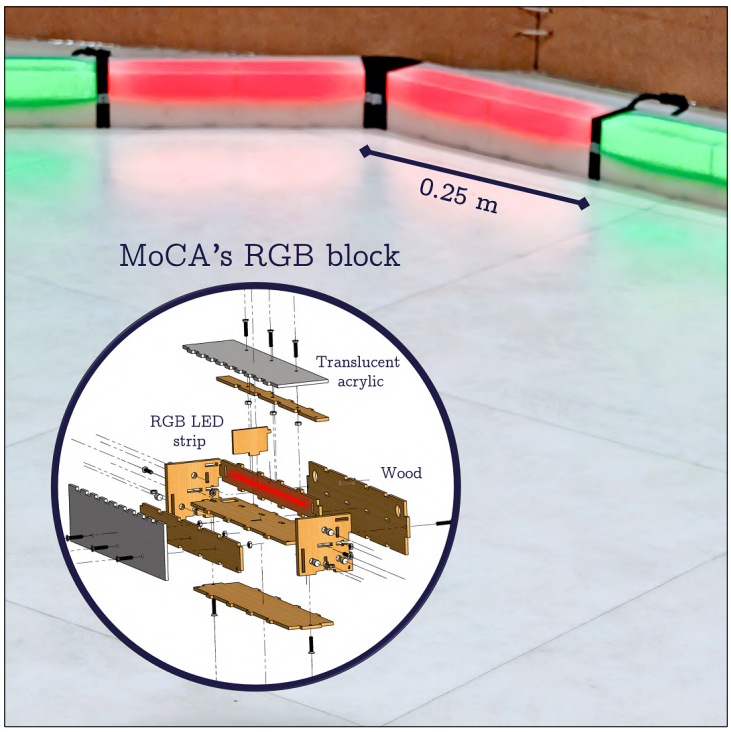


Figure 3.3: MoCA's RGB blocks. The picture shows the modular RGB blocks connected in the corner of an experimental arena for the e-pucks. Alongside, we also also illustrate the inner structure of a block.

during the automatic design process. We introduced MoCA along with TuttiFrutti, and its functionalities turned instrumental to support our further research. As it will be shown in Chapters 4 and 5, we used MoCA to investigate new problems in the automatic design of robot swarms. Figure 3.3 shows MoCA's RGB blocks. For a detailed description of MoCA's hardware and software we refer the reader to the technical documentation of the system—see Garzón Ramos et al. (2022).

3.4.2 Missions

We conduct experiments with twenty e-pucks that must perform missions in which environmental signals, expressed as colors, provide relevant information to the swarm. We experiment with TuttiFrutti in three missions: STOP, AGGREGATION, and FORAGING. STOP and AGGREGATION are adaptations we make from equivalent missions proposed by Hasselmann et al. (2018b) to study Gianduja. FORAGING is an abstraction of a foraging task, in a *best-of-n* fashion—similar to the experiments described by Valentini et al. (2015, 2017). In each case, the performance of the swarm is evaluated according to a mission-specific objective function. We select

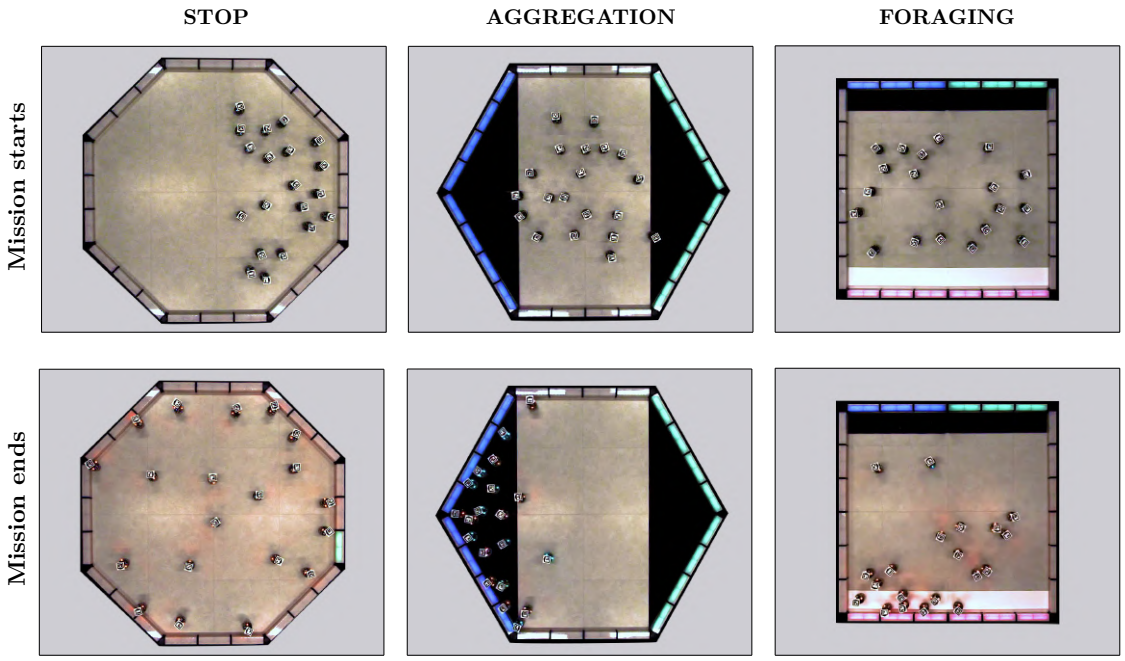


Figure 3.4: Experimental arenas for `TuttiFrutti`'s experiments. The figure shows the arenas used for the three missions in the study. The top row displays the initial positions of the robots at the start of an experiment. The bottom row shows their positions at the end, after performing the mission using control software generated by `TuttiFrutti`. The RGB blocks in the arena are configured on a per-mission basis.

these missions because we conjecture that, to successfully perform them, the robots need to identify, process, and/or transmit color-based information.

The time available to the robots to perform a mission is always $T = 120\,\text{s}$. We use `MoCA` to display colors on the walls of the arena, which are defined on a per-mission basis. Each RGB block can display the colors red, green, and blue $\{R, G, B\}$. In the context of these missions, when we refer to walls of a given color, we imply that the RGB blocks arranged in the wall display the named color—for example, *"the green wall"* stands for a wall in which the RGB blocks composing it display the color green. Figure 3.4 shows the arenas for the three missions. In the following, we provide the specifications for each of them.

STOP

The robots must move until one of the walls that surrounds the arena emits a stop signal by turning green. Once the wall turns green, all robots in the swarm must stop moving as soon as possible. The swarm operates in an octagonal arena of $2.75\,\text{m}^2$ and gray floor. The wall that emits the stop signal is selected randomly. At the beginning of each run, the robots are positioned in the right side of the

arena. Figure 3.4 (left) shows the arena for STOP.

The score of the swarm is determined by time during which the robots do not perform the intended behavior, before and after the stop signal:

$$f_{\text{ST.T}} = \sum_{t=1}^{\bar{t}} \sum_{i=1}^{N} \bar{I}_i(t) + \sum_{t=\bar{t}+1}^{T} \sum_{i=1}^{N} I_i(t), \tag{3.1}$$

which must be minimized. N and T represent the number of robots and the duration of the mission, respectively. The time at which the stop signal is displayed is represented by $\bar{t}$. The value of $\bar{t}$ is uniformly sampled between $(40, 60)$ s. The indicators $I_i(t)$ and $\bar{I}_i(t)$ are defined as:

$$I_i(t) = \begin{cases} 1, \text{ if robot } i \text{ is moving at time } t; \\ 0, \text{ otherwise;} \end{cases} \qquad \bar{I}_i(t) = 1 - I_i(t).$$

We selected this mission because it challenges `TuttiFrutti` to design collective behaviors with event-handling capabilities, which can help the swarm react when the stop signal appears.

AGGREGATION

The robots must aggregate in the left black region of the arena as soon as possible. The swarm operates in a hexagonal arena of about $2.60\,\text{m}^2$ and gray floor. Triangular black regions of about $0.45\,\text{m}^2$ are located at the left and right sides of the arena. The walls lining the left black region are blue and those lining the right black region are green—the colors do not change during the mission. Each black region is characterized by the color of the walls that lines it. That is, the *blue zone* refers to the black region lined by blue walls and the *green zone* refers to the black region lined by green walls. At the beginning of each run, the robots are randomly positioned in the center of the arena—between the black regions. Figure 3.4 (center) shows the arena for AGGREGATION.

The score of the swarm is determined by the time that the robots spend outside of the blue zone:

$$f_{\text{AG.T}} = \sum_{t=1}^{T} \sum_{i=1}^{N} I_i(t), \tag{3.2}$$

which must be minimized. N and T represent the number of robots and the

duration of the mission, respectively. The indicator $I_i(t)$ is defined as:

$$I_i(t) = \begin{cases} 1, \text{ if robot } i \text{ is not in the aggregation area at time } t; \\ 0, \text{ otherwise.} \end{cases}$$

We selected this mission because it challenges `TuttiFrutti` to design collective behaviors in which the swarm uses the blue walls as a reference to identify the aggregation zone and navigate toward it.

FORAGING

The robots must select and forage from the most profitable of two sources of items. The swarm operates in a squared arena of $2.25\,\mathrm{m}^2$ and gray floor. A rectangular white region of about $0.23\,\mathrm{m}^2$ is located at the bottom of the arena and represents the nest of the swarm. A rectangular black region of $0.23\,\mathrm{m}^2$ is located at the top of the arena and represents the two sources of items—the sources are separated by a short wall segment that does not display any color. This wall segment divides the black region in half. We account that an item is transported and successfully delivered when a robot travels from any of the sources to the nest. The walls lining the nest are red, the walls lining the left source are blue, and the walls lining the right source are green—the colors do not change during the mission. We consider two types of source of items: a *blue source*—the black region lined by blue walls; and a *green source*—the black region lined by green walls. At the beginning of each run, the robots are randomly positioned in the center of the arena—between the white and black areas. Figure 3.4 (right) shows the arena for FORAGING.

The score of the swarm is the aggregate profit of the total of items collected from the two sources:

$$f_{\text{FR.T}} = (\kappa)I_b + (-\kappa)I_g; \tag{3.3}$$
$$\kappa = 1,$$

which must be maximized. I_b and I_g represent the number of items collected from the blue and green sources, respectively. We added the factor κ to balance the profit of the items available in each source. We considered $\kappa = 1$. Items from the blue source account for a profit of $+1$ and items from the green source account for a penalization of -1. We selected this mission because it challenges `TuttiFrutti` to design collective behaviors in which the swarm uses the color displayed by the walls as a reference to navigate the environment. Robots can use the blue walls to navigate toward the blue source, the green walls to avoid the green source, and the

red walls to navigate toward the nest.

3.4.3 Baseline method

No standard automatic design method exists to address the class of missions we consider in this study. As described in Chapter 2, the few related work describes experiments conducted with neuroevolutionary methods applied in a mission-specific context. Therefore, we consider that a typical implementation of the neuroevolutionary approach is a suitable baseline to appraise the performance of `TuttiFrutti`. We introduce here `EvoColor`: a neuroevolutionary method for the automatic design of swarms of e-pucks that can display and perceive colors.

`EvoColor`

`EvoColor` is an adaptation of `EvoStick` (Francesca et al. 2014b), the neuroevolutionary method recurrently used as a yardstick in AutoMoDe studies—see Chapter 2. `EvoColor` produces control software for swarms of e-pucks that operate with the reference model RM 3—see Table 3.1. The control software has the form of a fully connected feed-forward artificial neural network with 41 input nodes (*in*), 8 output nodes (*out*) and no hidden layers. In this topology, the input and output nodes are directly connected by synaptic connections (*conn*) with weights (ω) in a range of $[-5, 5]$. The activation of each output node is determined by the weighted sum of all input nodes, filtered through a standard logistic function. `EvoColor` selects appropriate synaptic weights using an evolutionary process based on elitism and mutation. As in `TuttiFrutti`, the evolutionary process is conducted through simulations performed in ARGoS3, version beta 48, together with the argos3-epuck library. The evolution ends when a predefined simulations budget is exhausted. Table 3.3 summarizes the topology of the neural network and the parameters used in the evolutionary process.

The readings of the proximity (*prox*) and ground (*gnd*) sensors are passed directly to the network. Information about the number of neighboring peers (n) is provided through the function $z'(n) \in [0, 1]$, with $z'(n) = 1 - \frac{2}{1+e^{(n)}}$. The vector V_n and each vector in $V_{c \in \{R,G,B,C,M,Y\}}$ are translated into scalar projections onto four unit vectors that point at 45°, 135°, 225°, and 315° with respect to the front of the robot. Then, each projection is passed to the network through an independent input node. The last input of the network corresponds to a bias node. Four output nodes encode tuples (v') of negative and positive components of the velocity of the wheels. Each tuple is obtained from two independent output nodes and is defined

Table 3.3: Neural network topology and parameters of the evolutionary process in EvoColor. The neural network operates according to RM 3, see Table 3.1.

Architecture	
Fully-connected feed-forward neural network without hidden layers	
Input node	**Description**
$in_{a\in\{1,\ldots,8\}}$	readings of proximity sensors $prox_{i\in\{1,\ldots,8\}}$
$in_{a\in\{9,\ldots,11\}}$	readings of ground sensors $gnd_{j\in\{1,\ldots,3\}}$
$in_{a\in\{12\}}$	value of the density function $z'(n)$
$in_{a\in\{13,\ldots,16\}}$	scalar projections of V_n
$in_{a\in\{17,\ldots,40\}}$	scalar projections of $V_{c\in\{R,G,B,C,M,Y\}}$
$in_{a\in\{41\}}$	bias input
Output node	**Description**
$out_{b\in\{1,\ldots,4\}}$	tuples v' to map each velocity in the set $v_{k\in\{l,r\}}$
$out_{b\in\{5,\ldots,8\}}$	activation of each color in the set $\{\varnothing, C, M, Y\}$
Connection	**Description**
$conn_{s\in\{1,\ldots,328\}}$	synaptic connections with weights $\omega\in[-5,5]$
Evolution parameters	
Number of generations *	—
Population size	100
Elite individuals	20
Mutated individuals	80
Evaluations per individual	10
Post-evaluation per individual **	100

* The number of generations is computed according to the budget of simulations.
** The population obtained in the last generation is post-evaluated to select the best individual.

as $v' = ([-12, 0], [0, 12])$. The velocity of a wheel (v) is calculated as the sum of the two elements in a tuple (v'). Similarly, the color displayed by the RGB LEDs of the robot is selected by comparing the value of the output nodes that correspond to colors in the set $\{\varnothing, C, M, Y\}$. The color displayed corresponds to the maximum value found in all four colors.

EvoColor differs from EvoStick in two aspects: the reference model and how the output of the neural network is mapped to the velocity of the robots. First, EvoColor is based on RM 3 and EvoStick on RM 1.1. In accordance with RM 3, EvoColor does not integrate the capability of the e-pucks to detect ambient light sources, originally integrated in EvoStick. The second difference between EvoColor and EvoStick is how the output of the neural network is mapped to the velocity of the e-pucks. In EvoColor, we introduced a velocity mapping based on tuples to facilitate the evolution of standstill behaviors, as we expected that robots need

them to perform STOP and AGGREGATION.

In `EvoStick`, the control software directly maps a single output node of the neural network into velocity commands ($v = [-12, 12]$) for each wheel ($v_{k\in\{l,r\}}$)—a robot can stand still only if the velocity of the two wheels is set exactly to 0. A standstill behavior is then difficult to achieve since only one pair of values in the output nodes maps exactly to $v_l = 0$ and $v_r = 0$. Moreover, the output nodes cannot maintain a steady value because they are subject to the injection of sensory noise. In `EvoColor`, the control software maps the sum of elements of a tuple (v') to the velocity commands for each wheel $v_{k\in\{l,r\}}$. Each tuple is defined by two output nodes and provides a negative and a positive component to compute the velocity—see also (Quinn et al. 2003). We expected that this mapping could facilitate the evolution of standstill behaviors. First, robots can stand still if the elements of each tuple (v') are any pair of values of equal magnitude—steady values are not required provided that the output nodes that encode the same tuple vary proportionally. Second, the sum of the positive and negative components can cancel out the sensory noise injected in the output nodes that encode a tuple—given a proper tuning of the synaptic weights. If one compares `EvoColor` with `EvoStick`, the first has more freedom to tune neural networks that lead to standstill behaviors.

3.4.4 Protocol

For each mission, we conduct 10 independent design processes with `TuttiFrutti` and 10 with `EvoColor`. This results in 60 instances of control software—10 per method and mission. The design methods are given a budget of 100 000 simulation runs to produce each instance of control software. We evaluate the effectiveness of the methods by testing each instance of control software once in simulation and once with physical robots. The performance of the swarm is computed in both simulation and reality using ARGoS3. In the simulations, ARGoS3 computes the performance directly from the simulation data. In the experiments with physical robots, ARGoS3 is provided with the position of the real robots using a tracking system (Stranieri et al. 2013).

Statistics

The performance of the instances of control software obtained in the experiments is presented with box-plots. For each method, we report the performance obtained in simulation (thin boxes) and with physical robots (thick ones). In all cases, comparative statements are supported with an exact binomial test, at 95 % con-

fidence (Conover 1999). Statements like "*A* performs *significantly* better/worse than *B*" imply that the comparison is supported by an exact binomial test, at 95 % confidence. In addition, we estimate the overall performance of `TuttiFrutti` with respect to `EvoColor`. To this purpose, we aggregate the results by comparing the performance of the two design methods across each mission. In the context of the overall performance of the design methods, any statement like "*A* performs *significantly* better/worse than *B*" also implies that the comparison is supported by an exact binomial test, at 95 % confidence.

3.5 Results

We discuss first the behavior and performance of the swarms on a per-mission basis. Then, we elaborate on the aggregate performance across the three missions. Demonstration videos are provided in the Supplementary Videos of the dissertation (Garzón Ramos 2025). In the context of these results, references to colored robots imply that the robots display the named color—for example, *"cyan robots"* stands for robots that display the color cyan.

3.5.1 Per-mission results

We present a quantitative and qualitative analysis of the results obtained with `TuttiFrutti` and `EvoColor` for each mission. Performance plots are shown in Figure 3.5.

STOP

Figure 3.5 (left) shows the performance of `TuttiFrutti` and `EvoColor` in STOP. In this mission, `TuttiFrutti` performed significantly better than `EvoColor`.

We visually inspected the behavior of the swarm. In this mission, `TuttiFrutti` produced control software that effectively uses the robots' capabilities to display and perceive colors. The swarm disperses and homogeneously covers the arena, with the aim of rapidly detecting the stop signal. If a robot detects the stop signal, it stands still and disseminates the information by emitting a signal of an arbitrary color. When other robots perceive the signal emitted by their peer, they also transition to a standstill behavior and relay the signal. The process continues until all robots in the swarm are standing still. We consider that this behavior shows the potential of `TuttiFrutti` to produce collective behaviors that require

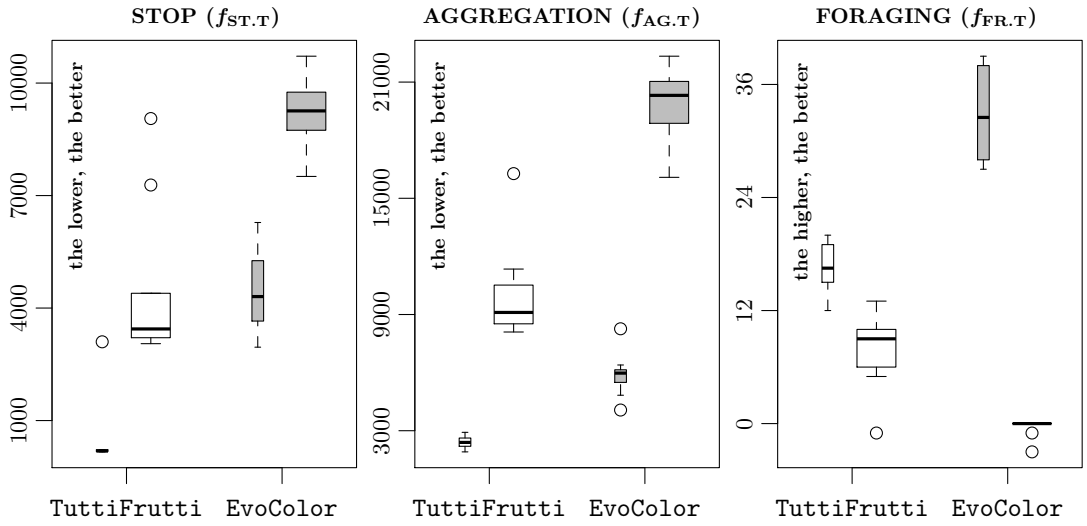


Figure 3.5: Performance obtained in the missions studied with `TuttiFrutti`. Results per design method are presented with grayscale box-plots, `TuttiFrutti` (□), `EvoColor` (■). Thin boxes represent results obtained in simulation and thick boxes the ones obtained with physical robots.

event-handling properties. The swarm collectively transitions from coverage to standstill when the stop signal appears. As we expected, `TuttiFrutti` produced control software on which communication protocols are established by correctly pairing the color of the signals that robots emit and the behavior other robots must adopt when they perceive them—similarly to the results obtained by Hasselmann et al. (2018b) with `Gianduja`.

`EvoColor`, unlike `TuttiFrutti`, designed collective behaviors that do not respond to the stop signal. The swarm adopts a rather simplistic behavior in which robots move until they are stopped by the walls. They then remain in a standstill behavior because they persistently push against the walls—no reaction can be observed in the swarm when the stop signal appears. This behavior was also observed in the experiments with physical robots, and in many cases, robots maintained standstill behaviors by pushing against other robots too.

In the experiments with physical robots, both `TuttiFrutti` and `EvoColor` showed a significant drop in performance with respect to the simulations. However, the difference in median performance between simulations and experiments with physical robots is larger for `EvoColor` than for `TuttiFrutti`. The swarms deployed with control software produced by `TuttiFrutti` show the same collective behavior observed in simulation, although the rapidness of discovering the stop signal and disseminating the information decreased. In the case of `EvoColor`, robots often do

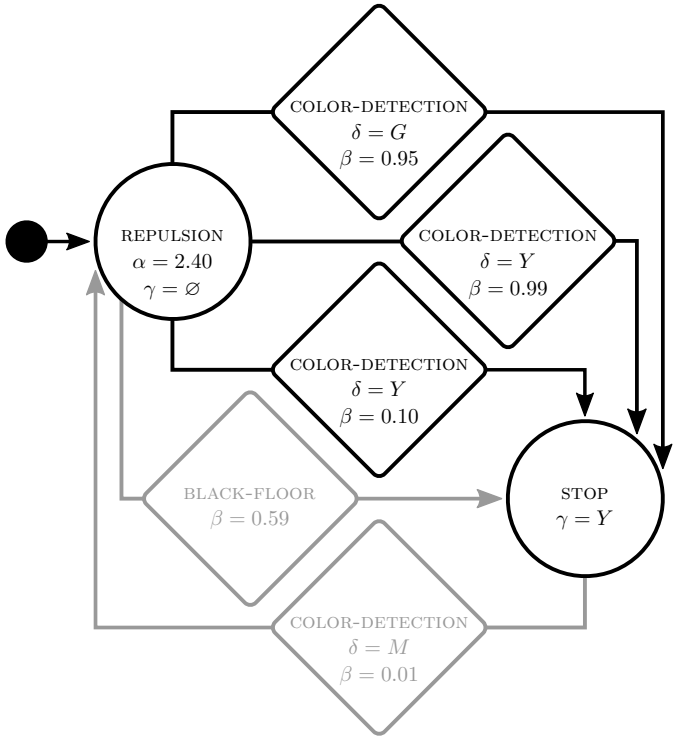


Figure 3.6: Instance of control software produced by `TuttiFrutti` for STOP. The probabilistic finite state machine shows the effective modules in black and non-reachable modules in light gray. Circular modules represent the low-level behaviors and rhomboid modules represent transition conditions.

not reach the walls and push against each other to remain still in place.

Figure 3.6 shows an example of the control software produced by `TuttiFrutti` for STOP. Robots start in REPULSION with no color displayed ($\gamma = \varnothing$). They transition to STOP and turn yellow ($\gamma = Y$) when COLOR-DETECTION is triggered either by a green wall ($\delta = G$) or by yellow robots ($\delta = Y$). In this sense, robots change their behavior when they perceive either the stop signal or the yellow signals that other robots emit.

AGGREGATION

Figure 3.5 (center) shows the performance of `TuttiFrutti` and `EvoColor` in AGGREGATION. In this mission, `TuttiFrutti` performed significantly better than `EvoColor`.

In this mission, `TuttiFrutti` also produced control software that effectively uses the capabilities of robots to display and perceive colors. As we expected, `TuttiFrutti` designed collective behaviors in which robots reach and remain in the blue zone by moving toward blue walls. This behavior is often accompanied by

navigation or communication strategies that increase the efficiency of the swarm. For example, some instances of control software include a repulsion behavior that drives robots away from the green walls—robots reach the blue zone faster by avoiding unnecessary exploration in the green zone. In other instances, robots that step in the blue zone, or perceive the blue walls, emit a signal of an arbitrary color—other robots then follow this signal to reach the blue zone. In this sense, robots communicate and navigate collectively the environment to aggregate faster. We observed that a few instances of control software combine the two strategies.

`EvoColor` designed collective behaviors in which robots use the colors displayed in the arena. Robots explore the arena until they step into one of the black regions—either in the blue or green zone. If robots step in the green zone, they move away from the green walls and reach the blue zone. If robots step in the blue zone, they attempt to stand still. In this sense, robots react and avoid the green walls as a strategy to aggregate in the blue zone.

The control software produced by `TuttiFrutti` and `EvoColor` showed a significant drop in performance when ported to the physical robots. As observed in STOP, the difference in median performance between simulations and experiments with physical robots is larger for `EvoColor` than for `TuttiFrutti`. Robot swarms that use the control software produced by `TuttiFrutti` display the same collective behavior observed in simulation. The decrease in performance occurs because few robots that leave the blue zone do not return as fast as observed in the simulations. The control software produced by `EvoColor` did not port well to the physical robots—i.e., robots appear to be unable to reproduce the behavior observed in the simulation. The robots ramble in the arena and seem to react to the presence of their peers; however, no specific meaningful behavior can be identified by visual inspection.

Figure 3.7 shows an example of the control software produced by `TuttiFrutti` for AGGREGATION. Robots start in COLOR-FOLLOWING displaying yellow ($\delta = Y$) and move towards cyan robots ($\gamma = C$). When they perceive the blue walls ($\delta = B$), COLOR-DETECTION triggers and the robots transition to a second module COLOR-FOLLOWING in which they move towards the blue walls ($\delta = B$) while emitting a cyan signal ($\gamma = C$). By cycling in these behaviors, robots can navigate to the blue zone either by moving towards the blue walls or by following the cyan signals that other robots emit. The transition conditions FIXED-PROBABILITY, GRAY-FLOOR, and NEIGHBOR-COUNT trigger the COLOR-FOLLOWING behavior that allows the robot to return to the aggregation area.

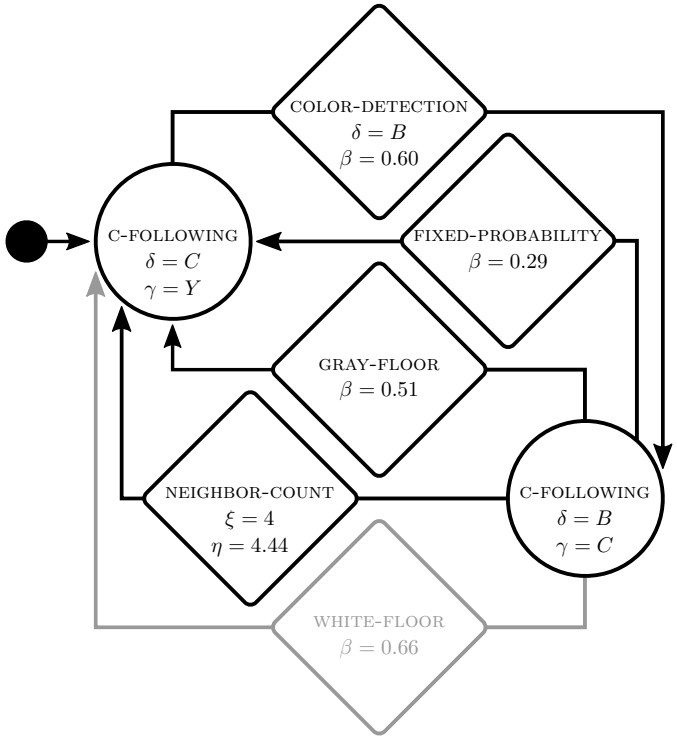


Figure 3.7: Instance of control software produced by `TuttiFrutti` for AGGREGATION. The probabilistic finite state machine shows the effective modules in black and non-reachable modules in light gray. Circular modules represent the low-level behaviors and rhomboid modules represent the transition conditions. Modules labeled as C-FOLLOWING stand for the low-level behavior COLOR-FOLLOWING.

FORAGING

Figure 3.5 (right) shows the performance of `TuttiFrutti` and `EvoColor` in FORAGING. In this mission, `EvoColor` performed significantly better than `TuttiFrutti` in simulation. However, `TuttiFrutti` performed significantly better than `EvoColor` in the experiments with physical robots.

As in the other missions, `TuttiFrutti` produced control software that effectively uses the capabilities the robots have to display and perceive colors. Robots explore the arena and forage only from the profitable source. However, contrary to what we expected, `TuttiFrutti` designed collective behaviors that do not use the three colors displayed in the arena. In fact, robots mostly forage by randomly exploring the arena while moving away from the green wall—in other words, they only avoid to step in the green source. Although the swarm can perform the mission with this behavior, we expected that robots could navigate faster by moving toward the blue and red walls. `TuttiFrutti` produced only a few instances of control software in which robots react to more than one color—see Figure 3.8 for an example. We

conjecture that TuttiFrutti exploited the convex shape of the arena to produce solutions that are effective at minimal complexity. That is, the performance of a swarm in this mission might not improve even if robots react to all three colors.

EvoColor designed collective behaviors in which the swarm does not react to the colors displayed in the arena. Robots forage from the blue source by following the walls of the arena in a clockwise direction. This behavior efficiently drives the robots around the arena and across the blue source. When the robots reach the intersection that divides the blue and green source, they continue moving straight and effectively reach the nest. By cycling in this behavior, the swarm maintains an efficient stream of foraging robots.

TuttiFrutti and EvoColor showed a significant drop in performance in the experiments with physical robots, compared to the performance obtained in the simulations. As in the other two missions, the difference in median performance between simulations and experiments with physical robots is larger for EvoColor than for TuttiFrutti. In the case of TuttiFrutti, we did not observe any difference in the behavior of the swarms with respect to the simulations. Conversely, the collective behaviors designed by EvoColor are affected to the point that the swarm is unable to complete the mission. In the control software produced by EvoColor, the ability of the robots to follow the walls strongly depends on the fine-tuning of the synaptic weights in the neural network—more precisely, it requires a precise mapping between the proximity sensors and wheels of the robots. In the physical robots, the noise of the proximity sensors and wheels likely differs from that of the simulations, and a fine-tuned neural network is less effective. This can possibly explain why the swarm is no longer able to maintain the stream of foraging robots, and on the contrary, robots stick to each other and to the walls.

We observe a *rank inversion* of the performance of the two methods in this mission. As described by Ligot and Birattari (2020), a rank inversion is a phenomenon that manifests when an instance of control software outperforms another in simulation, but it is outperformed by the latter when evaluated in physical robots. In this mission, TuttiFrutti was outperformed by EvoColor in simulation, but it outperformed EvoColor when ported to the physical robots. These results are consistent with those reported by Francesca et al. (2014b), and further discussed by Birattari et al. (2016) and Ligot and Birattari (2020), for comparisons between the modular and the neuroevolutionary approach to the automatic design of robot swarms.

Figure 3.8 shows an example of the control software produced by TuttiFrutti for FORAGING. Robots start in COLOR-FOLLOWING displaying cyan ($\gamma = C$)

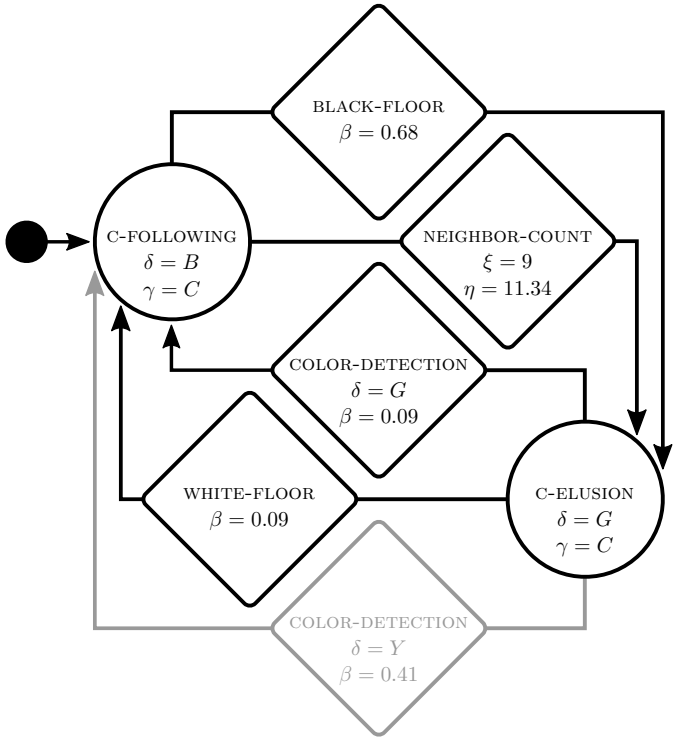


Figure 3.8: Instance of control software produced by `TuttiFrutti` for FORAGING. The probabilistic finite state machine shows the effective modules in black and non-reachable modules in light gray. Circular modules represent the low-level behaviors and rhomboid modules represent the transition conditions. Modules labeled as C-FOLLOWING and C-ELUSION stand for the low-level behaviors COLOR-FOLLOWING and COLOR-ELUSION, respectively.

and move towards the blue wall ($\delta = B$). If a robot steps into one of the two sources, BLACK-FLOOR triggers and the robot transitions to COLOR-ELUSION—it then becomes cyan ($\gamma = C$) and moves away from the green wall ($\delta = G$). When the robot steps into the nest, WHITE-FLOOR triggers and the robot switches back to COLOR-FOLLOWING. By cycling this behavior, robots move back and forth between the blue source and the nest. When robots are in COLOR-ELUSION, COLOR-DETECTION can trigger with a low probability ($\beta = 0.09$) if robots perceive the green wall ($\delta = G$). This transition mitigates the penalty caused by robots that enter the green source. If a robot steps into the green source, it transitions back to COLOR-FOLLOWING and moves towards the blue wall. Finally, the transition condition NEIGHBOR-COUNT can trigger when the robot perceives more than four neighboring robots. We do not find a clear effect of this last transition on the overall behavior of the robots.

3.5.2 Aggregate results

TuttiFrutti and EvoColor obtain similar results when the control software is evaluated with simulations. On the other hand, TuttiFrutti is significantly better than EvoColor when the control software is ported to the physical robots. These results, obtained across the three missions considered, indicate that EvoColor is more affected by the reality gap than TuttiFrutti. The stronger effects of the reality gap in EvoColor are evident in both the performance drop and the difference in behavior between simulation and real-world environments. The control software generated by EvoColor often exhibited diametrically different collective behaviors between simulation and reality. In contrast, the behaviors of the control software generated by TuttiFrutti remained similar across the two environments.

By introducing TuttiFrutti, we also investigated the impact of an extended design space in the optimization process of AutoMoDe. The size of the design space in Vanilla and Chocolate is $O(|B|^4\,|C|^{16})$, as estimated by Kuckling et al. (2018b). B and C represent, respectively, the number of modules in low-level behaviors and transition conditions. Using the same computation, we estimate that the design space in TuttiFrutti is $O(|4B|^4\,|C|^{16})$—that is, 256 times larger than the one searched by Chocolate. Notwithstanding the larger design space, we did not find evidence that TuttiFrutti is affected by the increased number of parameters to tune. In fact, TuttiFrutti produced effective control software for all the missions considered.

3.6 Discussion

TuttiFrutti showed to be effective in selecting, tuning and assembling control software with modules that provide means to the robot to perceive and display color signals. These signals, both produced in the environment or by peer robots, enabled the robots to identify and communicate relevant information on a per-mission basis. The collective behaviors designed by TuttiFrutti showed event-handling and navigation properties.

We also observed that TuttiFrutti can design collective behaviors that exhibit color-based communication between robots. For example, TuttiFrutti designed collective behaviors with communication properties in STOP and AGGREGATION—missions in which communication can influence the performance of the swarm. These collective behaviors were feasible due to the extended capabilities of the e-puck in RM 3—which enabled a larger space of possible control software than

the one considered by `Vanilla` and `Chocolate` with RM 1.1. As the design space of `TuttiFrutti` is larger than that of `Vanilla` and `Chocolate`, one could have expected that the automatic design process would have difficulties in producing meaningful control software. However, we did not find evidence that `TuttiFrutti` suffers from an increased difficulty to design collective behaviors for robot swarms. The reference model RM 3 and the set of modules introduced with `TuttiFrutti` allowed us to conceive STOP and AGGREGATION—variants of missions already studied with AutoMoDe, and FORAGING—a new mission framed within the *best-of-n* problem. By introducing `TuttiFrutti`, we enlarged the variety of collective behaviors and missions that can be studied with AutoMoDe.

The experiments we conducted with `TuttiFrutti` provided the first evidence that AutoMoDe can establish a mission-specific relationship between the colors perceived by the robots and the behavior they must adopt. In Chapter 2, we highlighted literature in which this relationship enabled the manual design of complex collective behaviors (Nouyan et al. 2008; Mathews et al. 2017; Garattoni and Birattari 2018). The more complex collective behaviors described in the literature share functional similarities with those designed by `TuttiFrutti`. For example, robots react to color signals that trigger in the environment and/or use color signals to communicate with each other. The possibility of designing more complex behaviors with the same functional capabilities introduced by `TuttiFrutti` motivated us to further explore applying its original ideas to new design problems. In doing so, we aimed to test AutoMoDe in the realization of robot swarms that can address more complex missions. As we will show in the following Chapters, `TuttiFrutti` served as the basis for developing new methods and experiments that helped bridge the gap between the complexity of missions achieved via manual and automatic design.

4. AutoMoDe-Mandarina

Since the introduction of `Vanilla` and `Chocolate`, AutoMoDe has been constrained to address missions specified by a single design criterion. We overcame this limitation by developing `Mandarina`: an automatic design method that can handle multiple concurrent design criteria in the generation of control software for robot swarms. In this chapter, we show that `Mandarina` enables the realization of robot swarms that perform sequences of missions, where each mission is an independent design criterion to be handled during the design process. We also show that automatically designed robot swarms can use environmental signals to switch their behavior and perform missions sequentially.

Mandarina is a method for the automatic design of robot swarms under concurrent design criteria. Given a mission specification with multiple design criteria and a space of possible instances of control software, Mandarina searches for an instance that maximizes the performance of the swarm in the given mission. More precisely, it conducts a multi-criteria design process to find a neutral compromise solution where all design criteria are satisfied to their best.

The notion of *multi-criteria design* builds upon multi-criteria decision making—that is, the problem of selecting an alternative on the basis of a preference relationship (Fishburn 1970). Multi-criteria decision making is relevant to many different domains (Zopounidis and Doumpos 2017) and is commonly addressed from the perspective of single-objective and multi-objective optimization. For a classical introduction to multi-criteria decision making, see Fishburn (1970); and for a relatively recent survey on evolutionary algorithms for multi-objective optimization, see Emmerich and Deutz (2018). In single-objective approaches, the decision criteria are aggregated (e.g., by means of scalarization) into a single objective function and the space of the alternatives is totally ordered. On the other hand, multi-objective approaches do not aggregate the decision criteria, and as a result, the space of the alternatives is only partially ordered—the notion of optimal solution is replaced by the one of set of non-dominated solutions, the Pareto set. As discussed in Chapter 2, the application of these notions to the automatic design of robot swarms is rare. Trianni and López-Ibáñez (2015) were the first to apply these concepts to neuroevolution within a semi-automatic design process. To the best of our knowledge, Mandarina is the first method to apply them in the fully automatic generation of modular control software.

We developed Mandarina on the basis of TuttiFrutti. They both produce probabilistic finite-state machines for e-pucks that interact with each other and with the environment by displaying and reacting to color signals. Mandarina and TuttiFrutti also share Iterated F-race (Balaprakash et al. 2007; Birattari et al. 2010) as their optimization algorithm, but they differ in how Iterated F-race is configured to drive the design process. Originally designed for single-criterion optimization, Iterated F-race has proven to be effective an effective algorithm for designing robot swarms in single-criterion problems—see Chapter 2. With Mandarina, we show that by making minor adjustments to the way Iterated F-race operates, it can also be suitable for designing robot swarms in problems with concurrent design criteria.

Mandarina exploits some originally unintended properties of Iterated F-race to handle more than one design criterion at a time. More precisely, it exploits its

non-parametric nature and the fact that it operates on ranks. These properties were originally intended for handling problems with large scale variations and complex distributions of the objective functions. We noticed that they turn out to be appropriate and natural for handling concurrent design criteria in a multi-criteria optimization setup. Thanks to these properties, `Mandarina` allows for an automatic design process that does not require aggregating the design criteria into a single objective function. In this chapter, we limit ourselves to demonstrating these properties in a bi-criteria problem, but we expect the methodology to be applicable in design problems with additional criteria.

The design process introduced with `Mandarina` fully aligns with the tenets of the automatic off-line design of robot swarms (Birattari et al. 2019): (i) an automatic method can address a whole class of missions without undergoing any modification; and, (ii) once a mission is specified, no human intervention is allowed for in any phase of the design process. On the other hand, the approaches that rely on manually aggregating design criteria, described in Chapter 2, unavoidably prevent practicing these tenets. Aggregating criteria requires mission-specific human intervention and the injection of domain knowledge. In the typical case, the designer is required to estimate/measure the range of scores spanned by each design criterion. Under this condition, mission-specific experimentation is necessary to understand the nature of the design criteria and to produce and refine an appropriate objective function. A method that requires aggregating design criteria via prior experimentation or expert knowledge belongs mostly to the semi-automatic approach (Birattari et al. 2020). Conversely, the fact that `Mandarina` operates without aggregating criteria makes the method a fully automatic one.

The following sections provide a detailed description of `Mandarina`, its characteristic elements, and the experiments we conducted with the method. As noted earlier, `Mandarina` shares many elements with `TuttiFrutti`. To maintain consistency with how `TuttiFrutti` was introduced in Chapter 3, and how other methods will be presented in Chapter 5, we briefly comment on the elements shared between `Mandarina` and `TuttiFrutti`—i.e., the robot platform and modular control architecture. However, the main focus of this Chapter is on the original research conducted with `Mandarina` and its multi-criteria design process.

4.1 Robot platform

Mandarina produces control software for the same version of the e-puck considered in TuttiFrutti—see Figure 3.1. The functional capabilities of the robot are adopted from TuttiFrutti without any modification and are formally defined by the reference model RM 3—see Table 3.1. RM 3 describes an e-puck endowed with proximity sensors that can detect nearby obstacles; ground sensors that differentiate between gray, black, and white floor; a range-and-bearing board that estimates the number of neighboring peers and their relative aggregate position; an omnidirectional vision turret that can perceive color signals emitted by neighboring objects and/or robots, and estimates their relative aggregate direction; RGB LEDs to emit color signals; and left and right wheels, whose velocity can be set independently. A detailed description of the robot and the reference model RM 3 is provided in Chapter 3.

4.2 Modular control architecture

Mandarina generates control software by fine-tuning and assembling predefined software modules into probabilistic finite-state machines. It operates on the thirteen parametric software modules that were originally conceived for TuttiFrutti—see Table 3.2. The set of modules comprises 6 low-level behaviors—the actions that a robot can execute—and 7 transition conditions—the events that trigger the transition between low-level behaviors. The six low-level behaviors are EXPLORATION, STOP, ATTRACTION, REPULSION, COLOR-DETECTION, and COLOR-ELUSION. The seven transition conditions are BLACK-FLOOR, GRAY-FLOOR, WHITE-FLOOR, NEIGHBOR-COUNT, INV-NEIGHBOR-COUNT, FIXED-PROBABILITY, and COLOR-DETECTION. Mandarina combines low-level behaviors and transition conditions to generate the control software of the robots. The parameters of the software modules are automatically tuned during the design process. As with TuttiFrutti, the modules in Mandarina allow robots to interact with their peers and with objects in the environment by perceiving and displaying color signals—according to the reference model RM 3. A detailed description of the implementation of the software modules is provided in Chapter 3.

4.3 Multi-criteria automatic design process

In Mandarina, the multi-criteria design process is cast into an optimization problem that is addressed with Iterated F-race. We use the implementation of Iterated F-race provided by the irace package (López-Ibáñez et al. 2020) version 2.2. Iterated F-race maximizes the performance of the robot swarm with respect to a set of mission-specific metrics: the objective functions associated to the design criteria. Starting from the specifications of a mission, Iterated F-race selects software modules, fine-tunes their parameters, and combines them into probabilistic finite-state machines—the control software of the robots. Likewise TuttiFrutti, the finite-state machines that Mandarina produces can have up to four states—i.e., low-level behaviors, and up to four outgoing transitions per state—i.e., transition conditions. A transition condition will always originate and end in different states.

During the design process, Iterated F-race searches the design space looking for probabilistic finite-state machines that perform well in the mission at hand. Each finite-state machine is assessed with simulations executed in ARGoS3 (Pinciroli et al. 2012). As with TuttiFrutti, we use ARGoS3 version beta 48, together with the argos3-epuck plugin. Iterated F-race is given a maximum number of simulations to find a candidate solution (i.e., a finite-state machine to address the mission). The maximum number of simulations determines the duration of the optimization process. When Iterated F-race exhausts the maximum number of simulations, the optimization process ends and Mandarina returns the best candidate solution found so far. The control software produced by Mandarina is then deployed to the robots without further modification.

Mandarina operates with a design process similar to that introduced by Francesca et al. (2015) for Chocolate, which was also used in TuttiFrutti in Chapter 3. The main difference is that, as mentioned at the beginning of this Chapter, Mandarina uses Iterated F-race to address missions whose specifications define multiple design criteria to drive the design process. These design criteria are formally specified by a set of objective functions to be optimized. In our research, we focused on the bi-objective case: missions are characterized by two objective functions. To achieve this, we did not need to modify Iterated F-race. We simply set it up in a way that differs from previous implementations of AutoMoDe. In Chocolate and TuttiFrutti, as in the other existing implementations of AutoMoDe, Iterated F-race evaluates candidate solutions with respect to the single objective function that characterizes the mission at hand. In Mandarina, we set up Iterated F-race so as to evaluate each candidate solution on the two objective functions defined in

the mission specification.

Figure 4.1 illustrates the operation of Iterated F-race, the original optimization algorithm as used in `Chocolate` and `TuttiFrutti`, and the optimization algorithm as used in `Mandarina`. Iterated F-race operates in three main steps (López-Ibáñez et al. 2016): (i) sampling candidate instances of control software according to a particular distribution; (ii) selecting the best instance(s) from the newly sampled ones by racing; and (iii) updating and refining the sampling distribution in order to bias the sampling towards the best instance(s) found. In each race, Iterated F-race iteratively evaluates candidate finite-state machines on a number of *problem instances*. Each problem instance is a specific realization of the mission at hand—for example, a specific realization of initial conditions such as position/heading of the robots. While in `Chocolate` and `TuttiFrutti`, which address single-objective problems, the evaluation of a candidate solution on a problem instance produces a single number (the score in the mission); in `Mandarina`, it produces two numbers: the corresponding scores of the two objective functions.

Iterated F-race uses statistical tests—i.e., Friedman tests (Conover 1999)—to discard candidate solutions that significantly perform worse than at least one other candidate solution. The candidate solutions are selected with respect to the average rank of their observed performance and not with respect to the numerical values. In `Mandarina`, Iterated F-race conducts the statistical tests on the two scores that are returned, and discards candidate solutions that significantly perform worse than at least one other candidate solution in the two of them. Eliminating candidate instances of control software under concurrent design criteria is only possible because Iterated F-race operates on the ranks, and not on the score obtained from the objective functions. It is not always possible to directly compare the score of an instance of control software across two objective functions. However, it is indeed possible to compare the average rank of its observed performance across them.

To tackle multi-objective optimization problems, it is customary to first identify a set of solutions that represent various performance compromises across the objectives—an approximation of the Pareto set. From this set, an a posteriori decision-making process is applied to select a preferred solution with respect to the observed performance—as shown by Trianni and López-Ibáñez (2015). In `Mandarina`, we take a different approach. `Mandarina` returns a single solution: the one that statistically does not perform worst than any other solution in the two design criteria. In other words, it returns only one solution that consistently proved to be a good performing neutral compromise among the criteria considered.

From a practical point of view, the modifications we made to the design process

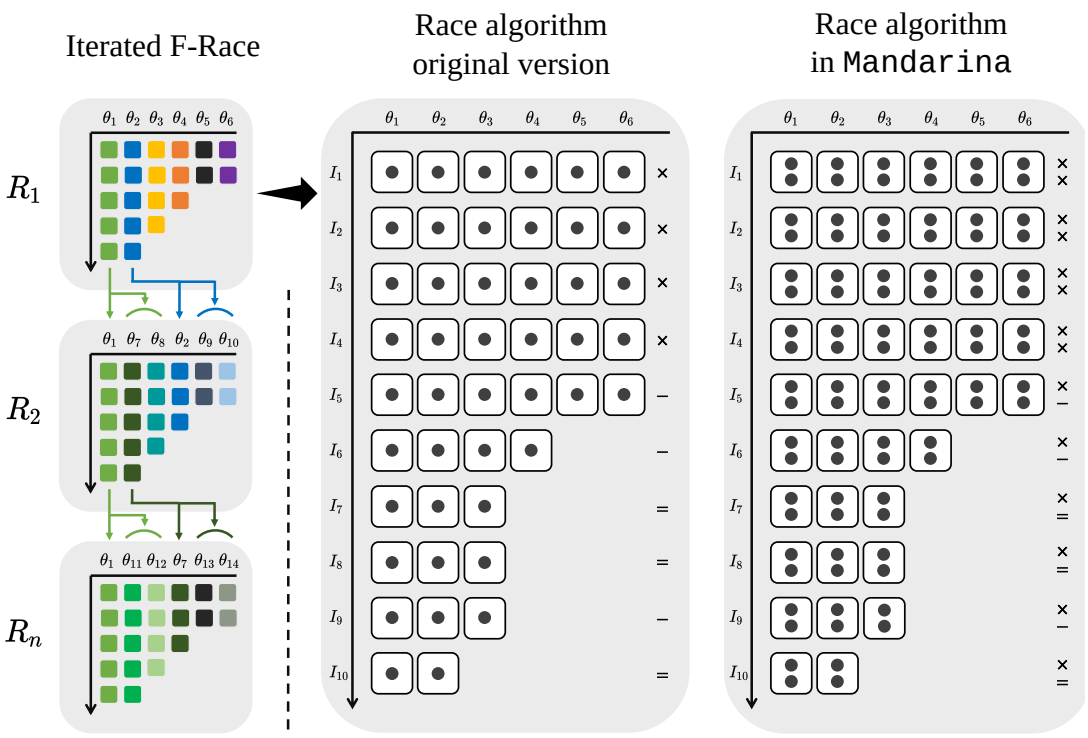


Figure 4.1: Illustration of Iterated F-race, the original race algorithm, and of how it is used in `Mandarina`. Iterated F-race conducts iterative races (R_i) in which candidate solutions (θ_j)—i.e., instances of the control software being developed—are evaluated with respect to a number of problem instances (I_k). Each box is the evaluation of one candidate solution on one problem instance. Dots (•) in a box represent the number of objective functions evaluated.

In the first race (R_1), Iterated F-race uniformly samples the parameter space to generate candidate solutions ($\theta_{\{1,\ldots,6\}}$). In the subsequent races ($R_{\{2,\ldots,n\}}$), Iterated F-race refines and updates the sampling distribution to bias the sampling towards the best instance(s) found so far. For example, in R_2, $\theta_{\{7,8\}}$ and $\theta_{\{9,10\}}$ result from biased sampling towards θ_1 and θ_2, respectively. The refinement of the sampling process is illustrated by the color gradient of the boxes.

The original race algorithm evaluates one single objective function. In `Mandarina`, Iterated F-race evaluates two objective functions. In a race, candidate solutions are discarded after conducting statistical tests: '×' indicates that no statistical test is performed; '–' indicates that the test discarded at least one candidate solution; and '=' indicates that the test did not discard any candidate solution. Statistical tests are conducted only after sufficient statistical evidence is gathered on the performance of the candidate solutions in an initial set of problem instances ($I_{1,\ldots,5}$).

`Mandarina` requires evaluating both objective functions on an instance I_k before conducting statistical tests to discard configurations. As no configuration can be discarded based on just one of the two objective functions, no statistical test is needed after evaluating only the first objective function of instance I_k.

originally conceived by Francesca et al. (2015) for AutoMoDe are minor. Indeed, in Mandarina, we only made very few technical adjustments to the typical use of Iterated F-race to enable the evaluation of candidate control software with respect to two objective functions. However, the conceptual implications of making these modifications in Mandarina are important. Iterated F-race can address multi-criteria design problems if the objective functions to be optimized concurrently are presented to the algorithm as independent problem instances. This property was originally unintended in the conception of Iterated F-race. In our research, we do not explore the extent to which this property generalizes and/or whether Iterated F-race could address design problems that consider more than two objective functions. However, we find reasonable to think that Iterated F-race could turn viable to tackle the design of robot swarms with respect to more than two criteria—provided that they can be presented as independent problem instances as well.

4.4 Experimental setup

We assess Mandarina on multi-criteria missions that are specified as bi-objective optimization problems. In the absence of well-defined benchmarks, we devise a framework in which we combined instances of aggregation, foraging, and coverage to create missions. The missions included in this framework draw inspiration from missions used in previous automatic design studies (Francesca et al. 2014b, 2015; Trianni and López-Ibáñez 2015; Garzón Ramos and Birattari 2020; Kuckling et al. 2022; Kegeleirs et al. 2024b). Alongside Mandarina, we also conduct experiments with baseline modular and neuroevolutionary design methods.

4.4.1 Mission framework

We experiment with a swarm of twenty e-puck robots that must perform missions composed of two sequential parts. Each part is a sub-mission to be accomplished and is evaluated by an independent objective function, a design criterion. The mission framework comprises a set of fifteen missions, which result from combining the six sub-missions into sets of two: $\binom{6}{2} = 15$. In these missions, the swarm must perform one sub-mission for a given amount of time, then switch its behavior to perform the second one for the same amount of time.

The robots operate in an octagonal arena of $2.75\,\mathrm{m}^2$ surrounded by MoCA's RGB blocks—see Figure 4.2. The robots are randomly positioned at the beginning of each experimental run. The RGB blocks are arranged in walls and each of

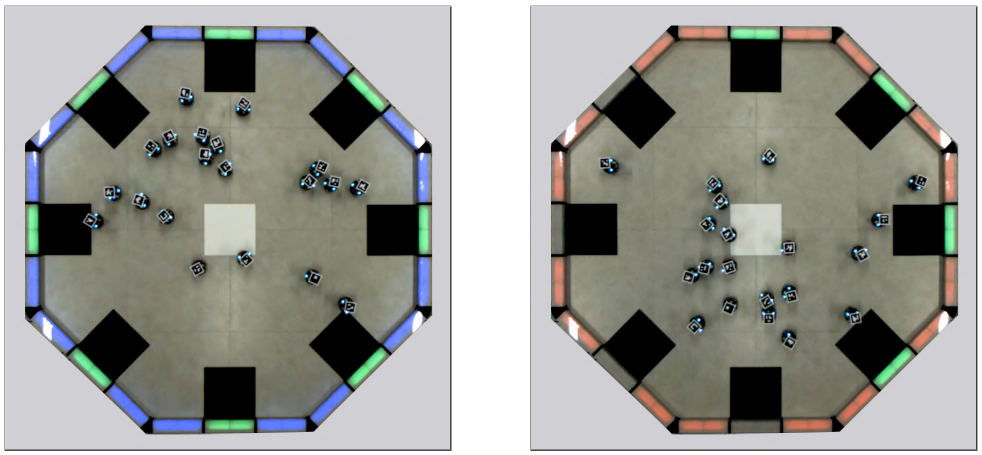


Figure 4.2: Experimental arenas for Mandarina's experiments. The figure shows examples of the two possible states of the arena. On the left, the RGB blocks of the walls display blue and all RGB blocks adjacent to a black patch are switched on, displaying green. On the right, the RGB blocks of the walls display red and some RGB blocks adjacent to a black patch have switched off, displaying no color. The robots are randomly positioned.

them can possibly display a different color. The floor of the arena is gray with nine square patches, each measuring 25 cm on each side. One of the patches is white, and the other eight are black. In every mission, the RGB blocks adjacent to black patches initially turn green, and afterward, they will randomly switch off with uniform probability. The remaining RGB blocks turn red or blue to inform the robots about the sub-mission to be executed. Typically, in automatic design studies, the arena is modified on a per-mission basis to allow for varying mission specifications. This is the procedure we followed in the experiments with TuttiFrutti, presented in Chapter 3. Here, we specify a diverse set of bi-criteria missions (and their sub-missions) in a single experimental arena by changing the color of MoCA's RGB blocks at run-time.

Sub-missions

We consider a set of six sub-missions (S$\{1, \ldots, 6\}$), which can be performed by e-pucks that comply with the reference model RM 3. These sub-missions are a sample of the class of missions that can be addressed with Mandarina and other methods that adopt the same reference model. Each sub-mission is specified by a description of a task to be executed and a corresponding objective function.

Sub-mission 1 (S1): the robots must occupy the black patches whose adjacent RGB blocks display green. The swarm is given 1 point for every 100 cumulative timesteps that the robots spend on each suitable patch. For example, a single robot

in a patch will be given one point after 100 timesteps, but 10 robots in a patch will be given 1 point after 10 timesteps. The score of the swarm is the number of points it obtains in the allotted time:

$$f_{\mathrm{S1}} = \sum_{t=1}^{T'} \sum_{i=1}^{H} I_i(t), \tag{4.1}$$

which must be maximized. H is the number of patches and T' the time available to the robots to perform the sub-mission. The indicator $I_i(t)$ is defined as:

$$I_i(t) = \begin{cases} 1, \text{ if at time } t \text{ the robots accumulate 100 timesteps in the patch } i; \\ 0, \text{ otherwise.} \end{cases}$$

Sub-mission S1 is inspired by aggregation missions in which the robots must gather at an indicated place (Francesca et al. 2014b, 2015).

Sub-mission 2 (S2): the robots must iteratively travel from any black patch to the white one. The swarm is given 1 point every time a robot completes a trip. The score of the swarm is the number of points it obtains in the allotted time:

$$f_{\mathrm{S2}} = I_{T'}\,, \tag{4.2}$$

which must be maximized. $I_{T'}$ is the number of trips executed in the time T' available to the robots to perform the sub-mission. Sub-mission S2 is inspired by foraging missions in which the robots must travel between two locations: a food source and a nest (Francesca et al. 2014b; Garzón Ramos and Birattari 2020).

Sub-mission 3 (S3): the robots must occupy the black patches adjacent to RGB blocks that are switched off. The swarm is given one point if at least two robots spend 50 timesteps in the corresponding black patch. The count of timesteps starts as soon as the two robots step into the patch, and it will continue as long as they both remain on it. The count is not affected if more than two robots occupy the patch. The score of the swarm is the number of points it obtains in the allotted time:

$$f_{\mathrm{S3}} = \sum_{t=1}^{T'} \sum_{i=1}^{H} I_i(t), \tag{4.3}$$

which must be maximized. H represents the number of patches and T' the time available to the robots to perform the sub-mission. The indicator $I_i(t)$ is defined

as:

$$I_i(t) = \begin{cases} 1, \text{ if at time } t \text{ two robots accumulate 50 timesteps in the patch } i;; \\ 0, \text{ otherwise.} \end{cases}$$

Sub-mission S3 is inspired by strictly cooperative missions in which the robots must jointly perform a single task (Ijspeert et al. 2001; Trianni and López-Ibáñez 2015).

Sub-mission 4 (S4): the robots must iteratively enter and leave the white patch. The swarm is awarded 1 point every time a robot performs these two actions. The score of the swarm is the number of points it obtains in the allotted time:

$$f_{\text{S4}} = I_{T'} \,, \tag{4.4}$$

which must be maximized. $I_{T'}$ is the number of times a robot entered and left the white patch in the time T' that is available to the robots to perform the sub-mission. Also sub-mission S4 is inspired by foraging missions, as sub-mission S2, but here, the robots start and end at a single location.

Sub-mission 5 (S5): the robots must disperse and cover the arena. We consider the coverage to be the most effective when the minimum distance between any two pair of robots is maximized. The score of the swarm is the cumulative sum of the minimum inter-robot distance, over time:

$$f_{\text{S5}} = \sum_{t=1}^{T'} \min \left(d_{ij}(t) \right) , \tag{4.5}$$

which must be maximized. Here, d_{ij} is the minimum distance between any pair of robots (i, j) at time t, and T' is the time available to the robots to perform the sub-mission. Sub-mission S5 is inspired by dispersion and coverage missions in which the robots must maintain a fixed inter-robot distance to achieve a specific spatial distribution (Francesca et al. 2015; Kegeleirs et al. 2024b).

Sub-mission 6 (S6): the robots must remain within a 25 cm distance from the walls of the arena, without entering the black patches. The score of the swarm is the aggregate time that the robots spend in the suitable areas:

$$f_{\text{S6}} = \sum_{t=1}^{T'} \sum_{i=1}^{N} I_i(t) \,, \tag{4.6}$$

which must be maximized. N is the number of robots and T' is the time available

to the robots to perform the sub-mission. The indicator $I_i(t)$ is defined as:

$$I_i(t) = \begin{cases} 1, \text{ if the robot } i \text{ is in gray floor and within 25 cm from a wall at time } t; \\ 0, \text{ otherwise.} \end{cases}$$

Sub-mission S6 is inspired by missions in which the robots must display a specific spatial distribution (Francesca et al. 2015; Kuckling et al. 2022), like sub-mission S5. However, the robots here must maintain a specific distance from an element in their environment, irrespective of the distance to their peers.

We selected the sub-missions S$\{1, \ldots, 6\}$ because they pose specific challenges to a multi-objective optimization process. First, the performance of candidate solutions is stochastic and depends on the robots' initial positions (e.g., in S5 and S6) and on the way the RGB blocks turn off (e.g., in S1 and S3). Second, estimating realistic bounds for the objective functions is not straightforward without conducting preliminary experiments (e.g., in S2 and S4). Indeed, it is challenging to anticipate to what degree the interactions between the robots will hinder a candidate solution from reaching a theoretical upper performance bound. Third, the objective functions can take discrete or continuous variables (e.g., in S4 and S5), and their range can span over different orders of magnitude (e.g., in S3 and S6). Finally, the objective functions can represent conflicting goals (e.g., in S1 and S3).

Specification of bi-criteria missions

We define the set M of bi-criteria missions ($m_{\mathrm{S}p.\mathrm{S}q}$) by pairing sub-missions (Sp, Sq) in fifteen combinations—see Table 4.1. In all cases, the robots must execute the two sub-missions, one after the other. The time T available to the robots to execute a mission is 120 s. The time T' available to execute each sub-mission is 60 s. Accordingly, the swarm's performance in a mission is assessed for the initial 60 s with regard to one sub-mission and for the remaining 60 s with regard to the other. The two scores are returned after each experimental run.

We expect the swarm to be able to perform a mission $m_{\mathrm{S}p.\mathrm{S}q}$ regardless of the order of Sp and Sq. Therefore, in our experiments, the order in which the sub-missions must be executed is randomly defined at the beginning of the evaluation of the mission (i.e., for each problem instance). For example, the execution of $m_{\mathrm{S}p.\mathrm{S}q}$ can result uniformly in the sequence S$p\rightarrow$ Sq or S$q\rightarrow$ Sp. We anticipate that using a fixed predefined order of sub-missions could lead to control software instances specialized for that specific order, making it likely to fail when the order

Table 4.1: Set of missions (M) considered in Mandarina's study. The missions are paired combinations ($m_{Sp.Sq}$) of the six sub-missions (S$\{1, \cdots, 6\}$). In each combination, the colors blue and red characterize the sub-missions Sp (■) and Sq (■). In a mission ($m_{Sp.Sq}$), the sub-missions Sp and Sq must be executed in sequence, but the order of the sequence (Sp ■ ⇄ ■ Sq) is randomly defined in every experimental run. Sp and Sq are executed during an equivalent period of time. The execution of a mission $m_{Sp.Sq}$ returns the score of the swarm with respect to Sp and Sq, regardless the order of the sequence.

No.	Mission	Combination of sub-missions
1	$m_{S1.S2}$	S1 ■ ⇄ ■ S2
2	$m_{S1.S3}$	S1 ■ ⇄ ■ S3
3	$m_{S1.S4}$	S1 ■ ⇄ ■ S4
4	$m_{S1.S5}$	S1 ■ ⇄ ■ S5
5	$m_{S1.S6}$	S1 ■ ⇄ ■ S6
6	$m_{S2.S3}$	S2 ■ ⇄ ■ S3
7	$m_{S2.S4}$	S2 ■ ⇄ ■ S4
8	$m_{S2.S5}$	S2 ■ ⇄ ■ S5
9	$m_{S2.S6}$	S2 ■ ⇄ ■ S6
10	$m_{S3.S4}$	S3 ■ ⇄ ■ S4
11	$m_{S3.S5}$	S3 ■ ⇄ ■ S5
12	$m_{S3.S6}$	S3 ■ ⇄ ■ S6
13	$m_{S4.S5}$	S4 ■ ⇄ ■ S5
14	$m_{S4.S6}$	S4 ■ ⇄ ■ S6
15	$m_{S5.S6}$	S5 ■ ⇄ ■ S6

of Sp and Sq changes. On the contrary, our stochastic approach helps minimize the risk of biasing the design process toward a specific order.

In every mission, we use a fixed color coding (blue and red) to characterize the associated sub-missions—see Table 4.1. At every moment in time, the walls of the arena display blue or red to indicate the sub-mission to be executed. For example, in a sequence Sq→ Sp, the walls will initially display the color blue, indicating that the swarm must execute the sub-mission Sq. After 60 seconds, the walls switch to red, indicating the end of Sq and the start of the sub-mission Sp. The color of the walls is an environmental signal that the robots can perceive. We anticipate that the design methods will design robot swarms that, although not pre-programmed to do so, will rely on the color of the walls to transition between behaviors and accomplish the two sub-missions—as in the experiments conducted with `TuttiFrutti`.

4.4.2 Baseline methods

We conduct experiments with three automatic design methods other than `Mandarina`: `TuttiFrutti` and `EvoColor`, introduced in Chapter 3, and `NEAT-Color`, an original implementation of the neuroevolutionary approach. Like `Mandarina`,

these three methods produce control software for e-pucks defined by the reference model RM 3, and are specialized in the design of collective behaviors for robots that can perceive and react to color signals. However, unlike `Mandarina`, they do not naturally work with concurrent design criteria. `TuttiFrutti`, `EvoColor`, and `NEAT-Color` are limited to using a single score as an evaluation criterion during the design process. We consider these methods as our baseline because, to address a bi-criteria design problem, they need to aggregate the concurrent design criteria into a single performance measure—the standard approach in the literature, as discussed in Chapter 2.

The implementations of `TuttiFrutti` and `EvoColor` in this study are identical to those from Chapter 3. `NEAT-Color` is a neuroevolutionary method in which the evolutionary process selects both the network topology and the synaptic weights of the artificial neural network. `NEAT-Color` is based on `NEAT`—an algorithm introduced by Stanley and Miikkulainen (2002) and tested by Hasselmann et al. (2021) in the context of the automatic design of robot swarm. The input and output nodes in `NEAT-Color` are the same as for `EvoColor`, which are defined according to the reference model RM 3–see Table 3.1. In `NEAT-Color`, the topology of the network is initialized with a disconnected network. `NEAT-Color` selects the network architecture and fine-tunes the synaptic weights via an evolutionary optimization process based on elitism and mutation. We include `NEAT-Color` as a more sophisticated alternative method to `EvoColor`.

In the experiments, we conduct the design process using `TuttiFrutti` and `EvoColor` as originally introduced in Chapter 3. The parameters for conducting the design process with `NEAT-Color` are the ones defined by Hasselmann et al. (2021) for `NEAT`. To apply these methods in our study, we considered popular approaches to combine the concurrent design criteria into a single performance measure.

4.4.3 Aggregating multiple criteria into a single criterion

We use `TuttiFrutti`, `EvoColor`, and `NEAT-Color` alongside unary metrics that aggregate the two scores of a mission into a single measure. We consider the weighted sum (`WS`), the hypervolume (`HV`), and the l^2-norm (`L2`). These unary metrics provide the single performance value that `TuttiFrutti`, `EvoColor`, and `NEAT-Color` require to conduct the design process.

The weighted sum approach (WS)

We scalarize the two objective functions ($f_{\mathrm{S}p}$ and $f_{\mathrm{S}q}$) of a mission ($m_{\mathrm{S}p.\mathrm{S}q}$) through a weighted sum ($f_{m_{\mathrm{S}p.\mathrm{S}q}} = (\alpha).f_{\mathrm{S}p} + (1-\alpha).f_{\mathrm{S}q}$) regulated by a parameter $\alpha \in [0,1]$. This metric can be used alongside `TuttiFrutti`, `EvoColor`, and `NEAT-Color`, but it requires defining a suitable α value for the mission before the design process begins. Suitable values for α are typically determined through mission-specific expert knowledge or prior experimentation, which help define the relationship between $f_{\mathrm{S}p}$ and $f_{\mathrm{S}q}$. However, we chose not to adopt either of these approaches, as relying on them would prevent us from achieving a fully automatic design process. Therefore, we consider the absence of a well-defined performance range for $f_{\mathrm{S}p}$ and $f_{\mathrm{S}q}$, which makes hardly possible to predefine suitable α values. To address this issue, we adopt the approach of Trianni and López-Ibáñez (2015) and we conduct experiments with a set of five α values in every mission ($\alpha \in \{0.2, 0.4, 0.5, 0.6, 0.8\}$).

By applying five α values to the weighted sum, the experiments yield a large set of instances of control software for each baseline method and mission. Therefore, a subsequent decision-making process is necessary to select solutions from this set. We conduct this decision-making process manually. After all experiments are completed, we evaluate the instances produced with the five α values and we rank them according to their score. For each method, we identify the best, median, and worst combination of α values for every mission. We then use these α values to assemble three sets of instances of control software for each method: the best (B), median (M), and worst (W) instances produced. It should be noted that this a posteriori manual selection process does not align with the tenets of a fully automatic design process (Birattari et al. 2019). However, we include it in our study to provide an estimate of the best, median, and worst performance one can expect when using a weighted sum in these missions.

The hypervolume approach (HV)

The hypervolume quantifies the area in the objective space that a candidate solution dominates. In our experiments, the sub-missions and their associated objective functions define the objective space for each mission. To calculate the hypervolume, the objective space must be normalized using a reference point, indicating the upper performance bounds of the two objective functions under consideration. However, in our experiments, we cannot pre-establish this normalization due to the absence of a score range for $f_{\mathrm{S}p}$ and $f_{\mathrm{S}q}$. Iterated F-race has a way to address this problem: it dynamically computes these bounds for every problem instance based on the

observed scores of all candidate solutions on that specific instance. Therefore, in the case of `TuttiFrutti`, the hypervolume can be computed using the implementation provided by the maintainers of the `irace` software package (López-Ibáñez et al. 2020), originally proposed by Fonseca et al. (2006). In the case of `EvoColor` and `NEAT-Color`, there is no straightforward way to dynamically compute the performance bounds during the evolutionary optimization process. Consequently, we do not apply the hypervolume to the neuroevolutionary methods.

In our experiments, we use the hypervolume in a way that differs from the typical approach found in the multi-objective optimization literature (Guerreiro et al. 2021). The hypervolume is often used to identify a set of solutions with different performance trade-offs—i.e., an aproximation of the Pareto front—and then select a preferred solution through an informed decision-making process. This was the approach followed by Trianni and López-Ibáñez (2015). Conversely, we use the hypervolume in `TuttiFrutti` to find a single no-preference solution to the design problem. `TuttiFrutti` will look for a solution that, on an individual basis, covers the largest area in the objective space across all tested problem instances. We expect that an instance that maximizes its coverage of the objective space will yield a neutral compromise solution, where the two sub-missions are performed at their best. We consider this application of the hypervolume a fully automatic design method because it does not require prior experimentation, or manual intervention and decision making after the design process.

The l^2-norm approach (`L2`)

We compute the Euclidean distance in the objective space between a reference point and the point defined by the two scores of a candidate solution. Like the hypervolume, `TuttiFrutti` uses Iterated F-race to dynamically compute performance bounds for each problem instance, normalizing the objective space and establishing an appropriate reference point. Then it aggregates the two scores of the candidate solution using the l^2-norm. In this case, we also do not apply the l^2-norm to `EvoColor` and `NEAT-Color` as they do not provide a straightforward way to dynamically compute the performance bounds.

The l^2-norm is a metric used in no-preference multi-objective optimization methods (Miettinen 1998). These methods aim at finding a single solution that satisfies a decision maker who has no special expectations or preference on performance trade-offs. In essence, all objective functions are treated as equally important, and the decision maker is satisfied with finding a single solution that optimizes them.

This reasoning aligns well with our multi-criteria design problem: our goal is to find instances of control software that enable the swarm to execute its mission ($m_{\mathrm{S}p.\mathrm{S}q}$) by performing well in the two sub-missions it comprises (Sp, Sq)—no other information is provided to the design process. We expect that `TuttiFrutti` will produce neutral compromise solutions by incorporating the l^2-norm. We consider this application of the l^2-norm also a fully automatic design method.

4.4.4 Protocol

Table 4.2 lists the design methods considered in our experiments. We use `Mandarina` and the baseline methods to produce control software for the fifteen missions. We generate 150 instances of control software with `Mandarina`—10 per mission. In the experiments involving the weighted sum, we produce a total of 2250 instances of control software using `NEAT-Color`, `EvoColor`, and `TuttiFrutti`—10 per method, mission, and α value. For each method, we then select sets of the best, median, and worst instances, each comprising 150 instances—10 per mission. We also produce 150 instances of control software using `TuttiFrutti` and the hypervolume, and other 150 using `TuttiFrutti` and the l^2-norm—10 per mission in each case. All design methods are given a budget of 100 000 simulation runs to produce each instance of control software.

The instances of control software produced by a design method are evaluated twice—once for each possible order of the sub-missions. We do this to assess the swarm's capability to perform the mission regardless of the order of the sub-missions. In all cases, the design process and the evaluation of the control software is conducted using ARGoS3.

We also produce 180 demonstration videos of the behavior of the robots when the control software produced in simulation is ported to physical robots. We present 30 videos for each method among `NC-WS-`B, `EC-WS-`B, `TF-WS-`B, `TF-HV`, `TF-L2`, and `Mandarina`. They demonstrate the behavior of instances of control software for every mission and for each possible order of the sub-missions.

Statistics

We use a Friedman rank sum test (Conover 1999) to compare the relative performance of the methods across all 15 missions, taking into account the evaluations conducted in each sub-mission. Specifically, we conduct a Friedman two-way analysis of variance with repeated measures. This rank-based non-parametric test uses a block design. In our protocol, the treatment factor is the method under analysis

Table 4.2: Design methods under evaluation in Mandarina's study. We consider three versions of NEAT-Color, EvoColor and TuttiFrutti that adopt the weighted sum. Each of these versions comprises a set of control software instances produced using the best, median, and worst combinations of α values identified for each mission. We also consider two additional versions of TuttiFrutti, one that adopts the hypervolume and other that adopts the l^2-norm. The last method method under evaluation is Mandarina.

No.	Label	Method	Unary metric	Set of instances	Approach
1	NC-WS-W	NEAT-Color	weighted sum	worst	neuroevolution
2	NC-WS-M	NEAT-Color	weighted sum	median	neuroevolution
3	NC-WS-B	NEAT-Color	weighted sum	best	neuroevolution
4	EC-WS-W	EvoColor	weighted sum	worst	neuroevolution
5	EC-WS-M	EvoColor	weighted sum	median	neuroevolution
6	EC-WS-B	EvoColor	weighted sum	best	neuroevolution
7	TF-WS-W	TuttiFrutti	weighted sum	worst	AutoMoDe
8	TF-WS-M	TuttiFrutti	weighted sum	median	AutoMoDe
9	TF-WS-B	TuttiFrutti	weighted sum	best	AutoMoDe
10	TF-HV	TuttiFrutti	hypervolume	n.a.	AutoMoDe
11	TF-L2	TuttiFrutti	l^2-norm	n.a.	AutoMoDe
12	Mandarina	Mandarina	n.a.	n.a.	AutoMoDe

and the blocking factor is the sub-missions. We present the results of the Friedman test with the average rank of the methods along with its 95% confidence interval. The average rank indicates the relative performance of a method with respect to the others across the complete set of experimental results. We selected the Friedman test to compare methods because it is invariant to the magnitude of the objective functions of the sub-missions and can be applied with no assumptions about the distribution of the performance data. First adopted by Francesca et al. (2015) in the initial studies with Chocolate, the Friedman test has since proven to be a valuable tool for assessing the relative expected performance of automatic design methods.

In our experiments, we use different performance measures to conduct the optimization process (scores, hypervolume, and the l^2-norm). A priori, focusing solely on the aggregated results based on one of these measures could unfairly favor the design methods that use that same measure. Therefore, to make fair comparisons, we report the relative aggregate performance of the methods when evaluated according to the three: the scores, hypervolume, and the l^2-norm. We present results of three Friedman tests computed according to the scores, hypervolume, and the l^2-norm. By considering and presenting them all, we compensate for possible bias. A method is considered significantly better than another if it has a lower average rank, and there is no overlap in the confidence intervals of the two methods.

The per-mission results are presented through scatter plots that show the performance of control software instances in the objective space—the scores. These scatter plots are provided for each mission. For the weighted sum approach, we only include the best sets from `NEAT-Color`, `EvoColor`, and `TuttiFrutti` as they illustrate the best performance one could expect.

4.5 Results

In Chapter 3, we discussed how `TuttiFrutti` addresses individual missions using environmental and inter-robot signaling, presenting the results on a per-mission basis. Given that `Mandarina` and `TuttiFrutti` can generate identical control software and address the missions in similar ways, we shift our focus here to the differentiating element of `Mandarina`: the optimization processes. In the following, we focus on the overall relative performance of the methods rather than how they address individual missions. We present the results on a per-method basis, starting with those obtained with AutoMoDe, followed by the results obtained with neuroevolution. We briefly comment on the observed behavior on physical robots at the end of the section.

Figure 4.3 shows three statistical comparisons of the methods under analysis. Figure 4.3.A shows the results of a Friedman test that is applied directly to the scores obtained from the evaluation of the generated control software. Figure 4.3.B also shows the results of a Friedman test, but in this case the test is applied after aggregating the scores using the hypervolume. Figure 4.3.C shows the results of a Friedman test that is applied after aggregating the scores using the l^2-norm. Figures 4.4 and 4.5 show scatter plots with the scores obtained for each mission.

4.5.1 AutoMoDe methods

`Mandarina`

In all three Friedman tests (Figure 4.3), the average rank of `Mandarina` is significantly lower than the average rank of the baseline methods. `Mandarina` outperformed the baselines, whether the comparison is conducted directly using scores, the hypervolume, or the l^2-norm. This indicates that `Mandarina` was more effective than the other methods in producing neutral compromise instances of control software that satisfied the design criteria. In the per-mission results (Figures 4.4 and 4.5), the score of `Mandarina`'s instances is mostly concentrated in the top-right

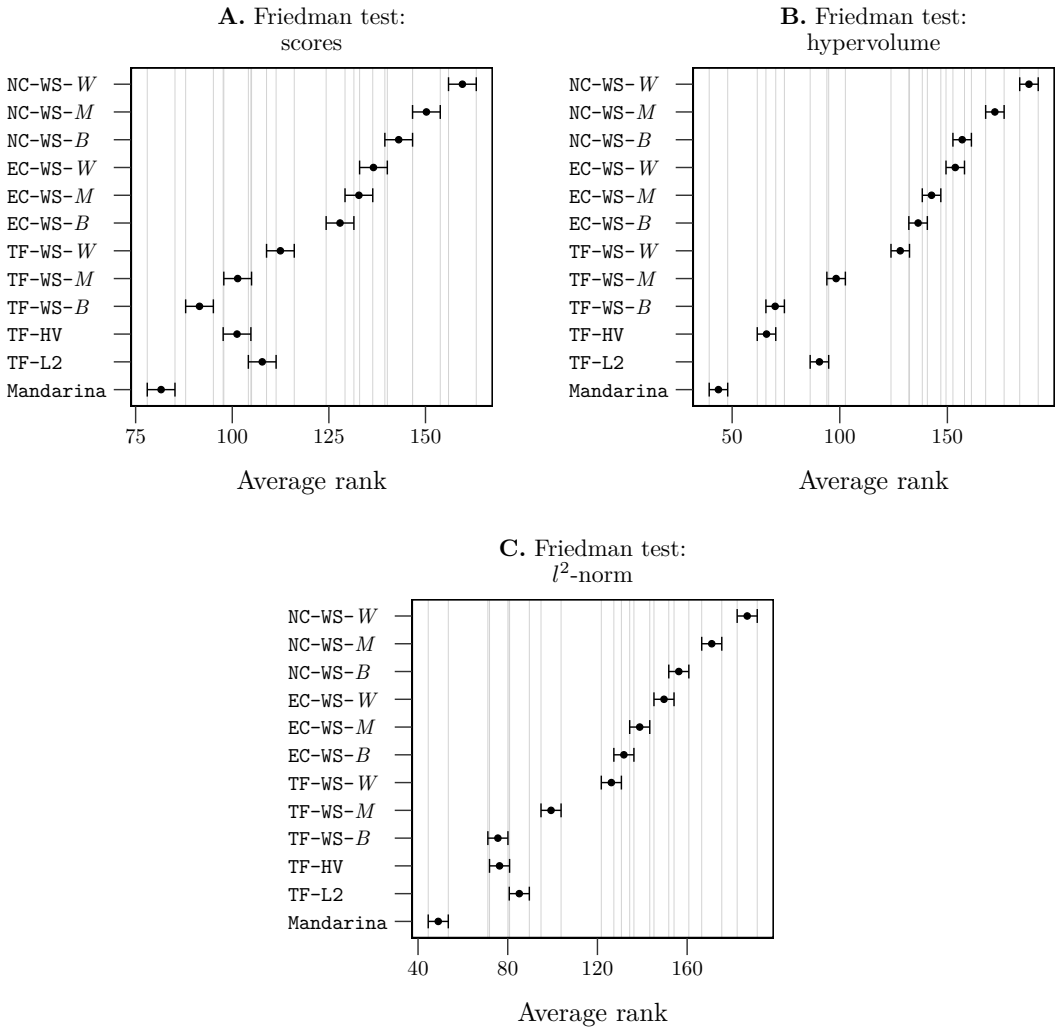


Figure 4.3: Friedman tests with aggregate results from `Mandarina`'s study. A. The test is directly applied to the scores. B. The test is applied after aggregating the scores using the hypervolume. C. The test is applied after aggregating the scores using the l^2-norm. In all cases, the test considers the results obtained across all missions and provides a relative ranking between the methods. The plot shows the average rank of each method and its 95 % confidence interval. The lower the rank, the better.

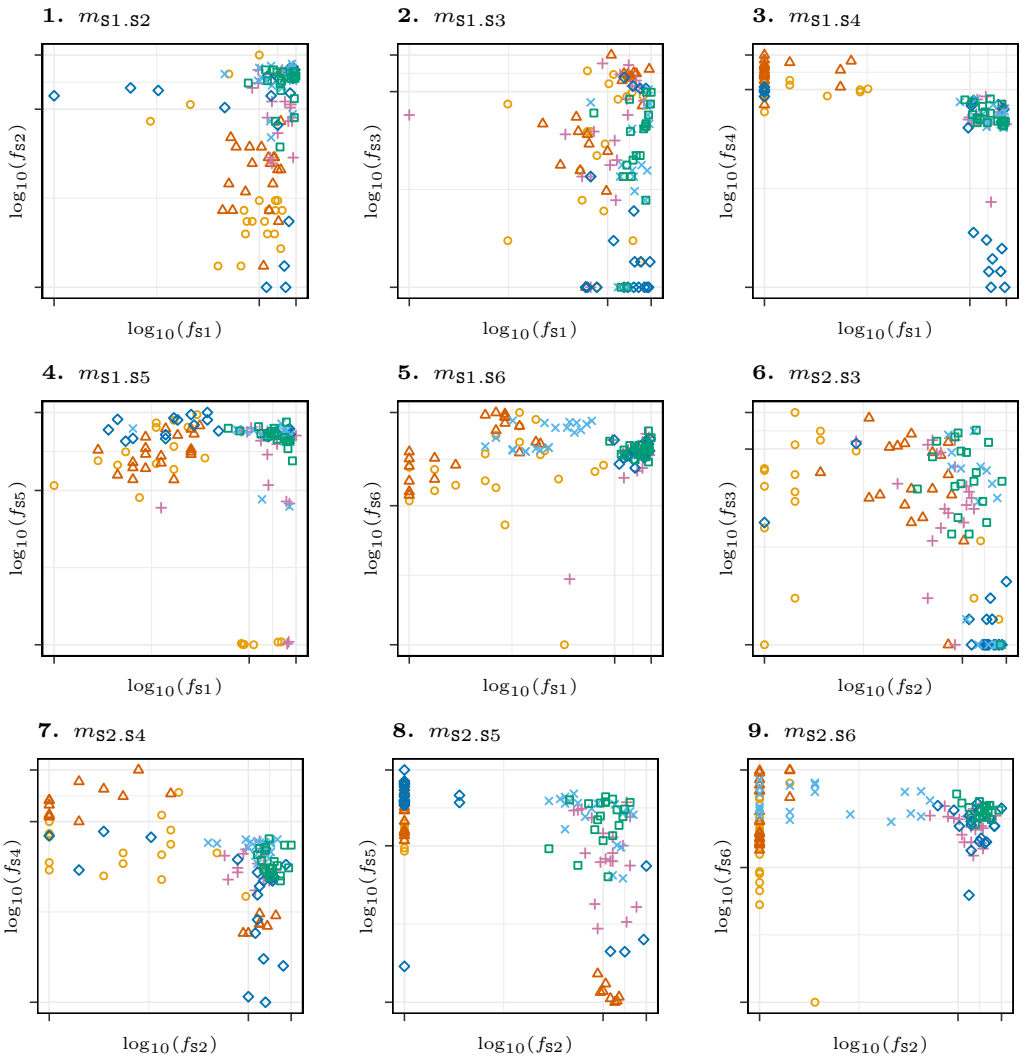


Figure 4.4: Performance obtained in the missions studied with `Mandarina`: missions 1 to 9. The scatter plots show the performance (scores) of control software instances in the objective space for the 15 missions, and for the methods `NC-WS-`B (○), `EC-WS-`B (△), `TF-WS-`B (×), `TF-HV` (+), `TF-L2` (◇), and `Mandarina` (□). The results are displayed using a logarithmic scale to accommodate the performance ranges of the sub-missions, which span varying orders of magnitude. In all plots, the higher, the better. These results have been obtained with simulations in ARGoS3.

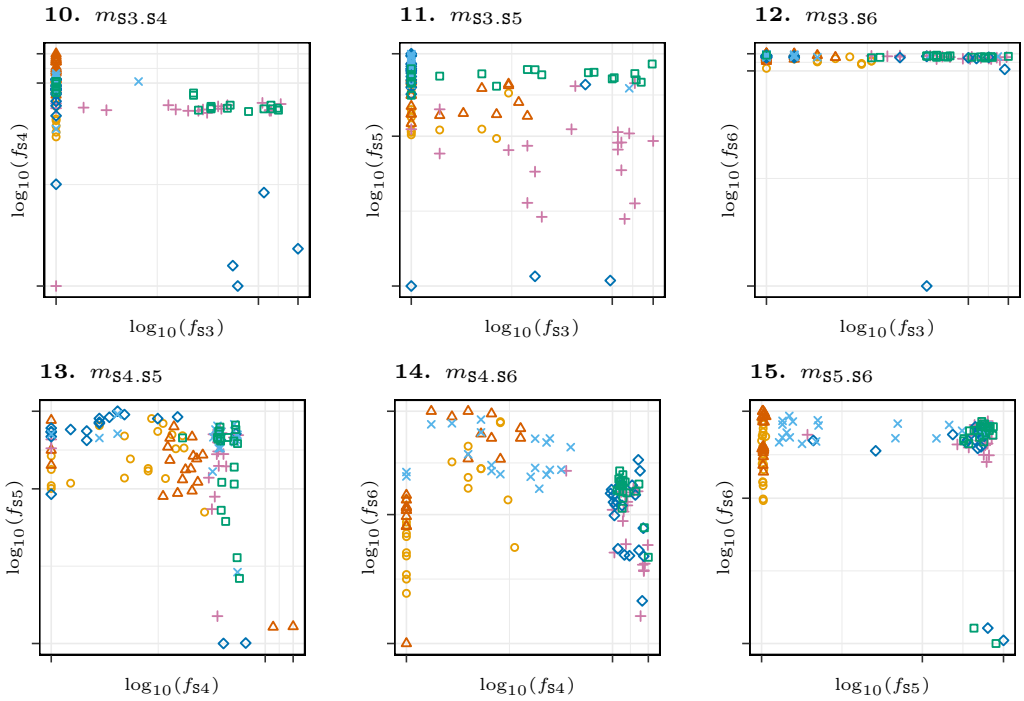


Figure 4.5: Performance obtained in the missions studied with `Mandarina`: missions 10 to 15. The scatter plots show the performance (scores) of control software instances in the objective space for the 15 missions, and for the methods `NC-WS-`B (○), `EC-WS-`B (△), `TF-WS-`B (×), `TF-HV` (+), `TF-L2` (◇), and `Mandarina` (□). The results are displayed using a logarithmic scale to accommodate the performance ranges of the sub-missions, which span varying orders of magnitude. In all plots, the higher, the better. These results have been obtained with simulations in ARGoS3.

region of the scatter plots. This shows that, in most cases, the collective behaviors designed by Mandarina aim at maximizing the performance of the swarm in the two sub-missions.

TuttiFrutti and the weighted sum approach

By exploring a wide range of weight combinations, we generated a repertoire of control software instances that satisfy the design criteria in varying degrees. From this repertoire, we identified the best (TF-WS-B), median (TF-WS-M), and worst (TF-WS-W) sets using the post hoc decision-making process described in Section 4.4. In the three Friedman tests (Figure 4.3), the average rank of the best, median, and worst sets of instances of TuttiFrutti is significantly lower than the average rank of the sets of EvoColor and NEAT-Color. TuttiFrutti generated sets of instances that generally outperform those produced by EvoColor and NEAT-Color when using the weighted sum approach. On the other hand, the relative performance between the selected sets (TF-WS-B, TF-WS-M, TF-WS-W) and the other variants of TuttiFrutti (TF-HV and TF-L2) differs depending on the metrics used to conduct the test. TF-WS-B has a significantly lower rank than TF-HV and TF-L2 when the instances are directly evaluated with respect to the scores (Figure 4.3.A). However, the rank difference fades if the instances are evaluated with respect to the hypervolume and the l^2-norm—see Figures 4.3.B and 4.3.C, respectively. In the per-mission results (Figures 4.4 and 4.5), the score of TF-WS-B's instances is spread in the top-right region of the scatter plots. This indicates that the collective behaviors designed by TF-WS-B aim at maximizing the performance of the swarm in the two sub-missions, in most cases.

TuttiFrutti and the hypervolume approach

In all Friedman tests (Figure 4.3), TF-HV has a significantly lower rank than the methods based on EvoColor and NEAT-Color. However, compared to other methods based on TuttiFrutti, the average rank of TF-HV also varied depending on the metrics used to conduct the tests. If the instances are evaluated with respect to the hypervolume, TF-HV ranks similarly to TF-WS-B (the best set)—see Figure 4.3.B. However, if the instances are directly evaluated using the scores, the average rank of TF-HV is similar to that observed in TF-WS-M (the median set)—see Figure 4.3.A. A similar phenomenon appears when comparing TF-HV and TF-L2. If the instances are evaluated with respect to the hypervolume, the rank of TF-HV is significantly lower than that of TF-L2—see Figure 4.3.B. However, when the instances are

evaluated based on the l^2-norm, which is the measure used in the optimization process of TF-L2, our experimental setup was unable to detect significant differences between TF-HV and TF-L2—see Figure 4.3.C. In the per-mission results (Figures 4.4 and 4.5), the score of TF-HV's instances is spread near the top-right region of the scatter plots. This shows that most of the designed collective behaviors also aim at maximizing the performance of the swarm in the two sub-missions.

TuttiFrutti and the l^2-norm approach

The average rank of TF-L2 is significantly lower than that of the methods based on EvoColor and NEAT-Color in the three Friedman tests (Figure 4.3). However, as discussed previously, the relative average rank of TF-L2 compared to other methods based on TuttiFrutti varies depending on the metric used to conduct the tests. If the instances are directly evaluated with respect to the scores, the average rank of TF-L2 is between TF-WS-M and TF-WS-W—see Figure 4.3.A. This rank shows a sub-par performance for the method. If the instances are evaluated with respect to the hypervolume, the average rank of TF-L2 is situated between TF-WS-B and TF-WS-M—see Figure 4.3.B. Unlike the previous case, this second rank shows that the method has moderate performance. TF-L2 ranks the best if the instances are evaluated with respect to the l^2-norm—see Figure 4.3.C. In this third case, we did not observe a significant difference between the average rank of TF-L2 and that of TF-WS-B and TF-HV, and the average rank of TF-L2 is significantly lower than that of TF-WS-M. In the per-mission results (Figures 4.4 and 4.5), TF-L2 instances scored with mixed performance. In a minor share of missions, the instances are distributed close to the top-right region of the scatter plots. However, in a larger share, they concentrate near the axes of the scatter plots. This shows that, in most cases, TF-L2 failed to consistently design collective behaviors that aim at maximizing the performance of the swarm in the two sub-missions.

4.5.2 Neuroevolutionary methods

EvoColor and the weighted sum approach

Likewise with TuttiFrutti, we also used EvoColor to generate a repertoire of behaviors and we identified the best (EC-WS-B), median (EC-WS-M), and worst (EC-WS-W) sets. In the three Friedman tests (Figure 4.3), the average rank of the sets of EvoColor is only significantly lower than that of the sets of NEAT-Color. All other alternative methods outperformed EvoColor, except for NEAT-Color.

The per-mission results (Figures 4.4 and 4.5) show that the score of EC-WS-B's instances is concentrated near the axes of the scatter plots, attaining high values in a single sub-mission but not in the other. Instead of generating neutral compromise solutions that maximize the performance in both sub-missions, EvoColor primarily designed collective behaviors that prioritize maximizing the performance of one of them. By only addressing a single design criterion, EvoColor instances are able to achieve the highest performance values in single sub-missions—compared to the neutral compromise solutions produced by the modular methods. However, by doing so, EvoColor did not produce instances of control software that fully satisfy all the design criteria.

NEAT-Color and the weighted sum approach

The experiments with NEAT-Color showed results that are qualitatively similar to those obtained with EvoColor. Also in this case, we generated a repertoire of behaviors and we identified the best (NC-WS-B), median (NC-WS-M), and worst (NC-WS-W) sets. In the three Friedman tests, all alternative methods outperformed the sets of NEAT-Color—see Figure 4.3. The per-mission results (Figures 4.4 and 4.5) show that, like EC-WS-B, the score of NC-WS-B's instances concentrates near the axes of the scatter plots. However, in general, NEAT-Color's instances often achieve lower scores compared to EvoColor's instances. The collective behaviors designed by NEAT-Color also prioritize maximizing the performance of a single design criterion. In this sense, NEAT-Color also failed to produce instances of control software that fully satisfy the design criteria.

4.5.3 Assessment with physical robots

We assessed the control software produced by NC-WS-B, EC-WS-B, TF-WS-B, TF-HV, TF-L2, and Mandarina in physical e-pucks. Video demonstrations are provided in the Supplementary Videos of the dissertation (Garzón Ramos 2025). The demonstrations show that modular methods—TF-WS-B, TF-HV, TF-L2, and Mandarina—transfer better from simulation to reality than those based on neuroevolution—NC-WS-B, EC-WS-B. In the case of modular methods, the behavior observed in physical e-pucks qualitatively resembles that observed in simulation. Conversely, in the case of neuroevolutionary methods, the e-pucks do not show any meaningful behavior that resembles the one observed in simulation. These results are consistent to those obtained with TuttiFrutti and EvoColor in the experiments presented in Chapter 3. Neuroevolutionary methods tend to overfit the simulation environment,

which prevents their control software from porting well to physical robots. Our results show that this phenomenon also seems to occur in the multi-criteria design problem we considered in this Chapter.

4.6 Discussion

The optimization-based design of robot swarms (fully or semi-automatic) has been traditionally studied on missions specified as single-criterion design problems, whether they are inherently single-criterion or adapted from multi-criteria ones. Consequently, there is currently no established experimental framework for specifying multi-criteria missions and studying how automatic design methods perform when addressing them. Building on our previous study with `TuttiFrutti`, we developed a new framework to explore this design problem.

To conduct our experiments, we specified bi-criteria missions by combining instances of well-known swarm robotics problems—foraging, aggregation, and coverage—into missions consisting of sequential parts. More precisely, (i) we specified a diverse set of sub-missions to be combined in a set of missions, (ii) we systematically performed experiments on the resultant combinations, and (ii) we compared the methods on their aggregated performance across all missions in the set. These sequential missions allowed us to estimate the expected performance of the methods under study and demonstrated to be an appropriate experimental framework for research on multi-criteria design. In our experiments, we focused on studying a bi-criteria case, but we believe the framework can be extended to a larger set of concurrent design criteria. Likewise, the missions we devised here are intended to be performed by e-puck robots that comply with the reference model RM 3, but we believe that the experimental protocol can be used to develop new experimental setups for other robot platforms or environments.

The missions in our experiments are distinctive from the related literature in that they consist of a combination of sub-missions that must be executed sequentially. In this class of missions, the modular methods (`Mandarina` and `TuttiFrutti`) outperformed the neuroevolutionary ones (`EvoColor` and `NEAT-Color`). `Mandarina` and `TuttiFrutti` designed robot swarms that, as anticipated, can use the environmental signals (red and blue lights) to switch between behaviors and execute each sub-mission with a tailored action. On the contrary, `EvoColor` and `NEAT-Color` designed robot swarms that do not seem to react to the color signals. Instead, they typically stick to a single behavior that either addresses the two

sub-missions to a very limited extent, or that is only tailored to address one of them. By observing the results of the two approaches, we argue that the modular methods performed better because they succeeded in designing robot swarms that can switch behaviors at run-time, which we consider necessary for executing the sequential missions. Switching between behaviors is a rather complex action inherently enabled on `Mandarina` and `TuttiFrutti` due to their modular control architecture—i.e., the probabilistic finite-state machine. The neuroevolutionary methods, on the contrary, must evolve the switching behavior from scratch within the artificial neural network. Neither `EvoColor` nor the more sophisticated `NEAT-Color` achieved this task successfully.

In `Mandarina`, we leverage Iterated F-race's non-parametric, rank-based nature to address challenges that arise in multi-criteria design problems. On the one hand, thanks to its non-parametric nature, we could use `Mandarina` in all missions without requiring information about the behavior/distribution of the objective functions. The objective functions of the sub-missions were ultimately treated as black-boxes during the optimization process. On the other hand, thanks to its rank-based nature, `Mandarina` could effectively handle pairs of objective functions with different score ranges, discrete and continuous, and across various orders of magnitude.

We compared `Mandarina` with other alternative methods that also use Iterated F-race—i.e., the baselines derived from `TuttiFrutti`. The aggregated results show that `Mandarina` achieved significantly better performance. `Mandarina` differs from the multi-criteria methods based on `TuttiFrutti` in that Iterated F-race directly operates on the scores obtained in each sub-mission. Indeed, `Mandarina` operates without normalization or estimation of the performance bounds of the objective functions. On the contrary, in the methods based on `TuttiFrutti`, Iterated F-race operates on performance estimations that result from aggregating the scores into single performance measures—the weighted sum, the hypervolume, or the l^2-norm. We conjecture that in `Mandarina`, Iterated F-race could perform more accurate statistical comparisons of the performance of the candidate solutions by directly comparing the scores of each sub-mission. This could possibly have contributed to the search for better performing control software instances. In the methods based on `TuttiFrutti`, aggregating the scores into a single measure may have hindered the ability of Iterated F-race to statistically distinguish between the performance of candidate solutions.

We dedicated a substantial part of our research to estimating the expected performance of weighted sum methods with respect to the other approaches under

evaluation. To do so, we conducted a systematic search to find suitable weight combinations for each method and mission individually. `TF-WS-`B, `EC-WS-`B, and `NC-WS-`B are sets of control software instances generated using the best weight combinations found in the search. In a sense, these three sets represent the expected outcome of a design process in which a designer is able to set appropriate weights to scalarize the objective functions across all missions. The results of our experiments showed that, even in this best-case scenario, the most effective weighted sum method (`TF-WS-`B) did not outperform other multi-criteria approaches (e.g., `TF-HV` and `TF-L2`). On the contrary, it was outperformed by one of them: `Mandarina`. The weighted sum is currently the common approach for framing/addressing multi-criteria problems in the optimization-based design of robot swarms. Here, we provide empirical evidence to support the idea that exploring multi-criteria design methods beyond the conventional weighted sum is worthwhile. First, a designer can avoid the time-consuming and costly process of manually searching for suitable weights. Second, methods like `Mandarina` can produce solutions that perform better than those obtained with the weighted sum.

The results we obtained with `Mandarina` motivated us to further investigate other problems that also consider multiple design criteria in the fully and semi-automatic design of robot swarms. As we will show in Chapter 5, we applied notions of multi-criteria design, and the mission framework presented here, to the generation of control software for robot swarms by demonstration.

5. Further applications

`TuttiFrutti` and `Mandarina` introduced original ideas that enlarged the range of problems that AutoMoDe can address. Building on these ideas, we further explored the versatility of AutoMoDe by developing four new specialized methods. In this chapter, we present `Mate`, `Habanero`, `Pistacchio`, and DTF-`MO`. We conceived these four methods to study the automatic design of spatially-organizing behaviors, stigmergy-based coordination, robot shepherding, and the realization of collective behaviors by demonstration.

Throughout the development of this thesis, we made efforts to present and communicate our research not only to the scientific community, but also to the general public. We conclude this chapter by discussing the outcome of these outreach efforts.

We present a series of studies in which we leverage environmental and inter-robot signaling to address design problems in the realization of robot swarms. For each design problem, we describe the objective of the study, the AutoMoDe method we developed to address it, the experiments conducted, the results obtained, and we provide a brief discussion. To maintain consistency throughout the chapter, we present the methods with a systematic structure, occasionally highlighting common elements across them. We focus on commenting on the most relevant points for each study. For a detailed description of the research work to be presented here, we refer the reader to the original publications by Mendiburu et al. (2022), Salman et al. (2024), Garzón Ramos and Birattari (2024), and Szpirer et al. (2024a).

5.1 Automatic design of spatially-organizing behaviors

Spatially-organizing behaviors involve the arrangement and distribution of robots and objects in space. As discussed in Chapter 2, the first studies with `Chocolate` tested the generation of spatially-organizing behaviors (Francesca et al. 2015). However, although `Chocolate` was equipped with modules that were though capable of producing spatially-organizing behaviors, the results were unsatisfactory when the robots were required to maintain specific relative positioning. The swarm was able to occupy designated regions in the arena, but the robots failed to achieve meaningful spatial distribution once there. The main hypotheses for this outcome have been that (i) `Chocolate` actually does not have a proper set of modules to effectively address this class of missions, or (ii) the method fails to properly combine the modules into good-performing control software.

We developed `Mate` to investigate this issue. `Mate` is an AutoMoDe method specialized in designing spatially-organizing behaviors for robot swarms. It builds on `Chocolate` and, like other AutoMoDe methods, it assembles predefined software modules into probabilistic finite-state machines. The single difference between `Mate` and `Chocolate` is the addition of a new module: FORMATION, a low-level behavior that enables robots to form hexagonal patterns. By introducing this module, which facilitates the relative positioning of robots, we expect `Mate` to overcome the limitations of `Chocolate`.

FORMATION is a low-level behavior that drives the robot using virtual forces. The robots detect each other and respond to artificial forces based on their relative positions. These forces are calculated using the Lennard-Jones potential

model (Jones 1924). To allow selective responses to these forces, we incorporated a signaling protocol using the robots' LEDs and cameras—a restricted version of TuttiFrutti's signaling. Robots running FORMATION signal to each other to maintain the desired relative distance using their LEDs, according to the Lennard-Jones model. When a sufficient number of robots maintain this specified distance, the swarm achieves a structured spatial distribution, forming a pattern.

We evaluated Mate in three missions in which the performance of the swarm depends on the robots' ability to operate within spatial distribution constraints. In these experiments, we compared the performance of Mate with that of Chocolate and a neuroevolutionary baseline. If the addition of FORMATION leads to a significant improvement in Mate's performance over Chocolate, it would suggest that Chocolate's task-agnostic modules are likely not well-suited for designing spatially-organizing behaviors, contrary to the initial expectations. In this sense, we also expect that Mate will be capable of selectively using the signaling capability embedded in FORMATION to obtain performance advantages over Chocolate, which cannot rely on it.

The following sections provide a brief description of Mate, its characteristic elements, and the experiments we conducted with the method.

5.1.1 AutoMoDe-Mate

Robot platform

Mate produces control software for the same version of the e-puck considered in TuttiFrutti—see Figure 3.1. However, the functional capabilities of the robot are adapted to meet the requirements of Chocolate and the FORMATION module. Mate operates on an e-puck whose capabilities are formally defined by the reference model RM 3.1—see Table 5.1.

Modular control architecture

Mate generates control software by fine-tuning and assembling predefined software modules into probabilistic finite-state machines. It operates on the twelve parametric software modules that were originally conceived for Chocolate and one additional module named FORMATION—see Table 5.2. The parameters of the software modules are automatically tuned during the design process.

The new module, FORMATION, is a low-level behavior in which the robot is subject to virtual forces that are computed through the Lennard-Jones poten-

Table 5.1: The control interface for the e-puck according to the reference model RM 3.1. Robots can perceive a cyan (C) signal. Robots can display no color ($\varnothing$) or cyan (C). V_c is calculated likewise V_n—the positions of perceived signals are aggregated into a unique vector. V_{LJ} is a force vector computed using the Lennard-Jones model, considering robots perceived as displaying cyan.

Input	**Value**	**Description**
$prox_{i\in\{1,\ldots,8\}}$	$[0,1]$	reading of proximity sensor i
$light_{i\in\{1,\ldots,8\}}$	$[0,1]$	reading of light intensity sensor i
$gnd_{j\in\{1,\ldots,3\}}$	$\{black, gray, white\}$	reading of ground sensor j
n	$\{0,\ldots,20\}$	number of neighboring robots detected
V_n	$([0.5,20];[0,2]\,\pi\,\mathrm{rad})$	their relative aggregate position
$cam_{c\in\{C\}}$	$\{yes, no\}$	color cyan perceived
V_{LJ}	$([0,1];[0,2]\,\pi\,\mathrm{rad})$	Lennard-Jones force vector
Output	**Value**	**Description**
$v_{k\in\{l,r\}}$	$[-0.12,0.12]\,\mathrm{m\,s^{-1}}$	target linear wheel velocity
$LEDs$	$\{\varnothing, C\}$	color cyan displayed by the LEDs

Period of the control cycle: 0.1 s.

tial (Jones 1924). This molecular interaction model has already proven suitable for developing and optimizing distributed control strategies that allow robot swarms to form hexagonal lattices (Pinciroli et al. 2008a,b). As in other implementations of the artificial potential field approach (Spears et al. 2004), a group of robots executing FORMATION tends to adopt a spatial configuration that minimizes its overall potential energy. That is, robots aim to remain in equilibrium positions that are reached when the swarm is homogeneously distributed in space. A robot executing FORMATION reacts to virtual forces that originate from one or more neighboring peers executing FORMATION as well. FORMATION sums the effects of the Lennard-Jones force with the short-range obstacle avoidance that is also embedded in other low-level behaviors of `Chocolate`. FORMATION aggregates all virtual forces that act on the robot and computes a desirable motion vector.

The e-pucks that execute FORMATION display cyan using their LEDs. This signal allows the robots to detect and locate nearby peers also executing FORMATION, using their omnidirectional cameras. The force vector V_{LJ} that drives the motion of the robot is computed through an average sum of the Lennard-Jones force for the all robots perceived with the camera. Robots executing FORMATION aim to remain equidistant at a target distance with respect to neighboring peers—defined by the parameter d^*. We expect that by doing so they will tend to form hexagonal patterns——as previously reported for the Lennard-Jones model (Kellogg et al. 2002). The size and density of the patterns depend on the target distance specified

Table 5.2: `Mate`'s software modules. The modules are defined on the basis of the reference model RM 3.1, see Table 5.1.

Low-level behavior	**Parameter**	**Description**
EXPLORATION	$\{\tau\}$	movement by random walk
STOP	n.a.	standstill state
ATTRACTION	$\{\alpha\}$	physics-based attraction to neighboring robots
REPULSION	$\{\alpha\}$	physics-based repulsion from neighboring robots
PHOTOTAXIS	n.a.	physics-based attraction to a light source
ANTI-PHOTOTAXIS	n.a.	physics-based repulsion from a light source
FORMATION	$\{d^*\}$	motion driven by Lennard-Jones force
Transition condition	**Parameter**	**Description**
BLACK-FLOOR	$\{\beta\}$	black floor beneath the robot
GRAY-FLOOR	$\{\beta\}$	gray floor beneath the robot
WHITE-FLOOR	$\{\beta\}$	white floor beneath the robot
NEIGHBOR-COUNT	$\{\xi, \eta\}$	number of neighboring robots greater than ξ
INV-NEIGHBOR-COUNT	$\{\xi, \eta\}$	number of neighboring robots less than ξ
FIXED-PROBABILITY	$\{\beta\}$	transition with a fixed probability

by d^*. Large values of d^* form large and sparse lattices of robots and small values form small and dense ones. In our experiments, the target distance between robots is tuned by the automatic design process in the range $d^* \in (0.07, 0.25)$ m.

We chose the Lennard-Jones model to develop FORMATION because it requires only a minimal set of sensors and actuators to be implemented; and relies solely on local interactions. We expect that the automatic design process will select this module and fine-tune the target distance between robots to perform missions with spatial distribution constraints.

Automatic design process

`Mate` produces control software using the automatic design process described in `TuttiFrutti`—see Chapter 3. In `Mate`, the probabilistic finite-state machine is also restricted to a maximum of four states—the low-level behaviors—and four outgoing edges per state—the transition conditions. Transitions always occur between different states, and self-transitions are not permitted. The modules and their parameters are selected off-line through an optimization process conducted with Iterated F-race. The design process is conducted with simulations performed in ARGoS3, version beta 48, together with the argos3-epuck library. The duration of the optimization process is determined by a predefined simulations budget. The design process ends when the budget is exhausted and `Mate` returns the best configuration found. This configuration is then uploaded to the physical robots

and evaluated in the target environment.

5.1.2 Experimental setup

Missions

We conduct experiments with twenty physical e-pucks that must perform missions that impose constraints on how the swarm self-distributes in the environment. We experiment with `Mate` on three missions: ANY-POINT CLOSENESS, NETWORKED COVERAGE, and CONDITIONAL COVERAGE. The first two are variations of missions previously proposed by Francesca et al. (2015), and CONDITIONAL COVERAGE is a mission we conceived on the basis of a collective decision-making mission introduced by Hasselmann and Birattari (2020).

The time available to the robots to perform a mission is always $T = 120\,\text{s}$. In the three missions, the e-pucks operate in a dodecagonal arena of about $4.9\,\text{m}^2$ surrounded by walls. The floor is gray and might contain black and white regions. The presence of black and/or white regions is specified on a per-mission basis and they denote the *target region* where robots must operate while exhibiting spatially-organizing behaviors. Figure 5.1 shows the arenas for the three missions.

ANY-POINT CLOSENESS: The robots must uniformly cover a *target region* in the arena. The target region is a black square area of $1\,\text{m}^2$. At the beginning of each run, the robots are positioned in the right side of the arena. Figure 5.1 (left) shows the arena for ANY-POINT CLOSENESS.

The score of the swarm is determined by the expected distance from any point in the target region to its closest robot:

$$f_{\text{AC.M}} = E[d(T)]^2, \tag{5.1}$$

which must be minimized. $E[d(T)]$ is the expected distance, at time T, between a generic point within the target region and its closest robot inside the target region. We estimate/approximate $E[d(T)]$ by computing the average distance between *any point* within the target region (in practice, a sufficiently large set of randomly sampled points) and its closest robot inside the target region. More precisely,

$$E[d(T)] \approx \sum_{p=1}^{P} min_i(d_{ip})/P,$$

where $p \in [1, 1000]$ is a point in a set of points randomly and uniformly sampled

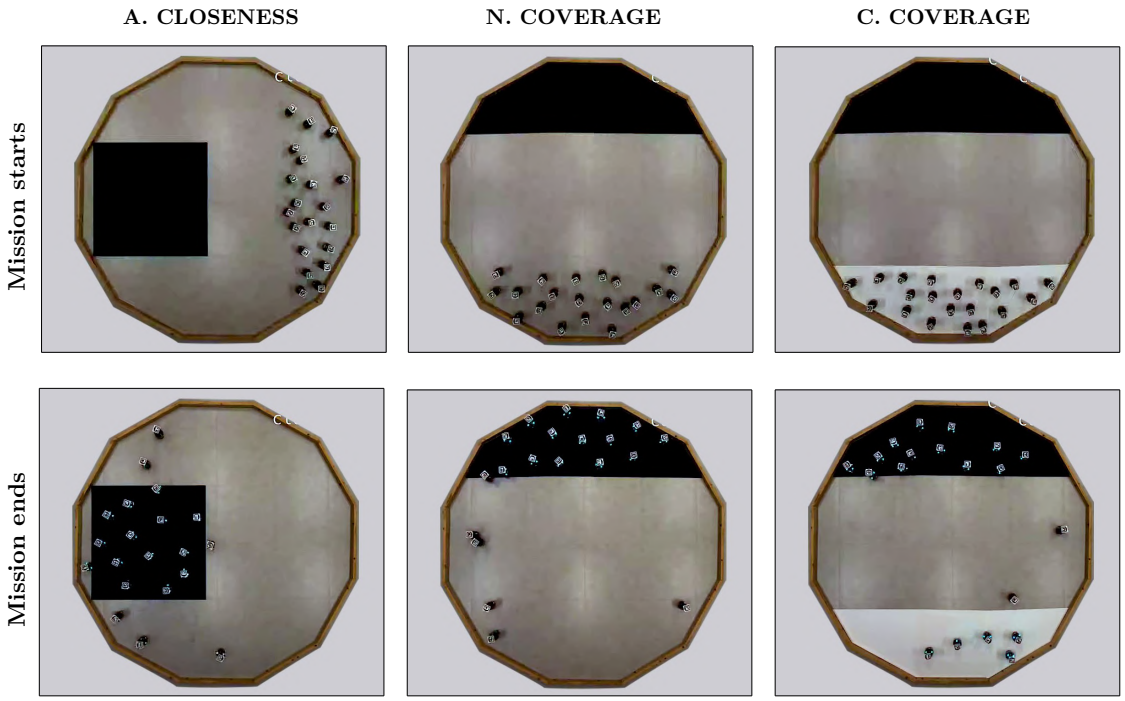


Figure 5.1: Experimental arenas for `Mate`'s experiments. The figure shows the arenas used for the three missions in the study. The top row displays the initial positions of the robots at the start of an experiment. The bottom row shows their positions at the end, after performing the mission using control software generated by `Mate`.

from within the target region; and $i \in [1, 20]$ is a robot in the set of robots that are inside of the target region at time T. The distance d_{ip} is therefore the distance between a random point p and a robot i that are both within the target region. We selected this mission because it challenges `Mate` to design collective behaviors in which robots identify the target region and engage with their peers to cover it uniformly.

NETWORKED COVERAGE The robots must establish a network of robots that covers a target region in the arena. The *target region* is a black area of about $1\,\mathrm{m}^2$. The coverage range for each robot is smaller than the distance at which the robot can establish a network. We consider that each robot covers a circular area of 0.15 m radius. When robots are in the target region, they establish a network if they are at 0.3 m from each other. The area covered by the swarm is the sum of the area covered by all the robots. At the beginning of each run, the robots are randomly positioned on the bottom side of the arena. Figure 5.1 (center) shows the arena for NETWORKED COVERAGE.

The score of the swarm is determined by the ratio between the area covered by

the robots in the network and the total area of the target region:

$$f_{\text{NC.M}} = \sum_{t=0}^{T} A_{N_B}(t)/A_{TR}, \tag{5.2}$$

which must be maximized. $A_{N_B}(t)$ is the area of the target region that is covered by the network of robots N_B, and $A_{TR} = 1\,\text{m}^2$ is the total area of the target region. N_B is the network of robots with the largest number of individuals within the target region. The performance is measured at every time step ($\Delta t = 0.1\,\text{s}$). We selected this mission because it grows in complexity with respect to ANY-POINT CLOSENESS. It challenges `Mate` to design collective behaviors in which the swarm establishes a network of robots that maintain an arbitrary distance between themselves, while covering the target region.

CONDITIONAL COVERAGE: The robots must selectively cover one out of two target regions in the arena. The target regions are a black and a white area of about $1\,\text{m}^2$. The target region to cover is conditioned at each experimental run by the starting position of the robots: if the robots start the experiment in the black area, they must travel to the opposite side of the arena and cover the white one; conversely, if robots start the experiment in the white area, they must travel to the opposite side and cover the black one. Each robot covers a circular area of 0.15 m radius. Figure 5.1 (right) shows the arena for CONDITIONAL COVERAGE.

The score of the swarm is the ratio between the area of the target region they cover and its total area:

$$f_{\text{CC.M}} = \sum_{t=0}^{T} A_m(t)/A_{TR}, \tag{5.3}$$

which must be maximized. $A_{N_B}(t)$ is the area of the target region that is covered by the network of robots N_B, and $A_{TR} = 1\,\text{m}^2$ is the total area of the target region. The performance is measured at every time step ($\Delta t = 0.1\,\text{s}$). We selected this mission because it challenges `Mate` to design collective behaviors in which the robots identify the region to be covered by reacting to the initial conditions of the mission—which differ from one execution to the other.

Baseline methods

`Chocolate`: we use `Chocolate` as originally implemented by Francesca et al. (2015)—see Chapter 2. `Chocolate` operates unmodified on the basis of the reference model RM 1.1 of the e-puck, described in Table 2.1. In principle, `Chocolate` has the low-level behaviors necessary for robots to maintain a constant relative distance.

To design such behavior, `Chocolate` could potentially produce a fine-tuned finite-state machine that constantly transitions between ATTRACTION and REPULSION—ultimately keeping the robots at a constant distance. This behavior is analogous to that implemented in FORMATION using the Lennard-Jones model, which is based on the balance of attraction and repulsion forces.

`EvoSpace`: we use a neuroevolutionary method that integrates the virtual physics model of `Mate`. Likewise `Mate` extends `Chocolate`, `EvoSpace` extends `EvoStick`—see Chapter 2. `EvoSpace` is a method conceived to produce control software for the reference model RM 3.1 of the e-puck. The artificial neural network in `EvoSpace` includes input information obtained from the Lennard-Jones model embedded in `Mate`'s FORMATION. `EvoSpace` tunes the synaptic weights of the neural network via artificial evolution—using the evolutionary algorithm implemented for `EvoStick`. Likewise `Mate`, `EvoSpace` optimizes the neural network until a predefined budged of simulation runs is exhausted. We expected that `EvoSpace` could, in principle, design control software that benefits from this input to produce spatially-organizing behaviors similar to those expected from `Mate`. Table 5.3 summarizes the topology of the neural network and the parameters used in the evolutionary process.

Protocol

For each mission, we conduct 10 independent design processes with `Mate`, `Chocolate`, and `EvoSpace`. This results in 90 instances of control software—10 per method and mission. The design methods are given a budget of 200 000 simulation runs to produce each instance of control software. We evaluated the effectiveness of the methods by testing each instance of control software once with physical robots. The performance of the swarm is computed using ARGoS3, with information on the physical robots provided by a tracking system (Stranieri et al. 2013).

Statistics: We use box-plots to present the performance of the instances of control software obtained in the experiments. In all cases, comparative statements are supported with a Wilcoxon paired rank sum tests, at 95 % confidence (Conover 1999). In addition, we present the results of Friedman test that aggregates the overall performance of the methods across the three missions—see also Chapter 4.

5.1.3 Results

Figure 5.2 shows the performance of each method across the three missions. Demonstration videos of the behavior of the robots are provided in the Supplementary

Table 5.3: Neural network topology and parameters of the evolutionary process in `EvoSpace`. The neural network operates according to RM 3.1, see Table 5.1.

Architecture	
Fully-connected feed-forward neural network without hidden layers	
Input node	**Description**
$in_{a\in\{1,\ldots,4\}}$	scalar projections of aggregated vector from $prox_{i\in\{1,\ldots,8\}}$
$in_{a\in\{5,\ldots,8\}}$	scalar projections of aggregated vector from $light_{i\in\{1,\ldots,8\}}$
$in_{a\in\{9\}}$	aggregated readings of ground sensors $gnd_{j\in\{1,\ldots,3\}}$
$in_{a\in\{10\}}$	value of the density function $z'(n)$
$in_{a\in\{11,\ldots,13\}}$	scalar projections of $V_{c\in\{C\}}$
$in_{a\in\{14\}}$	bias input
Output node	**Description**
$out_{b\in\{1,2\}}$	wheel velocities for $v_{k\in\{l,r\}}$
$out_{b\in\{3,4\}}$	activation cyan color $\{\varnothing, C\}$
Connection	**Description**
$conn_{s\in\{1,\ldots,56\}}$	synaptic connections with weights $\omega\in[-5,5]$
Evolution parameters	
Number of generations *	—
Population size	100
Elite individuals	20
Mutated individuals	80
Evaluations per individual	10
Post-evaluation per individual **	100

* The number of generations is computed according to the budget of simulations.
** The population obtained in the last generation is post-evaluated to select the best individual.

Videos of the dissertation (Garzón Ramos 2025).

Per-mission results

ANY-POINT CLOSENESS: `Mate` performed significantly better than `Chocolate` and `EvoSpace`—see Figure 5.2 (left). As expected, the performance of the swarm is highly conditioned by the ability of the robots to position themselves with respect to their peers. In this mission, the distribution of robots in the target region is more relevant than the number of robots inside it. `Mate` designed collective behaviors that distribute the robots more uniformly than those designed by `Chocolate` and `EvoSpace`. The capability of `Mate` to design such behaviors is reflected in a better performance. Despite that `Chocolate` designed collective behaviors in which more robots reach the target region, the swarm did not distribute uniformly. `EvoSpace`

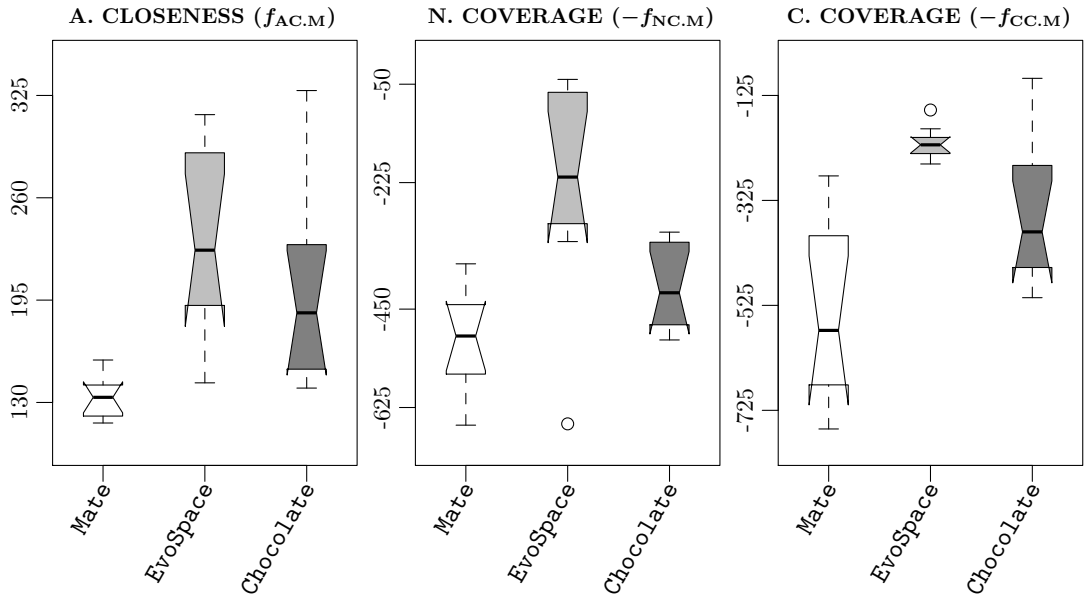


Figure 5.2: Performance obtained in the missions studied with `Mate`. Results per design method are presented with grayscale box-plots, `Mate` (□), `EvoSpace` (▒), `Chocolate` (■). These results have been obtained with physical robots. In all plots, the lower, the better.

designed collective behaviors in which the robots are not very efficient in exploring the arena and do not reach it to cover it satisfactorily.

NETWORKED COVERAGE: Also in this case, `Mate` performed significantly better than `Chocolate` and `EvoSpace`—see Figure 5.2 (center). The ability of the robots to signal each other and remain at a fixed distance with respect to their peers plays an important role in this mission. To cover a large area, the robots should maximize the relative distance between each other. However, they must constrain this distance to avoid losing the connectivity of the network. Robot swarms designed by `Mate` self-organized in the form of repetitive and cohesive lattices of robots—which are effective in the execution of the mission. On the contrary, although the swarms designed by `Chocolate` are capable of establishing small networks, they neither maintain a stable connectivity nor properly cover the target region. They tend to line up on its boundaries. `EvoSpace` mainly designed collective behaviors in which robots explore the arena until they reach the target region, and then remain in place by continuously rotating. The control software produced by `EvoSpace` suffered strong effects from the reality gap and was ineffective when ported to the physical robots.

CONDITIONAL COVERAGE: As in the other two missions, `Mate` performed significantly better than `Chocolate` and `EvoSpace`—see Figure 5.2 (right). FORMA-

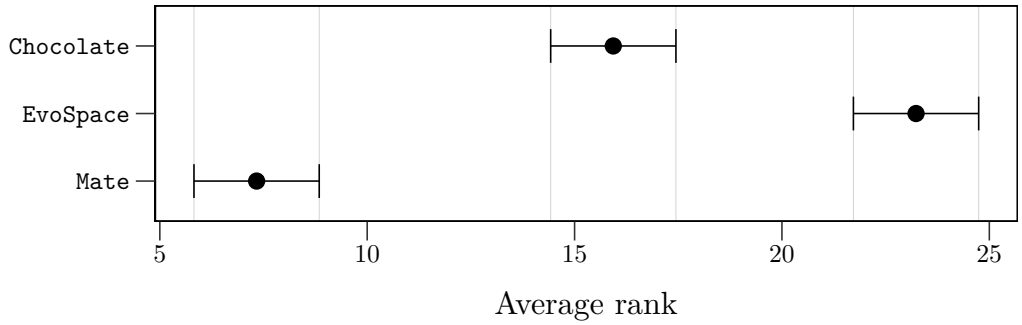


Figure 5.3: Friedman tests with aggregate results from `Mate`'s study. The test considers the results obtained across all missions and provides a relative ranking between the methods. The plot shows the average rank of each method and its 95 % confidence interval. The lower the rank, the better.

TION contributed to designing spatially-organizing behaviors in which the robots efficiently cover the target region. On the other hand, the performance of `Chocolate` and `EvoSpace` was strongly affected because the control software produced relies on the individualistic behaviors. In these behaviors, robots reach the target region but do not consider the presence of their peers to position themselves. Therefore, they are unable to achieve a meaningful spatial distribution. Compared with the other two missions, CONDITIONAL COVERAGE turned to be more challenging for the design methods. This is probably due to the decision-making component of the mission. The number of robots that remain outside the target region at the end of the experimental run increased for the three methods with respect to the other two missions. The highest increase was observed in the results obtained by `Chocolate`, and the lowest increment was observed in the results obtained by `Mate`.

Aggregate results

Figure 5.3 shows the aggregate results of the experiments with physical robots across the three missions. The plot represents the average rank of the three methods and their 95% confidence interval. In these experiments, `Mate` performed significantly better than `Chocolate` and `EvoSpace`. As previously described, `Mate` designed collective behaviors that achieved a better spatial distribution of robots than the baseline methods. The class of missions that we studied is highly dependent on this ability, and therefore `Mate` outranked the two baselines.

5.1.4 Discussion

We evaluated `Mate` in missions in which the robots must operate while taking into account static environmental cues (black and white regions) and the relative distances they maintain from one another. The control software designed by `Mate` proved effective in the three missions. In ANY-POINT CLOSENESS, the robots uniformly covered an indicated target region in the arena. In NETWORKED COVERAGE, the robots established coverage networks while maintaining connectivity at a fixed distance. And in CONDITIONAL COVERAGE, `Mate` designed collective behaviors that allowed the swarm to selectively cover one out of two possible target regions. In all these cases, the signaling protocol embedded in FORMATION allowed the robots to selectively establish spatially-organizing behaviors to perform the mission at hand.

Initially, we expected that `Chocolate` and `EvoSpace` would be able to address the same missions as `Mate`. `Chocolate` includes behaviors modules that could emulate the effects of FORMATION. However, the collective behaviors designed by `Chocolate` did not exhibit the spatial distribution properties observed in those designed by `Mate`. In a similar way, `EvoSpace` incorporates the model of virtual physics that we used to conceive FORMATION. However, we did not notice any meaningful use of this information in the behavior of the robots. We argue that the inability of `Chocolate` and `EvoSpace` to design behaviors in which robots maintain specific relative positions with respect to each other translated into lower performance.

The results of this study highlight limitations of existent automatic methods for the design of spatially-organizing behaviors for robot swarms. This design problem had been mostly addressed with neuroevolution in the past (e.g., in pattern formation and flocking). However, neuroevolutionary methods are prone to suffer strong effects from the reality gap—as shown in our experiments too. Moreover, other alternative approaches like AutoMoDe do not provide yet the means to design complex spatially-organizing behaviors with particular robot positioning schemes. For example, existing AutoMoDe methods (including `Mate`) cannot produce control software for the organization of robot swarms in complex patterns or chains of robots. With the introduction of `Mate`, we approach the effective realization of this type of collective behavior. Here, we focused on demonstrating that the addition of a single module, FORMATION, allows AutoMoDe to effectively tackle missions that `Chocolate` could not. We believe that this idea can be extended to develop new modules for the design of more complex spatially-organizing behaviors.

5.2 Automatic design of stigmergy-based behaviors

In swarm robotics, collective behaviors based on stigmergy have traditionally been designed manually. This process is time-consuming, costly, difficult to replicate, and heavily dependent on the designer's expertise. Here, we show that stigmergy-based behaviors can be automatically designed using AutoMoDe. We demonstrate this ability with a group of robots capable of laying and sensing artificial pheromones. We introduce `Habanero`, a method from the AutoMoDe family specialized in the automatic design of stigmergy-based behaviors for robot swarms. The e-puck robots used in this study are equipped with `Phormica`, a hardware module that lays artificial pheromone trails by focusing UV light onto a floor coated with photochromic material—see Chapter 2. When exposed to UV light, the floor changes from white to magenta, and after the UV light is removed, the floor returns to its original color in about 50 second. These trails can be detected by other robots using their omnidirectional cameras. The artificial pheromones act as color signals that mark the environment and allow robots to communicate information within the swarm.

`Habanero` builds on `TuttiFrutti`, adopting its capabilities to produce control software for robots that perceive and respond to color signals, which in this case allows robots to detect and react to pheromone trails. The main difference between `TuttiFrutti` and `Habanero` is that `Habanero` incorporates the hardware and software specifically designed to lay and detect artificial pheromones. In this study, we demonstrate `Habanero` by generating control software for a set of missions in which robots must rely on stigmergy-based coordination. To assess the quality of the control software produced by `Habanero`, we compare its performance against three baseline methods: (i) control software generated through neuroevolution, (ii) control software manually developed by human designers, and (iii) a random-walk behavior. We investigate whether `Habanero`, and by extension AutoMoDe, is a viable method for designing pheromone-based stigmergy. As in the early AutoMoDe experiments (Francesca et al. 2015), we also investigate whether AutoMoDe can enable stigmergy-based coordination that outperforms control software developed by human designers. Finally, we explore whether `Habanero` can leverage the signaling capabilities of `Phormica` to create mission-specific coordination strategies, similar to the direct communication strategies observed in experiments with `TuttiFrutti`—see Chapter 3.

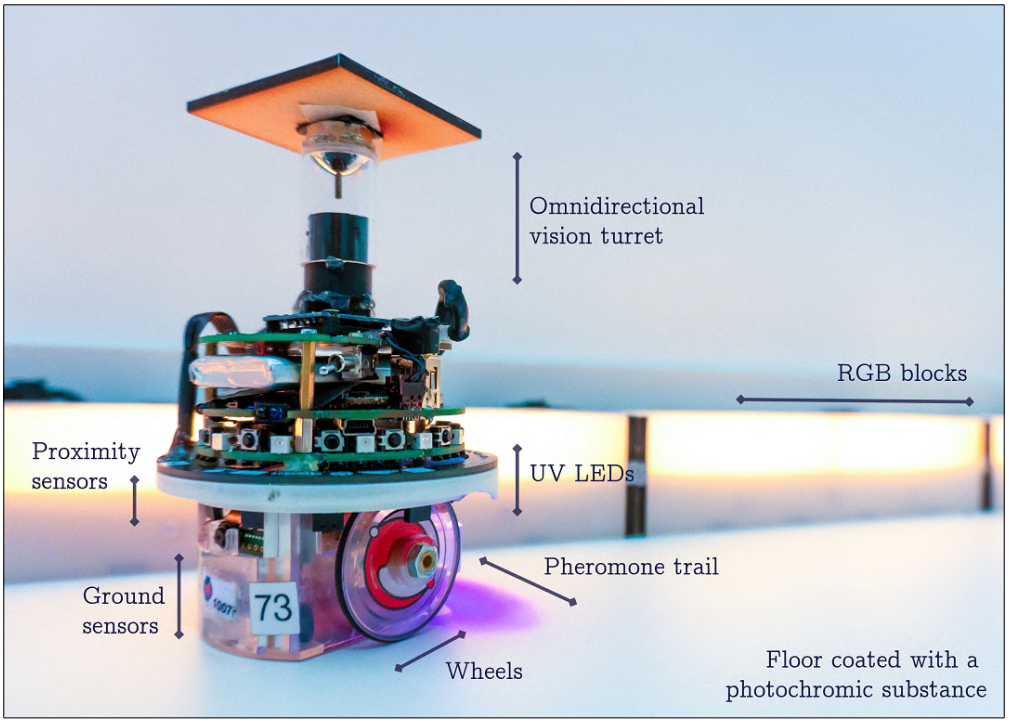


Figure 5.4: Extended version of the e-puck. The picture indicates the set of sensors and actuators defined by RM 4.1. Alongside, we show the RGB blocks and photochromic floor that we used in our experiments with `Habanero`.

5.2.1 AutoMoDe-`Habanero`

Robot platform

`Habanero` produces control software for a version of the e-puck equipped with hardware to lay and detect artificial pheromones, as shown in Figure 5.4. The robot's functional capabilities are adapted from reference model RM 3 to incorporate this new feature, and are formally defined by reference model RM 4.1. In addition to the features introduced in RM 3 (Table 3.1), RM 4.1 allows the control software to control the UV LEDs (*phe*) on the e-puck, enabling it to project pheromone trails of different thickness (*thin*, *thick*) or leave no trail ($\varnothing$). It also provides the ability to adjust the camera's field of view (*fov*), allowing for both narrow and omnidirectional perception ($\frac{1}{12}\pi, 2\pi$). In RM 4.1, we remove the capabilities associated with the LEDs and range-and-bearing board of the e-puck to restrict the robot interactions to those achievable via stigmergy. The reference model RM 4.1 is detailed in Table 5.4.

Table 5.4: The control interface for the e-puck according to the reference model RM 4.1. Robots can perceive: red (R); green (G); blue (B); cyan (C); magenta (M); and yellow (Y). Robots can set the UV LEDs to project: no pheromone trail ($\varnothing$); a thin trail (*thin*); or a thick one (*thick*). V_c is calculated by aggregating the positions of perceived color signals into a unique vector.

Input	**Value**	**Description**
$prox_{i\in\{1,\ldots,8\}}$	$[0, 1]$	reading of proximity sensor i
$gnd_{j\in\{1,\ldots,3\}}$	$\{black, gray, white\}$	reading of ground sensor j
fov	$\{\frac{1}{12}\pi, 2\pi\}$	camera field of view
$cam_{c\in\{R,G,B,C,M,Y\}}$	$\{yes, no\}$	colors perceived
$V_{c\in\{R,G,B,C,M,Y\}}$	$(1.0; [0, 2]\,\pi\,\mathrm{rad})$	their relative aggregate direction
Output	**Value**	**Description**
$v_{k\in\{l,r\}}$	$[-0.12, 0.12]\,\mathrm{m\,s^{-1}}$	target linear wheel velocity
phe	$\{\varnothing, thin, thick\}$	projection of UV lights

Period of the control cycle: 0.1 s.

Modular control architecture

`Habanero` generates control software by fine-tuning and assembling predefined software modules into probabilistic finite-state machines. It adapts the parametric modules originally designed for `TuttiFrutti`—see Table 5.5. `Habanero`'s set of modules includes two independent low-level behaviors and a transition condition that react to pheromone trails: TRAIL-FOLLOWING, TRAIL-ELUSION, and TRAIL-DETECTION. The implementations of these modules are similar to those for COLOR-FOLLOWING, TRAIL-ELUSION, and TRAIL-DETECTION in `TuttiFrutti`. As a novelty, `Habanero`'s modules have parameters for selectively depositing pheromone trails (*phe*) and adjusting the camera's field of view (*fov*)—according to the reference model RM 4.1. `Habanero` also includes a new module, WAGGLE, which enables robots to maintain in-place rotations. The parameters of the software modules are automatically tuned during the design process.

Automatic design process

`Habanero` produces control software using the automatic design process described for `TuttiFrutti`—see Chapter 3. In `Habanero`, the probabilistic finite-state machine is also restricted to a maximum of four states—the low-level behaviors—and four outgoing edges per state—the transition conditions. Transitions always occur between different states, and self-transitions are not permitted. The modules and their parameters are selected off-line through an optimization process conducted with Iterated F-race. The design process is conducted with simulations performed

Table 5.5: `Habanero`'s software modules. The modules are defined on the basis of reference model RM 4.1, see Table 5.4.

Low-level behavior*	**Parameter**	**Description**
EXPLORATION	$\{\tau, phe\}$	movement by random walk
STOP	$\{phe\}$	standstill state
COLOR-FOLLOWING	$\{\delta, phe, fov\}$	steady movement towards robots/objects of color δ
COLOR-ELUSION	$\{\delta, phe, fov\}$	steady movement away from robots/objects of color δ
TRAIL-FOLLOWING	$\{phe, fov\}$	steady movement towards pheromone trails
TRAIL-ELUSION	$\{phe, fov\}$	steady movement away from pheromone trails
WAGGLE	$\{phe\}$	in-place waggle motion
Transition condition	**Parameter**	**Description**
BLACK-FLOOR	$\{\beta\}$	black floor beneath the robot
GRAY-FLOOR	$\{\beta\}$	gray floor beneath the robot
WHITE-FLOOR	$\{\beta\}$	white floor beneath the robot
FIXED-PROBABILITY	$\{\beta\}$	transition with a fixed probability
COLOR-DETECTION	$\{\delta, fov, \beta\}$	robots/objects of color δ perceived
TRAIL-DETECTION	$\{fov, \beta\}$	pheromone trail perceived

* All low-level behaviors can set the UV LEDs (phe) alongside the action described. Modules that use the camera can set its field of view (fov) alongside the action described.

in ARGoS3, version beta 48, together with original libraries to simulate the e-puck according to RM 4.1 and also the photochromic floor on which the robots operate. The duration of the optimization process is determined by a predefined simulations budget. The design process ends when the budget is exhausted and `Habanero` returns the best configuration found. This configuration is then uploaded to the physical robots and evaluated in the target environment.

5.2.2 Experimental setup

Missions

We conduct experiments with eight physical e-pucks that must perform missions that impose constraints on how the swarm self-distributes in the environment. We experiment with `Habanero` on four missions: STOP, AGGREGATION, DECISION MAKING, and RENDEZVOUS POINT. STOP, DECISION MAKING, and RENDEZVOUS POINT are adaptations of missions conducted in `TuttiFrutti`. AGGREGATION is a common realization of an aggregation behavior.

The time available to the robots to perform a mission is always $T = 180\,\mathrm{s}$. In the three missions, the e-pucks operate in a rectangular arena of about $1.8\,\mathrm{m}^2$ surrounded by `MoCA`'s RGB blocks. The floor of the arena is white and has been coated with a photochromic material that acts as a medium to lay the pheromone

trails (Salman et al. 2020). The photochromic material turns magenta when exposed to UV light. Once the UV light is removed, the magenta color gradually fades and the floor returns to white in about 50 s. Figure 5.5 shows the arenas for the four missions.

STOP: The robots must move until one of the walls that surrounds the arena emits a stop signal by turning blue. Once the wall turns blue, all robots in the swarm must stop moving as soon as possible. The wall that emits the stop signal is randomly selected. At the beginning of each run, the robots are randomly positioned in the arena. Figure 5.5 (top-left) shows the arena for STOP.

The score of the swarm is determined by time during which the robots do not perform the intended behavior, before and after the stop signal:

$$f_{\text{ST.H}} = \sum_{t=1}^{\bar{t}} \sum_{i=1}^{N} \bar{I}_i(t) + \sum_{t=\bar{t}+1}^{T} \sum_{i=1}^{N} I_i(t), \tag{5.4}$$

which must be minimized. N and T represent the number of robots and the duration of the mission, respectively. The time at which the stop signal is displayed is represented by $\bar{t}$. The value of $\bar{t}$ is uniformly sampled between $(70, 90)$ s. The indicators $I_i(t)$ and $\bar{I}_i(t)$ are defined as:

$$I_i(t) = \begin{cases} 1, \text{ if robot } i \text{ is moving at time } t; \\ 0, \text{ otherwise;} \end{cases} \qquad \bar{I}_i(t) = 1 - I_i(t).$$

We selected this mission because it challenges `Habanero` to design collective behaviors with event-handling capabilities by relying on pheromone-based stigmergy, similar to the behaviors observed with `TuttiFrutti`.

AGGREGATION: The robots must approach one another to form a cluster and remain close until the end of the mission. At the beginning of each run, the robots are randomly positioned in the arena. Figure 5.5 (top-right) shows the arena for AGGREGATION.

The score of the swarm is the average distance from a robot to any other robot:

$$f_{\text{AG.H}} = \sum_{t=1}^{T} d_{\text{avg}}(t), \tag{5.5}$$

which must be minimized. At each time step t, the average distance d_{avg} between the robots is added to $f_{\text{AG.H}}$. The performance is measured at every time step ($\Delta t = 0.1$ s). We selected this mission because it challenges `Habanero` to design

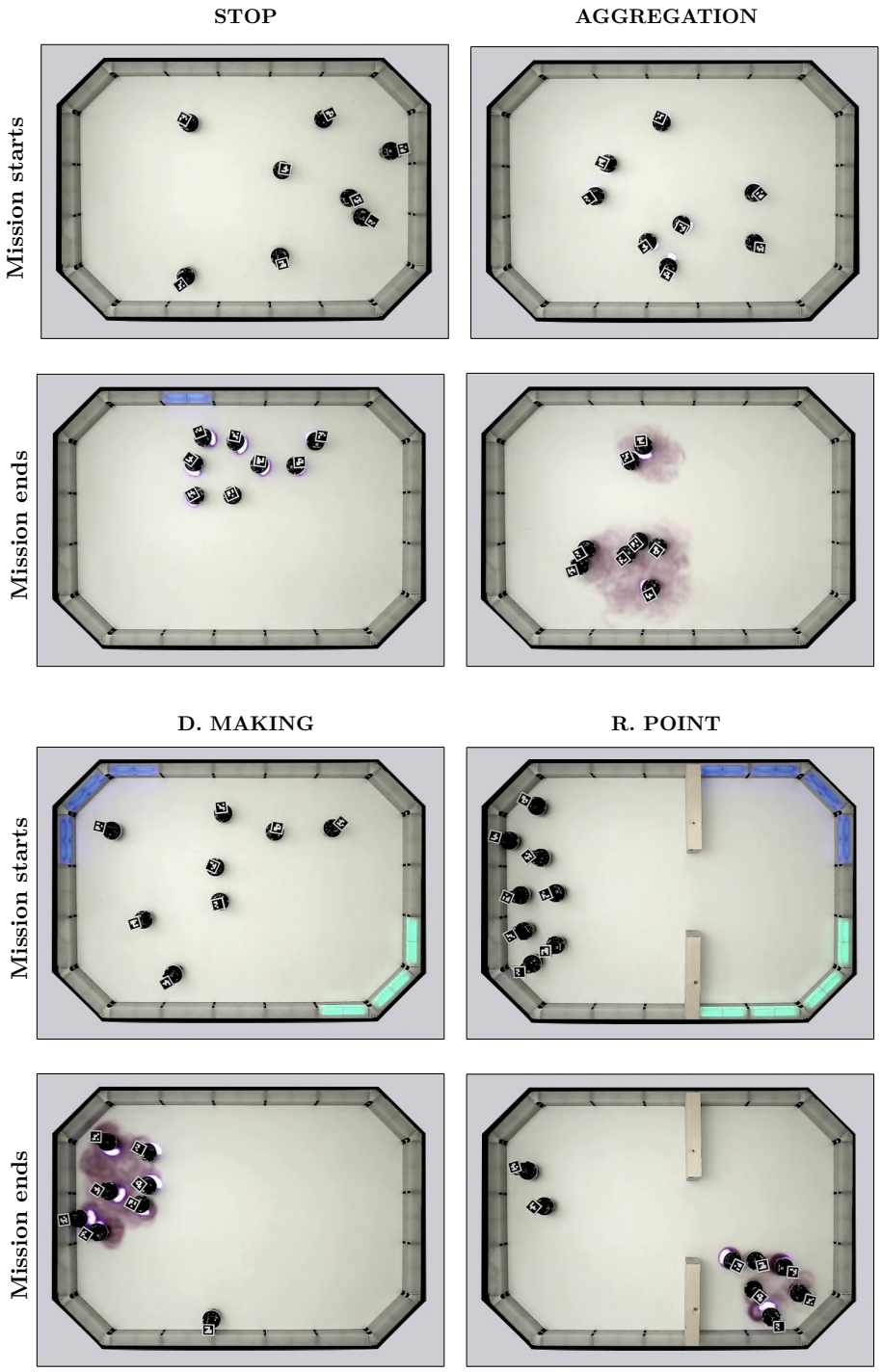


Figure 5.5: Experimental arenas for `Habanero`'s experiments. The figure shows the arenas used for the four missions in the study. For each mission, the top row displays the initial positions of the robots at the start of an experiment. The bottom row shows their positions at the end, after performing the mission using control software generated by `Habanero`. The RGB blocks in the arena are configured on a per-mission basis. The floor of the arena is coated with a photochromic substance that enable the robots to lay artificial pheromones.

collective behaviors in which robots must communicate to maintain a specific spatial distribution.

DECISION MAKING: The robots must select between a green and a blue region, signaled to the robots by RGB blocks that display blue and green colors. The two regions are squared portions of the arena located on its corners. The green and blue signals disappear after a random amount of time, which is uniformly sampled between 70 and 90 s. Figure 5.5 (bottom-left) shows the arena for DECISION MAKING.

At every time step t, the score increases by +1 for every robot that is in the green region, and by +2 for every robot that is in the blue one. The score is the total of points accumulated by the end of the mission:

$$f_{\text{DM.H}} = \sum_{t=1}^{T} \sum_{i=1}^{N} I_i(t), \tag{5.6}$$

which must be maximized. The indicator $I_i(t)$ is defined as:

$$I_i(t) = \begin{cases} 1 & \text{if robot } i \text{ is in green region,} \\ 2 & \text{if robot } i \text{ is in blue region,} \\ 0 & \text{otherwise.} \end{cases}$$

We selected this mission because it challenges `Habanero` to identify and select among environmental features that are relevant to the mission—as investigated with `TuttiFrutti`.

RENDEZVOUS POINT: The robots must reach the green region and stay there until the end of the mission. A blue region is added as a decoy to possibly confuse the robots. Like in the past mission, the two regions are squared portions of the arena. A wall with a narrow gate laterally divides the arena into two sections: the left side, where the robots are deployed at the beginning of the experiment; and the right side, which contains the green and blue regions. Both green and blue signals disappear after a random amount of time, which is uniformly sampled between 70 and 90 s. Figure 5.5 (bottom-left) shows the arena for DECISION MAKING.

The score is determined by the number of robots inside the green region at the end of the mission:

$$f_{\text{DM.H}} = K_{\text{in}} - K_{\text{out}}, \tag{5.7}$$

which must be maximized. K_{in} is the number of robots inside the green region

at the end of the mission, and K_{out} is the number of robots outside. We selected this mission because it challenges `Habanero` to design collective behaviors in which robots use the pheromones to navigate their environment—a behavior observed in `TuttiFrutti`.

Baseline methods

`EvoPheromone`: we use a neuroevolutionary method that can produce control software for the e-puck as formally described by the reference model RM 4.1—the same as `Habanero`. Like `EvoColor` and `EvoSpace`, `EvoPheromone` also builds on `EvoStick`—see Chapter 2. `EvoSpace` tunes the synaptic weights of the neural network via artificial evolution using the evolutionary algorithm implemented for `EvoStick`. `EvoPheromone` optimizes the neural network until a predefined budged of simulation runs is exhausted. `EvoPheromone` has access to the same capabilities as `Habanero`, and therefore we expect it to be able to design pheromone-based stigmergy behaviors for the robots. Table 5.6 summarizes the topology of the neural network and the parameters used in the evolutionary process.

`C-Human`: we follow the protocol developed by Francesca et al. (2015) to compare automatically generated control software with that obtained by manual design. In these experiments, 10 human designers were requested to produce control software using the software modules of `Habanero`. In a sense, a human designer acts as an optimization agent that assembles a finite-state machine and fine-tunes its parameters. `C-Human` produces control software for the e-puck formally described by reference model RM 4.1—the same as `Habanero`. The human designers who participated in this study had various levels of expertise in swarm robotics—ranging from bachelor students to post-doctoral researchers in swarm robotics. We provided the designers with a visualization tool to produce and manipulate finite-state machines, to visualize simulations, and to compute the value of the objective function (Kuckling et al. 2021a). All simulations were executed in ARGoS3 with special libraries to simulate the pheromone module and the photochromatic floor—the same as `Habanero`. The designers were allotted 4 hours per mission—as originally devised by Francesca et al. (2015).

`R-Walk`: although not an automatic design method, we include a random-walk behavior in the study as a lower bound on the performance of robot swarms. In `R-Walk`, the robots move in a straight line in the arena, when they encounter an obstacle, they rotate for a random number of control steps and then resume their straight motion. `Random-Walk` is also conceived for the e-puck formally described

Table 5.6: Neural network topology and parameters of the evolutionary process in `EvoPheromone`. The neural network operates according to RM 4.1, see Table 5.4.

Architecture	
Fully-connected feed-forward neural network without hidden layers	
Input node	**Description**
$in_{a\in\{1,\dots,8\}}$	readings of proximity sensors $prox_{i\in\{1,\dots,8\}}$
$in_{a\in\{9,\dots,11\}}$	readings of ground sensors $gnd_{j\in\{1,\dots,3\}}$
$in_{a\in\{12,\dots,35\}}$	scalar projections of $V_{c\in\{R,G,B,C,M,Y\}}$ with $fov_{\{\frac{1}{12}\pi\}}$
$in_{a\in\{36,\dots,59\}}$	scalar projections of $V_{c\in\{R,G,B,C,M,Y\}}$ with $fov_{\{2\pi\}}$
$in_{a\in\{60\}}$	bias input
Output node	**Description**
$out_{b\in\{1,\dots,4\}}$	tuples v' to map each velocity in the set $v_{k\in\{l,r\}}$
$out_{b\in\{5,\dots,7\}}$	activation of UV LEDs with $\{\varnothing, thin, thick\}$
Connection	**Description**
$conn_{s\in\{1,\dots,420\}}$	synaptic connections with weights $\omega\in[-5,5]$
Evolution parameters	—
Number of generations *	—
Population size	100
Elite individuals	20
Mutated individuals	80
Evaluations per individual	10
Post-evaluation per individual **	100

* The number of generations is computed according to the budget of simulations.
** The population obtained in the last generation is post-evaluated to select the best individual.

by the reference model RM 4.1.

Protocol

For each mission, we conduct 10 independent design processes with `Habanero`, `EvoPheromone`, and `C-Human`. This results in 120 instances of control software—10 per method and mission. We also include 10 runs of `R-Walk` per mission. This amounts a total of 160 observations in the experiments. The design methods are given a budget of 100 000 simulation runs to produce each instance of control software. We evaluate the effectiveness of the methods by testing each instance of control software once with physical robots. The performance of the swarm is computed using ARGoS3, with information of the physical robots provided by a tracking system (Legarda Herranz et al. 2022).

Statistics: We use box-plots to present the performance of the instances of control software obtained in the experiments. In all cases, comparative statements are supported with a Wilcoxon paired rank sum tests, at 95 % confidence (Conover 1999). In addition, we present the results of Friedman test that aggregates the overall performance of the methods across the four missions—see Chapter 4.

5.2.3 Results

Figure 5.6 shows the performance of each method across the four missions. Demonstration videos of the behavior of the robots are provided in the Supplementary Videos of the dissertation (Garzón Ramos 2025).

Per-mission results

STOP: `Habanero` and `C-Human` performed similarly well, and both performed significantly better than `EvoPheromone` and `R-Walk`—see Figure 5.6 (top-left). In this mission, the robots must halt and stand still as soon as a stop signal is perceived. Unlike experiments with `TuttiFrutti`, the e-pucks in this study are incapable of direct communication. Therefore, the robots that detect the signal can only rely on stigmergy to alert any peers that are in a position from which the signal cannot be seen. `Habanero` designed collective behaviors in which robots detect the signal, waggle in place, and lay pheromone trails to alert their peers. Other robots mimic this behavior and propagate the alert in the swarm. The collective behaviors produced by `EvoPheromone` did not accomplish the mission in its true sense. The robots took advantage of stigmergy to gradually repel each other, approach the walls, and eventually stop against them. No reaction is perceived with respect to the stop signal—as observed with `EvoColor`. `C-Human` produced collective behaviors similar to those generated by `Habanero`.

AGGREGATION: `Habanero` performed significantly better than all other design methods—see Figure 5.6 (top-right). To aggregate, the robots cannot rely on any form of direct communication or on the ability to directly sense the presence of their peers in their vicinity. They must leverage the pheromone trails to attract their peers and aggregate using stigmergy. `Habanero` produced collective behaviors in which the robots lay pheromone trails only for short periods of time and keep searching the environment for pheromone traces left by their peers. By laying pheromone trails intermittently, the robots avoid saturating the environment and mark only isolated spots, which then serve as aggregation points. `EvoPheromone` designed robots that lay pheromone trails while moving along a circular trajectory

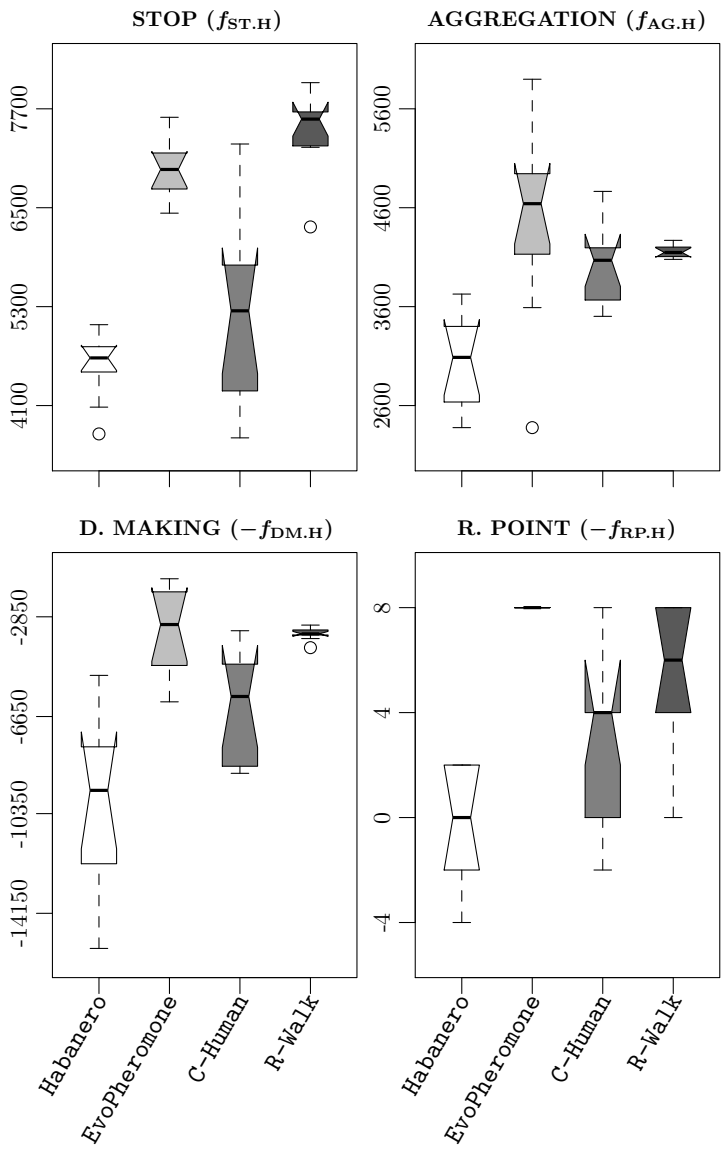


Figure 5.6: Performance obtained in the missions studied with `Habanero`. Results per design method are presented with grayscale box-plots, `Habanero` (□), `EvoPheromone` (▒), `C-Human` (▒), `R-Walk` (■). These results have been obtained with physical robots. In all plots, the lower, the better.

and gather at places with high pheromone concentration. However, this behavior did not translate well to the physical robots. The control software produced by `C-Human` continuously laid pheromone trails with the expectation that all robots would gather at one place. This caused the robots to remain trapped in local pheromone accumulations.

DECISION MAKING: Also in this case, `Habanero` performed significantly better than all other methods—see Figure 5.6 (bottom-left). In this mission, the robots must take the decision to aggregate in either the blue or green region. Halfway through each run of the experiment, the blue and green RGB blocks are switched off, leaving the robots without any visual cue to identify the two regions. In order to maximize the score, the robots must quickly aggregate in the region that provides the highest score per time step—i.e., the blue one—and remain there even once the environmental cues are removed. In all experimental runs, the robot swarm designed by `Habanero` correctly selected the blue region to aggregate. The robots relied on the pheromone trails not only to attract other robots to the blue region, but also to stay there after the signals are removed. The robot swarms generated by `EvoPheromone` were unable to aggregate in a single region: robots stay in the first region in which they enter without considering the colors. The robot swarm produced by `C-Human` was able to correctly aggregate in the blue region but is unable to remain there once the signals are removed.

RENDEZVOUS POINT: `Habanero` performed significantly better than all other methods—see Figure 5.6 (bottom-right). In this mission, the robots must cross the narrow gate to gather in the green region. As in DECISION MAKING, the blue and green RGB blocks are also switched off halfway through each run of the experiment. The robots therefore are required to leverage the pheromone trails to collectively navigate the environment towards the green region and remain there, before the environmental signals are removed. The robot swarms designed by `Habanero` rely on random walk to cross the gate and find the green region. Once there, the robots lay pheromone trails to mark the place, attract their peers, and remain there when the green light is removed. `EvoPheromone` designed collective behaviors in which robots move along the walls of the arena to eventually cross the gate and reach the green region. This behavior, also observed in `EvoColor`, fails when transferred to physical robots and causes them to remain stuck in the walls. The robot swarm produced by `C-Human` was able to correctly aggregate in the green region but is unable to remain there once the cues are removed.

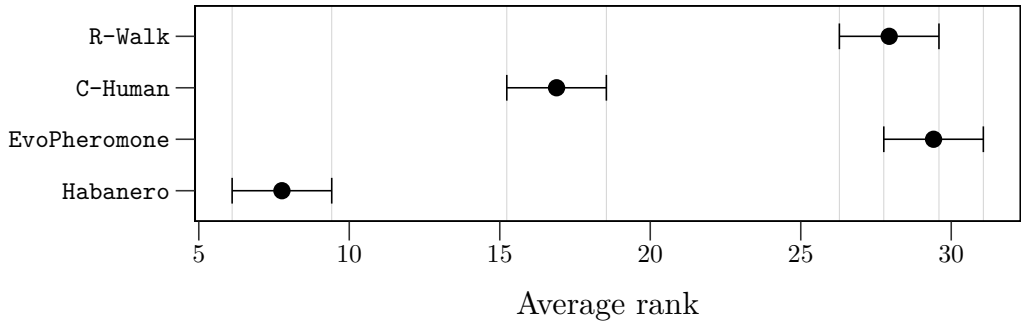


Figure 5.7: Friedman tests with aggregate results from `Habanero`'s study. The test considers the results obtained across all missions and provides a relative ranking between the methods. The plot shows the average rank of each method and its 95 % confidence interval. The lower the rank, the better.

Aggregate results

Figure 5.7 shows the aggregate results of the experiments with physical robots across the three missions. The plot represents the average rank of the three methods and their 95% confidence interval. In these experiments, `Habanero` performed significantly better than all other methods. `Habanero` successfully used pheromone trails to develop mission-specific interaction strategies for the robots, which remained effective when transferred to physical robots. The ability of `Habanero` to exploit this signaling mechanism gave the method a performance advantage over the baseline methods considered, and therefore it outranked them all.

5.2.4 Discussion

As shown in Chapters 3 and 4, automatic design can ease the realization of robot swarms across different missions, while minimizing human intervention. The experiments presented in this study show that this holds true also in the case of robot swarms that rely on pheromone-based stigmergy. Indeed, `Habanero` automatically designed stigmergy-based collective behaviors that were effective across all missions considered. For each mission, it found appropriate ways to use the pheromones effectively. As with `TuttiFrutti`, the interaction strategies generated by `Habanero` were tailored to each mission and varied from one to another. In these interaction strategies, the limited perception and computation capabilities of the individual robots were compensated at the swarm level by exploiting pheromone-based stigmergy.

The e-puck used in the experiments, as a single robot, has limited spatial coordination, memory, and communication abilities. However, spatial organizations, memory, and communication emerged at the collective level thanks to the pheromone-based stigmergy. Spatial organization: In AGGREGATION, DECISION MAKING, and RENDEZVOUS POINT, the e-pucks self-organized and distributed in space driven by their pheromone trails and other environmental signals. Memory: In DECISION MAKING and RENDEZVOUS POINT, the swarm of e-pucks retained relevant information about the past state of the environment by laying pheromone trails. Communication: The semantics of pheromone trails is mission-specific. For example, the pheromone trails that the e-pucks laid in STOP had a meaning (stop where you are) that is radically different from the meaning in AGGREGATION (come here). It is interesting to note that spatial organization, memory, and communication (including the semantics of pheromone trails) were not hand-coded in the modules on which `Habanero` operates: they were the product of the way in which `Habanero` automatically combined these modules on a per-mission basis.

With `Habanero`, we demonstrated that it is possible to generate pheromone-based collective behaviors through an automatic process that is repeatable and generally applicable. We believe this step toward the systematic development of stigmergy-based behaviors can motivate further research into designing more complex behaviors For instance, future work could focus on behaviors that require fine-tuned sensitivity to pheromone concentrations or the ability to simultaneously respond to different types of pheromone trails.

5.3 Automatic design of robot shepherding behaviors

Research on AutoMoDe has focused on missions that the swarm can perform by itself, interacting only with a static environment, and without the presence of other active entities. In this study, we investigate the design of robot swarms that perform missions by interacting with other robots that populate their environment. We frame this design problem into the robot shepherding problem (Lien et al. 2004). In robot shepherding, it is assumed that two groups of robots of different kind operate in the same environment—the shepherds and the sheep. Shepherds and sheep influence each other's behavior and constitute a heterogeneous system that must perform missions collectively. We use AutoMoDe to produce the control software of the shepherds so that they coordinate the sheep in a set of spatially-organizing

missions. The sheep operate with predefined fixed control software. In a sense, the sheep are active entities that populate the environment of the automatically designed swarm of shepherds.

To investigate this problem, we introduce `Pistacchio`: an automatic design method from the AutoMoDe family that we use to design the control software for the shepherds. `Pistacchio` is a simplified version of `TuttiFrutti` that we developed to allow experimentation with multiple swarms in a single environment. We use e-puck robots as both shepherds and sheep, which can interact via color signals. The shepherds can use different color signals to trigger various behaviors in the sheep. The goal of this study is to determine whether an automatic design process can effectively discover and exploit the dynamics between the two robot groups. In these experiments, the behavior of the sheep is a black-box to the design process. The dynamics of the sheep must be discovered while the automatic design process produces the control software of the shepherds.

We use simulations to evaluate `Pistacchio` in nine experimental scenarios that combine shepherding missions with sheep operating under three different predefined behaviors: attraction, repulsion, or a combination of the two. We compare the performance of `Pistacchio` with that of three baseline methods: a neuroevolutionary method, a manual design approach, and a random-walk. Unlike `TuttiFrutti`'s experiments, we focus here on whether AutoMoDe can effectively identify and leverage the dynamics of other active robots, rather than on fixed environmental signals. We expect that the automatic design process will leverage mission-specific signaling protocols, similar to those in `TuttiFrutti`, to allow the swarm of shepherds to coordinate the sheep.

5.3.1 AutoMoDe-`Pistacchio`

Robot platform

`Pistacchio` produces control software for the version of the e-puck introduced with `TuttiFrutti`—see in Figure 3.1. We adapt the functional capabilities defined in the reference model RM 3 to restrict interactions between shepherds and sheep to those triggered by stimuli that are visually perceivable—i.e., the relative distance between robots and the colors they display. In this way, we simplify the visualization and monitoring of the interactions between the two groups of robots. We formalize the capabilities of the e-puck in the reference model RM 3.3. In this reference model, we remove the capabilities associated with the range-and-bearing board of the

Table 5.7: The control interface for the e-puck according to the reference model RM 3.3. Robots can perceive: cyan (C); magenta (M); and yellow (Y). Robots can display no color ($\varnothing$); cyan (C); magenta (M); and yellow (Y). V_c is calculated by aggregating the positions of color signals into a unique vector.

Input	**Value**	**Description**
$prox_{i\in\{1,\ldots,8\}}$	$[0, 1]$	reading of proximity sensor i
$gnd_{j\in\{1,\ldots,3\}}$	$\{black, gray, white\}$	reading of ground sensor j
$cam_{c\in\{C,M,Y\}}$	$\{yes, no\}$	colors perceived
$V_{c\in\{C,M,Y\}}$	$(1.0; [0, 2]\,\pi\,\mathrm{rad})$	their relative aggregate direction
Output	**Value**	**Description**
$v_{k\in\{l,r\}}$	$[-0.12, 0.12]\,\mathrm{m\,s^{-1}}$	target linear wheel velocity
$LEDs$	$\{\varnothing, C, M, Y\}$	color displayed by the LEDs

Period of the control cycle: 0.1 s.

e-puck because these cannot be perceived by visual inspection. We also restrict the colors that e-pucks can perceive to $\delta \in \{C, M, Y\}$. The colors $\delta \in \{R, G, B\}$ are reserved for environmental signals, which are not used in this case. The reference model RM 3.3 is detailed in Table 5.7.

Modular control architecture

`Pistacchio` generates control software by fine-tuning and assembling predefined software modules into probabilistic finite-state machines. It incorporates most of the parametric modules originally designed for `TuttiFrutti` (see Table 5.8) but excludes those related to the e-puck's range-and-bearing board: ATTRACTION, REPULSION, NEIGHBOR-COUNT, and INV-NEIGHBOR-COUNT. As a novelty, `Pistacchio` introduces CIRCLING, a low-level behavior that allows the robot to move in circular patterns with a turning angle defined by the parameter θ. We include this module because we expect it to help shepherds maneuver around the sheep, potentially enclosing them—similar to natural shepherding behaviors. The parameters of the software modules are automatically tuned during the design process.

Automatic design process

`Pistacchio` also produces control software for the shepherds using the automatic design process described for `TuttiFrutti`—see Chapter 3. The probabilistic finite-state machine is restricted to a maximum of four states—the low-level behaviors—and four outgoing edges per state—the transition conditions. Transitions always

Table 5.8: `Pistacchio`'s software modules. The modules are defined on the basis of reference model RM 3.3, see Table 5.7.

Low-level behavior*	**Parameter**	**Description**
EXPLORATION	$\{\tau, \gamma\}$	movement by random walk
STOP	$\{\gamma\}$	standstill state
COLOR-FOLLOWING	$\{\delta, \gamma\}$	steady movement towards robots/objects of color δ
COLOR-ELUSION	$\{\delta, \gamma\}$	steady movement away from robots/objects of color δ
CIRCLING	$\{\theta, \gamma\}$	circular movement with angle θ
Transition condition	**Parameter**	**Description**
BLACK-FLOOR	$\{\beta\}$	black floor beneath the robot
GRAY-FLOOR	$\{\beta\}$	gray floor beneath the robot
WHITE-FLOOR	$\{\beta\}$	white floor beneath the robot
FIXED-PROBABILITY	$\{\beta\}$	transition with a fixed probability
COLOR-DETECTION	$\{\delta, \beta\}$	robots/objects of color δ perceived

* All low-level behaviors display a color $\gamma \in \{\varnothing, C, M, Y\}$ alongside the action described.

occur between different states, and self-transitions are not permitted. The modules and their parameters are selected off-line through an optimization process conducted with Iterated F-race. The design process is conducted with simulations performed in ARGoS3, version beta 48, together with argos3-epuck library. The duration of the optimization process is determined by a predefined simulations budget. The design process ends when the budget is exhausted and `Pistacchio` returns the best configuration found. This configuration is then uploaded to the shepherds and evaluated in the target environment.

5.3.2 Experimental setup

We considered a heterogeneous system of five shepherds and ten sheep, which jointly performed a set of missions. We devised the control software for the sheep so that they do not take action unless stimulated by the shepherds. In this way, the performance of the heterogeneous system strictly depends on the effectiveness of the shepherding behaviors that are designed. We experiment with `Pistacchio` on three missions: AGGREGATION, DISPERSION, and HERDING. For these three missions, we consider three variants of sheep behavior: *C1-Attraction*, *C2-Repulsion*, and *C3-Attraction&Repulsion*. By combining missions and sheep behavior, we produce 9 experimental scenarios on which to evaluate `Pistacchio`.

Sheep control software

Each sheep operated with one out of three predefined instances of control software: *C1-Attraction*, *C2-Repulsion*, and *C3-Attraction&Repulsion*. Shepherds could stimulate the sheep by physical proximity in a 3 cm range, or by displaying colors with their LEDs in a 40 cm range. The ground sensor of the sheep allows them to detect regions of interest in the environment.

In *C1-Attraction*, sheep are attracted to shepherds that display the color magenta. In *C2-Repulsion*, sheep are repelled by shepherds that display the color cyan. In *C3-Attraction&Repulsion*, sheep are both attracted to shepherds that display the color magenta and repelled by shepherds that display the color cyan. In all cases, the sheep display the color yellow and remain static if no stimuli is perceived—physical proximity or color. If a sheep steps into a white-floor region, it halts its movement and turns off its LEDs until the end of the mission.

C1-Attraction, *C2-Repulsion*, and *C3-Attraction&Repulsion* are probabilistic finite-state machines created with software modules that are similar to those of `Pistacchio`. We followed a manual trial-and-error process to design them. These finite-state machines were undisclosed to the design process that generates the behavior of the shepherds, which sees them as a black box.

Missions

The time available to the robots to perform a mission is always $T = 120\,\text{s}$. The robots operate in an octagonal arena of about $2.8\,\text{m}^2$ and gray floor. Figure 5.8 shows the arenas for the three missions.

AGGREGATION: At the beginning of each run, shepherds and sheep are randomly distributed in the arena. The shepherds must group the sheep. Figure 5.8 (left) shows the arena for AGGREGATION.

The score of the swarm is determined by the average distance from each sheep to the center of mass of all sheep at the end of the mission:

$$f_{\text{AG.P}} = \frac{\sum_{i=1}^{10} D_i(T)}{10}, \tag{5.8}$$

which must be minimized. $D_i(T)$ is the distance from a sheep i, at position (x_i, y_i), to the center of mass of all sheep (x_c, y_c) at time T.

DISPERSION: At the beginning of each run, shepherds and sheep are randomly distributed at the center of the arena. The shepherds must separate the sheep. Figure 5.8 (center) shows the arena for DISPERSION.

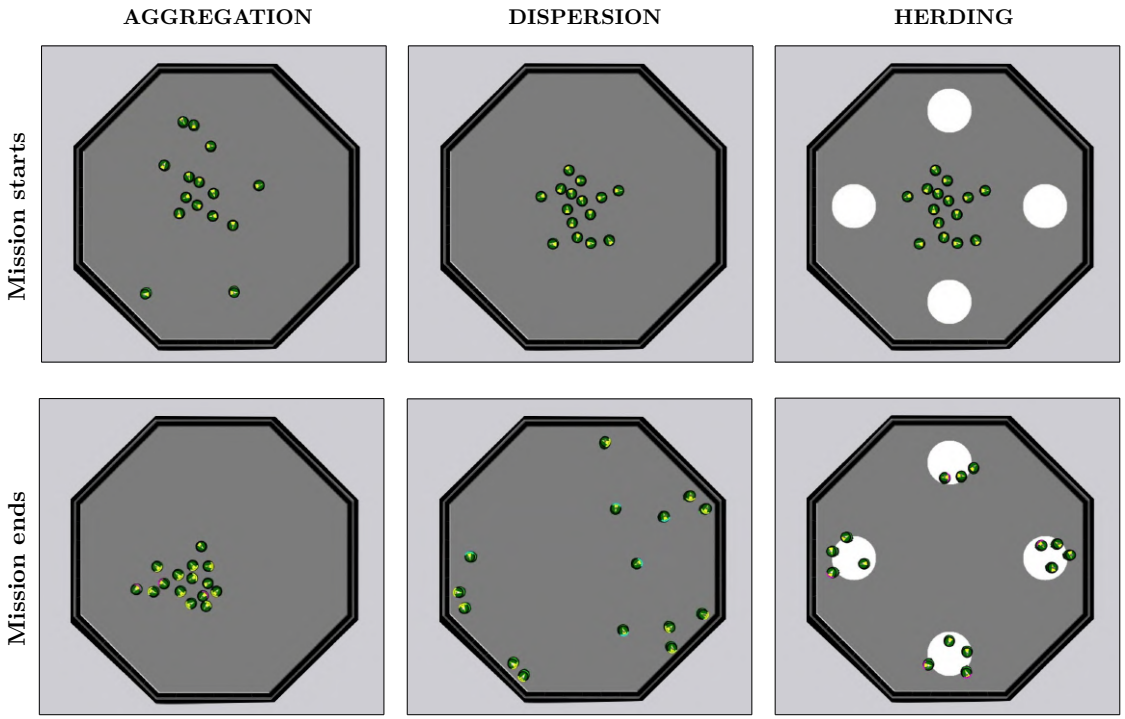


Figure 5.8: Experimental arenas for `Pistacchio`'s experiments. The figure shows the arenas used for the three missions in the study. The top row displays the initial positions of the robots at the start of an experiment. The bottom row shows their positions at the end, after performing the mission using control software generated by `Pistacchio`.

The score of the swarm is also determined by the average distance from each sheep to the center of mass of all sheep at the end of the mission:

$$f_{\text{DS.P}} = \frac{\sum_{i=1}^{10} D_i(T)}{10}, \tag{5.9}$$

which in this case must be maximized. $D_i(T)$ is the distance from a sheep i, at position (x_i, y_i), to the center of mass of all sheep (x_c, y_c) at time T.

HERDING: At the beginning of each run, shepherds and sheep are randomly distributed at the center of the arena. The shepherds must drive the sheep to four indicated locations. The indicated locations are circular white regions of about $0.3\,\text{m}^2$. The four locations are equivalent to each other. Figure 5.8 (right) shows the arena for HERDING.

The score of the swarm is the number of sheep that remain out of the four locations at the end of the mission:

$$f_{\text{DM.H}} = \sum_{t=1}^{T} \sum_{i=1}^{N} I_i(t), \tag{5.10}$$

which must be minimized. T is the duration of the mission.

By pairing sheep control software and missions, we presented varied challenges to the automatic design of the shepherding behaviors. The sheep could be more or less cooperative with the shepherds for the mission at hand. For example, we expect the shepherds to group the sheep more effectively when the sheep operate with *C1-Attraction*, and less effectively when they operate with *C2-Repulsion*. Conversely, we expect the shepherds to separate the sheep more effectively when the sheep operate with *C2-Repulsion*, and less effectively when they operate with *C1-Attraction*. *C3-Attraction&Repulsion* gives more freedom to the automatic design process to select the best performing strategy.

Baseline methods

`EvoCMY`: we use a neuroevolutionary method that can produce control software for the e-puck described by the reference model RM 3.3. `EvoCMY` is a restricted version of `EvoColor`, which we also developed to enable the experimentation with multiple swarms in a single environment—like `Pistacchio`. `EvoCMY` tunes the synaptic weights of the neural network via artificial evolution with the same design process as `EvoColor`—see Chapter 3. `EvoCMY` optimizes the neural network until a predefined budget of simulation runs is exhausted. Table 5.9 summarizes the topology of the neural network and the parameters used in the evolutionary process.

`C-Human`: we also consider here the protocol developed by Francesca et al. (2015) to compare automatic and manual approaches to the design of collective behaviors for robot swarms. In these experiments, 3 human designers were requested to produce control software using `Pistacchio`'s software modules. The human designers are tasked with assembling suitable finite-state machines and fine-tuning their parameters, adopting the role of an optimization agent. `C-Human` produces control software for the e-puck formally described by reference model RM 3.3—the same as `Pistacchio`. The human designers who participated in this study had more than one year of experience in swarm robotics and some familiarity with AutoMoDe, the e-puck, and ARGoS3. We provided the designers with a visualization tool to produce and manipulate finite-state machines, to visualize simulations, and to compute the value of the objective function (Kuckling et al. 2021a). All simulations were executed in ARGoS3—the same setting as `Pistacchio`. The designers were allotted 4 hours per mission, as originally devised by Francesca et al. (2015).

`R-Walk`: we also include here a random-walk behavior in the study as a lower bound on the performance of robot swarms. In `R-Walk`, the robots move in a

Table 5.9: Neural network topology and parameters of the evolutionary process in `EvoCMY`. The neural network operates according to RM 3.3, see Table 5.7.

Architecture	
Fully-connected feed-forward neural network without hidden layers	
Input node	**Description**
$in_{a\in\{1,\dots,8\}}$	readings of proximity sensors $prox_{i\in\{1,\dots,8\}}$
$in_{a\in\{9,\dots,11\}}$	readings of ground sensors $gnd_{j\in\{1,\dots,3\}}$
$in_{a\in\{12\}}$	value of the density function $z'(n)$
$in_{a\in\{12,\dots,23\}}$	scalar projections of $V_{c\in\{C,M,Y\}}$
$in_{a\in\{24\}}$	bias input
Output node	**Description**
$out_{b\in\{1,\dots,4\}}$	tuples v' to map each velocity in the set $v_{k\in\{l,r\}}$
$out_{b\in\{5,\dots,8\}}$	activation of each color in the set $\{\varnothing, C, M, Y\}$
Connection	**Description**
$conn_{s\in\{1,\dots,192\}}$	synaptic connections with weights $\omega\in[-5,5]$
Evolution parameters	
Number of generations *	—
Population size	100
Elite individuals	20
Mutated individuals	80
Evaluations per individual	10
Post-evaluation per individual **	100

* The number of generations is computed according to the budget of simulations.
** The population obtained in the last generation is post-evaluated to select the best individual.

straight line in the arena, when they encounter an obstacle, they rotate for a random number of control steps and then resume their straight motion. `Random-Walk` is also conceived for the e-puck formally described by reference model RM 3.3.

Protocol

In this study, we consider a number of experimental scenarios that is larger than those of previous studies considering `C-Human`. Therefore, we cannot replicate the experimental procedure exactly as reported by Francesca et al. (2015). In our experiments, we use `Pistacchio` and `EvoCMY` to produce repeatedly more instances of control software than what `C-Human` can produce. We do so because producing control software with these automatic methods costs less than producing it with `C-Human`. We adjust the number of evaluations to obtain an equivalent number of observations across methods for statistical analysis.

`Pistacchio` and `EvoCMY` are given a budget of 100 000 simulations to produce each instance of control software. We produce 90 instances of control software with `Pistacchio` and other 90 with `EvoCMY`—10 per scenario. Each of these instances is assessed once to obtain 90 observations per method. `C-Human` produced 9 instances of control software, 1 per scenario. We obtain the equivalent 90 observations by assessing each of these instances 10 times. `R-Walk` is assess 10 times in each scenario to obtain 90 observations.

In the experiments, neither the automatic methods nor the human designers had direct access to information about the sheep control software. The dynamics between shepherds and sheep had to be discovered during the design process via simulations.

Statistics: We use box-plots to present the performance of the instances of control software obtained in the experiments. As in the other studies, comparative statements are supported with a Wilcoxon paired rank sum tests, at 95 % confidence (Conover 1999). We also present here the results of a Friedman test that aggregates the overall performance of the methods across the four missions—see Chapter 4.

5.3.3 Results

Figure 5.9 shows the performance of each method across the nine scenarios. Figure 5.10 shows the average rank of the design methods over the nine experimental scenarios. Demonstration videos of the behavior of the robots are provided in the Supplementary Videos of the dissertation (Garzón Ramos 2025).

The per-mission results and the Friedman test did not detect any significant difference between the performance of `Pistacchio` and `EvoCMY`, but the two are significantly better than `C-Human` and `R-Walk`. Moreover, `C-Human` was significantly better than `R-Walk`. The results show that automatic design was more effective than manual design in addressing the shepherding problems we considered. Also, all design methods generated collective behaviors that are more effective than the simple random walk—the lower bound. Our simulation-only comparison between `Pistacchio` and `EvoCMY` was not sufficient to identify possible performance differences between the modular and the neuroevolutionary approach—differences that have been observed in the studies with physical robots presented earlier in this dissertation.

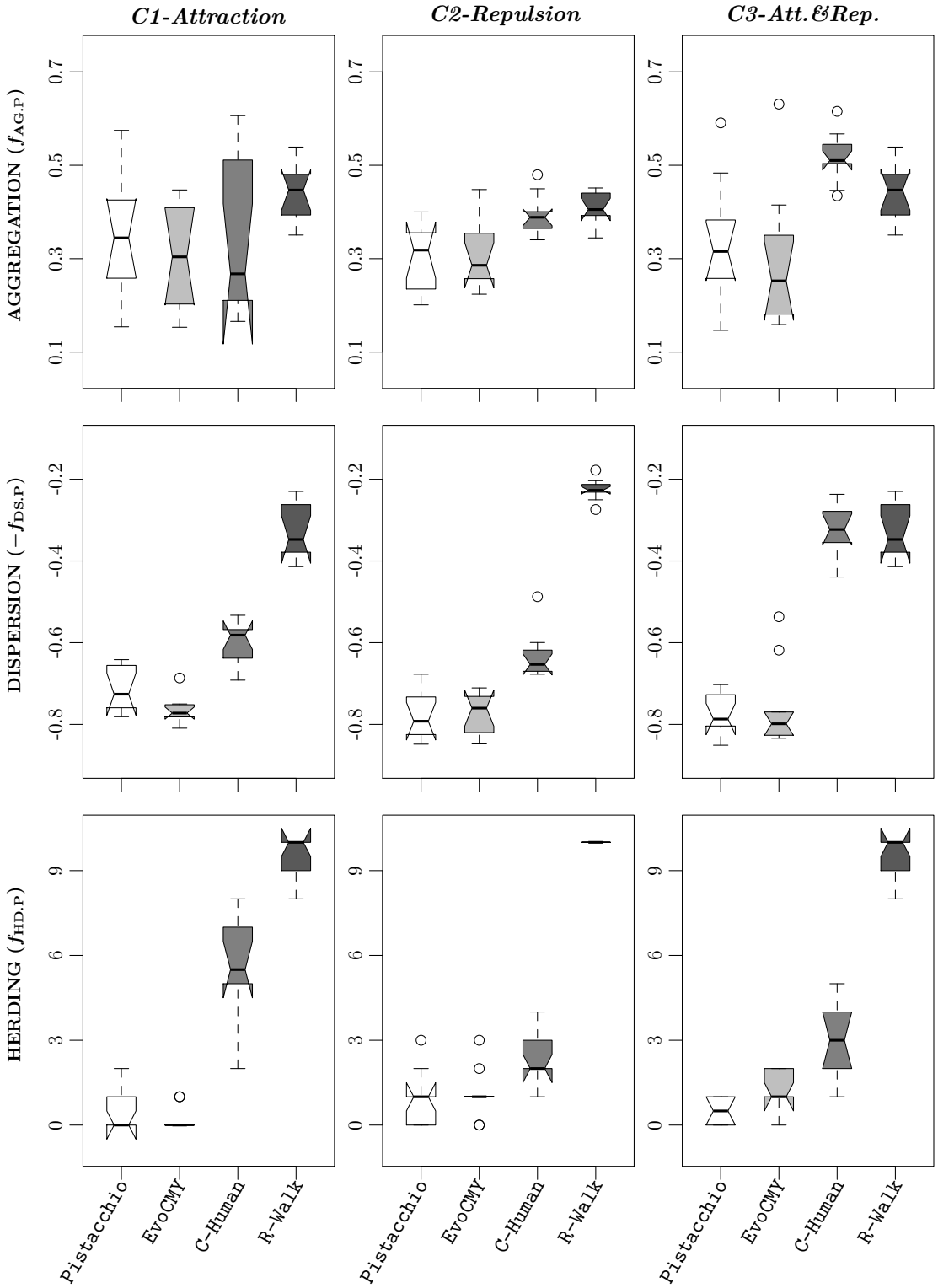


Figure 5.9: Performance obtained in the missions studied with `Pistacchio`. The plots show results per mission and sheep control software. Results per design method are presented with grayscale box-plots, `Pistacchio` (□), `EvoCMY` (▒), `C-Human` (■), `R-Walk` (■). These results have been obtained with simulations in ARGoS3. In all plots, the lower, the better.

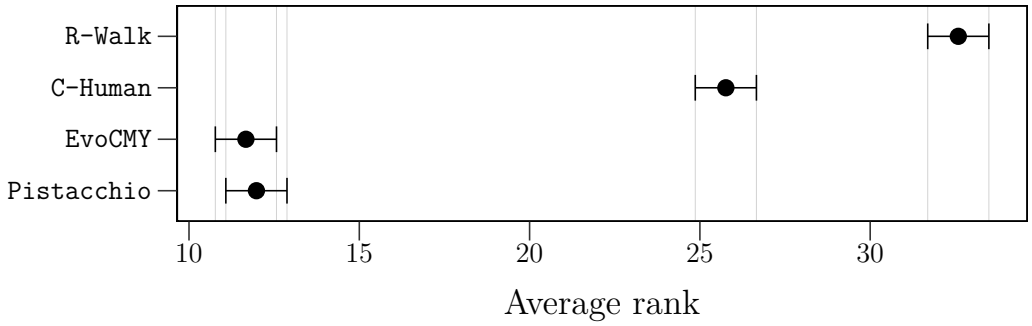


Figure 5.10: Friedman tests with aggregate results from `Pistacchio`'s study. The test considers the results obtained across all missions and provides a relative ranking between the methods. The plot shows the average rank of each method and its 95 % confidence interval. The lower the rank, the better.

Shepherding strategies obtained in the experiments

`Pistacchio` and `EvoCMY` leveraged coordination and localization via color signals to create effective interactions between shepherds and sheep. In most cases, the two methods designed behaviors in which the shepherds stimulated the sheep in similar ways. We describe here some shepherding strategies that were generated. The following are general observations we made over all instances of control software by visual inspection.

Grouping the sheep in aggregation: When the sheep operated with *C1-Attraction*, the shepherds displayed magenta to attract them and they were themselves also attracted to other shepherds that displayed magenta. In this way, shepherds and sheep remained close to each other, keeping the sheep close to their center of mass. The automatic methods designed a coordinated cooperative behavior between shepherds. When the sheep operated with *C2-Repulsion*, the shepherds displayed cyan and moved on a circular trajectory close to the walls of the arena. The sheep were therefore steadily repelled towards the center of the arena and formed a single group. When the sheep operated with *C3-Attraction&Repulsion*, the shepherds used a behavior similar to that observed in *C1-Attraction* or *C2-Repulsion*—no noticeable preference was observed.

Separating the sheep in dispersion: When the sheep operated with *C1-Attraction*, the shepherds remained close to the walls and displayed magenta to attract the sheep. This behavior dispersed the sheep along the edges of the arena, keeping them far from their center of mass. When the sheep operated with *C2-Repulsion*, the shepherds moved in circles in the center of the arena while displaying cyan. In this way, the shepherds separated the sheep by steadily pushing

them towards the walls. When the sheep operated with *C3-Attraction&Repulsion*, the shepherds used a behavior similar to that observed in *C1-Attraction* or *C2-Repulsion*—also no noticeable preference was observed.

Driving the sheep in herding: The shepherds reacted to the color yellow that the sheep displayed. When the sheep operated with *C1-Attraction*, the shepherds displayed magenta to attract the sheep. Simultaneously, they were also repelled from the color yellow the sheep displayed. In this way, the shepherds guided the sheep from the front. The two navigated the arena together until the sheep stepped into a white region and turned off their LEDs. When the sheep operated with *C2-Repulsion*, the shepherds displayed cyan to repel the sheep. Unlike the behavior observed in *C1-Attraction*, in this case, the shepherds were attracted to the color yellow that the sheep displayed. The simultaneous execution of these two behaviors resulted in shepherds that chased the sheep until the latter stepped into a white region and turned off their LEDs. The shepherds used a behavior similar to that observed in *C1-Attraction* or *C2-Repulsion* when the sheep operated with *C3-Attraction&Repulsion*. Also in this case, no noticeable preference was observed.

Color signaling and physical proximity were two alternative ways for the shepherds to interact with the sheep. Unlike `Pistacchio` and `EvoCMY`, `C-Human` produced control software that leveraged the color displayed by the shepherds only in a few cases. `C-Human` and `Pistacchio` operate on the same set of modules and software architecture, and can potentially produce control software that performs similarly. However, the human designers mainly used sub-optimal strategies (based on physical proximity) to enable the interaction between shepherds and sheep. In a way, they failed to discover and use the most effective strategy: the interaction via color signals.

5.3.4 Discussion

`Pistacchio` effectively searched the design space and exploited the dynamics between shepherd and sheep. The shepherds used color stimuli to interact with the sheep in a meaningful and good-performing way. In some cases, the shepherds coordinated with each other using the same stimuli. In our experiments with `TuttiFrutti`, `Mate`, and `Habanero`, we already showed that by enabling simple color signaling in AutoMoDe's modules, the automatic design process can generate mission-specific coordination and spatial-organization strategies. However, those results were obtained with a homogeneous robot swarm. In the experiments we presented here, `Pistacchio` generated similar mission-specific coordination and

spatial organization between shepherds and sheep. We show therefore that this ability of AutoMoDe can extend to heterogeneous setups.

So far, automatic design research has focused mainly on missions performed by a single robot swarm that operates alone. With our experiments, we showed that existing approaches are also well suited to address more complex heterogeneous scenarios. We consider heterogeneity in both the number of swarms that populate an environment and the type of control software they use. In our experiments, the sheep operated with finite-state machines. `Pistacchio` and `EvoCMY` generated effective shepherding strategies despite that their shepherds operated with different kinds of control software—finite-state machines and neural networks, respectively. A key element to enable a setup with heterogeneous control software was to formally define the capabilities of shepherd and sheep within a single reference model, RM 3.3.

A priori, we expected to observe a significant performance difference in AGGREGATION and DISPERSION when the sheep operated with diametrically different behaviors such as *C1-Attraction* and *C2-Repulsion*. However, no such performance difference was observed. This indicates that `Pistacchio` and `EvoCMY` simultaneously tailored the control software of the shepherds to the one of the sheep, and to the goal of the mission—regardless of the combination. On the other hand, `C-Human` was less effective. Human designers had difficulties on exploring the design space and finding good-performing shepherding strategies. We conceived the experiments in a way that the dynamics between shepherd and sheep had to be discovered during the design process. Automatic methods have a notable advantage in this task. The optimization process is more effective than a human designer in exploring the large and complex design space.

The shepherding problem was an appropriate framework to study the design of robot swarms that must interact with other robots. Our current experimental setup can be directly extended to missions that involve other types of interactions. The sheep we considered are rather individualistic: they react individually to the stimuli of the shepherds without considering the behavior of other sheep (beyond the physical proximity). Moreover, their naturally-static behavior made them easy to handle for shepherds. One could possibly create more complex missions with sheep that continuously move and operate with a more coordinated collective behavior—both cooperative or adversarial. In this sense, we believe research work presented here could bootstrap recent studies on the design of robot swarms that are robust to attacks from adversarial robots (Strobel et al. 2020, 2023; Castelló Ferrer et al. 2021; Reina et al. 2023)

5.4 Automatic design of robot swarms by demonstration

AutoMoDe is intended to minimize human intervention in the process of designing robot swarms. In most AutoMoDe methods, the only required input from the designer is defining the mission specifications and performance measures. However, a challenge still remains in in formulating a performance measure that accurately captures the desired swarm behavior—a task that requires expert knowledge. We contend that automatic design must evolve toward methods that use more natural and intuitive forms of mission specification—such as simplified language (Bozhinoski and Birattari 2022) or behavior demonstrations (Šošić et al. 2017; Gharbi et al. 2023). This is a necessary step to achieve truly automatic design processes that allow the design and deployment of robot swarms without the need for technical expertise.

In this study, we experiment with AutoMoDe and inverse reinforcement learning (Abbeel and Ng 2010) as an alternative to manually formulating an objective function to drive the design process. Inverse reinforcement learning can automatically generate an objective function based on user demonstrations. Both the objective function and the control software of the robots are learned simultaneously through an iterative process. This approach offers advantages over traditional AutoMoDe methods. Demonstrating desired behaviors is generally easier, more natural, and intuitive than manually creating an objective function that fully captures the desired swarm behavior.

As discussed in Chapter 2, we recently applied inverse reinforcement learning to AutoMoDe and structured a design process that combines the two (Gharbi et al. 2023). In this section, we build on those experiments to investigate the automatic design of robot swarms that perform sequences of missions—as previously studied with `Mandarina` in Chapter 4. We conduct this new study using DemoTuttiFrutti-`MO` (DTF-`MO`), a method from the AutoMoDe family that integrates inverse reinforcement learning with multi-criteria design. Similarly to `Mandarina`, the missions here consist of two sub-missions that must be performed sequentially. We also use `MoCA` to provide the robots with environmental signals that inform the swarm which sub-mission to execute at any given time. The sub-missions are defined through demonstrations of the desired spatial organization the swarm should achieve. DTF-`MO` addresses these sub-missions concurrently, with a design process that generates a single instance of control software based

on demonstrations for the two sub-missions. No human intervention is required other than providing the demonstrations. The challenges for DTF-MO in this design problem are: (i) identifying relevant features that define the sub-missions using only demonstrations, and (ii) designing a single instance of control software capable of handling the two sub-missions concurrently.

We compare the performance of DTF-MO with a baseline method that involves a higher degree of human intervention. In this baseline, the sub-missions are handled independently and the design process generates two separate instances of control software—one for each demonstrated sub-mission. Then, we manually hard-code a transition rule that allows the robots to switch between sub-missions. This procedure is based on experiments conducted by Duarte et al. (2016), as described in Chapter 2. We also include a second baseline in which the control software is completely produced by hand.

This is probably the most complex design problem addressed in this thesis. With this study, we investigate whether the results achieved with TuttiFrutti and Mandarina can be extended to a design process in which missions are specified with demonstrations.

5.4.1 DemoTuttiFrutti-MO (DTF-MO)

Robot platform

DTF-MO produces control software for the version of the e-puck introduced with TuttiFrutti—see Figure 3.1. The functional capabilities of the robot are adopted from RM 3 without further modification. The reference model RM 3 is detailed in Table 3.1.

Modular control architecture

DTF-MO generates control software by fine-tuning and assembling predefined software modules into probabilistic finite-state machines. It operates on the same parametric modules originally designed for TuttiFrutti. As with Mandarina, we leverage here TuttiFrutti's modules to conceive sequences of missions in which an environmental signal—specifically, a color displayed by the walls enclosing the arena in which the robots operate—is available to inform the robots that they should transition from one sub-mission to the other. The parameters of the software modules are automatically tuned during the design process. The modules are detailed in Table 3.2.

Automatic design process

The automatic design process in DTF-MO is an iterative procedure consisting of two steps. First, an objective function is learned on the basis of user demonstrations. Then, this objective function is used to generate instances of control software that perform the mission according to the learned objective function. The new iteration starts by using the generated control software to refine the estimation of the objective function, with the aim of fitting better the user demonstrations. In subsequent iterations, the objective function is improved, and therefore the method produces control software that attains more effectively the demonstrated behavior. This iterative process simultaneously refines both the learned objective function and the control software for the robots. The following sections provide a detailed description of this process.

Learning the objective function from demonstrations: DTF-MO builds on Demo-Cho (Gharbi et al. 2023), and also adopts the algorithm known as *apprenticeship learning via inverse reinforcement learning* (Abbeel and Ng 2004). In DTF-MO, the user is required to provide demonstrations for each sub-mission in the sequence, which specifically indicate the robots' positions and exemplify the desired spatial distribution of the swarm in the environment. For the apprenticeship learning algorithm, the position of the robots—i.e., swarm state—is mapped to a mission-independent feature vector $\phi(s) \in [0,1]^k$. This feature vector encapsulates the spatial distribution of the swarm—inter-robot distances—and the location of the swarm in the arena—landmarks-robots distances. It is important to stress here that the feature vector is defined only once, depends only on the environment and the robots, and is agnostic to the mission to perform. That is, it can map the desired behavior from different sub-missions without further modification.

DTF-MO applies mapping transformations, which are similar to those of Demo-Cho, to convert the positions of the robots into features that describe the spatial relationships between robots, and between robots and their environment. In our experimental setup, we consider a swarm of 20 e-pucks that operate in an arena that comprises three landmarks—see experimental setup in this section. The mapping results into 60 features that describe the distance between each robot and the three landmarks, and 20 features that describe the distance between each robot and its closest peer. The features related to landmarks are calculated as follows:

$$\phi_{rl} = \begin{cases} 1, & \text{if robot r is inside landmark l;} \\ 10^{-\frac{2}{d}D_{rl}}, & \text{otherwise.} \end{cases} \tag{5.11}$$

Here, d is the diameter of the arena and D_{rl} is the distance from robot $r = \{1, \ldots, n\}$ to landmark $l = \{1, \ldots, m\}$, where n and m are the number of robots and the number of landmarks, respectively. For each landmark, $r = 1$ is the closest robot to the landmark itself and $r = 20$ the farthest one.

The features related to distance between robots are calculated as follows:

$$\phi_r = 10^{-\frac{2}{d}D_r}, \tag{5.12}$$

where D_r is distance between robot $r = \{1, \ldots, n\}$ and its closest peer, and n is the number of robots. Here, $r = 1$ is the robot whose distance to its closest peer is the shortest; and $r = 20$, the one whose distance to its closest peer is the longest. Swarm robots are interchangeable, with mapping based on relative positions. The feature vector, $\phi(s) = (\phi_{11}, \ldots, \phi_{1m}, \ldots, \phi_{n1}, \ldots, \phi_{nm}, \phi_1, \ldots, \phi_n)$, describes the spatial configuration of the swarm, regardless of individual identities.

In Demo-`Cho`, the apprenticeship learning algorithm was limited to learning a single objective function from the demonstrations. In DTF-`MO`, the algorithm learns an objective function for each sub-mission. The sequences of missions we consider in this study comprise two sub-missions. Therefore, the apprenticeship learning algorithm must learn two objective functions to be optimized, $R_{w_1}(s) = w_1 \cdot \phi(s)$ and $R_{w_2}(s) = w_2 \cdot \phi(s)$, where w_1 and w_2 are the weight vectors that describe the importance given to each feature.

We set a maximum number of iterations as the termination criterion for the apprenticeship learning—the same criterion defined in Demo-`Cho`. The maximum number of iterations is chosen to be sufficiently large to allow w_1 and w_2 to converge.

Generating control software with multi-criteria optimization: DTF-`MO` generates control software based on the objective functions learned through apprenticeship learning, $R_{w_1}(s)$ and $R_{w_2}(s)$, which define two performance measures to be maximized. To do so, DTF-`MO` conducts a single-objective optimization process that considers both $R_{w_1}(s)$ and $R_{w_2}(s)$ concurrently. The method linearly combines the two objectives into a single one: $R(S) = R_{w_1}(s) + R_{w_2}(s) = w_1 \cdot \phi(s) + w_2 \cdot \phi(s)$. This simple approach is adopted from the experiments conducted in Chapter 4. The two objective functions are weighted equally, as there is no prior reason to assume that one sub-mission is more important than the other.

As in `TuttiFrutti` and `Mandarina`, the design process in DTF-`MO` is based on Iterated F-race and relies on simulations conducted in ARGoS3. During the design process, DTF-`MO` produces a set of candidate control software instances, each representing a potential solution to the sequence of missions. In order to select the

best performing instance, we assume that the distance in the feature space indicates how close a behavior is to the demonstrations given for each sub-mission. Each instance has two associated distances, one for each objective, $R_{w_1}(s)$ and $R_{w_2}(s)$. DTF-MO computes the l^2-norm to aggregate the distance values of an instance into a single measure. This measure can be used to compare instances across the two sub-missions. The smaller the l^2-norm, the closer the control software is to the demonstrations. This method has been shown to be effective in feature selection problems (Salman et al. 2022).

5.4.2 Experimental setup

Missions

The robots operate in a hexagonal arena of 2.60 m^2 with gray floor. The arena comprises three landmarks: a white circular region in the center and two black triangular regions at the left and right sides. These are the three landmarks considered in the learning process. The arena is surrounded by 24 MoCA's RGB blocks, which can change their color at run-time. The 4 blocks at the right corner of the arena constantly display the color green.

The robots have 120 s to perform the two sub-missions, with 60 s allocated to each. The walls of the arena (except for the right corner) are red for the first 60 s and then turn blue, indicating which sub-mission should be performed. This is the environmental signal DTF-MO can use to transition from one behavior to another. The reaction to this cue and the behavior transition must be automatically inferred during the design process. DTF-MO extracts and evaluates the position of the robots only at the end of each sub-mission—that is, after their allocated 60 s. Figure 5.11 shows the experimental arena.

We conceived five sub-missions that we paired to produce twelve sequences of missions.[1] Figure 5.11 shows the sub-missions as specified by the demonstrations. The sub-missions we conceived are inspired by the spatial-organizing behaviors considered in our previous studies, as well as those investigated by Francesca et al. (2015) with Chocolate. For clarity, we also provide a written description of the desired behavior specified by the demonstrations in Figure 5.11. SUB-MISSION A (m_A): the robots must aggregate in the center of the arena, in the white area. SUB-MISSION B (m_B): the robots must line in the edges of the arena, except for the two rightmost walls. SUB-MISSION C (m_C): the robots must cover the entire

[1] By combining the five missions, we could generate twenty sequences. We selected twelve of them for these experiments.

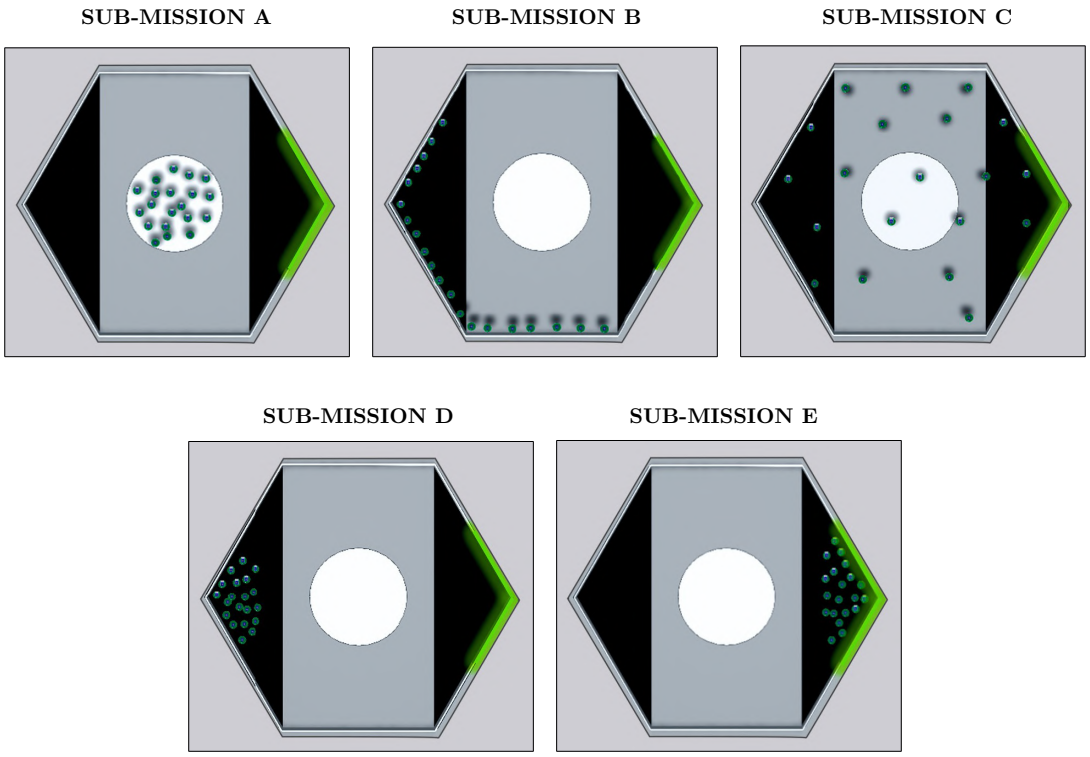


Figure 5.11: Experimental arenas for DTF-MO's experiments. The figure shows the three landmarks, RGB blocks, and a swarm of 20 e-pucks that exemplify the demonstrations given for each sub-mission. The green walls do not change color. Other walls switch from red to blue to indicate the transition between sub-missions. This is the signal that the automatic design process must learn to exploit. Demonstrations are provided in a user interface developed with the Unity game engine, which parses the information to a format usable in ARGoS3.

arena. SUB-MISSION D (m_{D}): the robots must aggregate in the left black area. SUB-MISSION E (m_{E}): the robots must aggregate in the right black area.

The twelve missions are organized in pairs, with each pair denoted as $m_{\mathrm{P{\cdot}Q}}$, where P and Q represent the sub-missions to be executed sequentially. The pairs are: $m_{\mathrm{A{\cdot}B}}$, $m_{\mathrm{B{\cdot}A}}$, $m_{\mathrm{C{\cdot}B}}$, $m_{\mathrm{B{\cdot}C}}$, $m_{\mathrm{C{\cdot}D}}$, $m_{\mathrm{D{\cdot}C}}$, $m_{\mathrm{E{\cdot}A}}$, $m_{\mathrm{A{\cdot}E}}$, $m_{\mathrm{B{\cdot}E}}$, $m_{\mathrm{E{\cdot}B}}$, $m_{\mathrm{D{\cdot}E}}$ and $m_{\mathrm{E{\cdot}D}}$.

Baseline methods

DemoTuttiFrutti-2SO (DTF-2SO): a method that addresses the sub-missions independently, generating separate instances of control software for each demonstrated sub-mission. Like DTF-MO, DTF-2SO learns objective functions for the sub-missions through apprenticeship learning, using the same demonstrations as DTF-MO. However, DTF-2SO conducts separate single-objective optimization processes for $R_{w_1}(s)$ and $R_{w_2}(s)$, producing two instances of control software—one that

maximizes $R_{w_1}(s)$ and another that maximizes $R_{w_2}(s)$. After the design process ends, we manually assemble the two instances by hard-coding a transition condition from one to the other. This transition is triggered by the change of the color of the walls, from blue to red, which indicates the change of sub-mission to be performed. Therefore, this environmental signal indicates the control software produced by DTF-2SO to switch its behavior. As mentioned earlier, this approach is similar to the one devised by Duarte et al. (2016). Other than this difference, DTF-2SO and DTF-MO are identical. The two methods use TuttiFrutti's modules and control architecture, and they design control software for the e-puck defined by reference model RM 3. DTF-2SO's optimization process is driven by Iterated F-race, like DTF-MO.

Manual design: Without a pre-existing theoretical baseline, we developed our own. Using TuttiFrutti's modules, we manually designed a control software instance for each sub-mission. Then, we manually assembled the two corresponding instances for every sequence. In this procedure, we use the same hard-coded transition we implemented in DTF-2SO.

Protocol

We generate 120 instances of control software with DTF-MO and other 120 with DTF-2SO—10 for each mission. In every design process, we specify the sequence with 10 demonstrations—five for each associated sub-mission. In all cases, the apprenticeship learning algorithm performs 15 iterations. In every iteration, Iterated F-race has a budget of 100 000 simulations to produce a control software instance. DTF-MO was given these 100 000 simulations to produce a single instance of control software for the two sub-missions. DTF-2SO was given an budget of 100 000 simulations to produce each of the two instances of control software, which are required by the two sub-missions. This resulted in an advantage for DTF-2SO, considering that its search space is less than half that of DTF-MO.

Statistics: We use box-plots to present the performance of the instances of control software obtained in the experiments. We use the l^2-norm to quantitatively compare the performance of the instances of control software produced with DTF-MO, DTF-2SO, and the manual baseline—as described for the automatic design process. We use heat-map plots to present the distribution of weights, w_1 and w_2, that the apprenticeship learning algorithm assigned to the linear combination of features. We conduct a visual inspection of the behaviors of the robots to verify their ability to reproduce the given demonstrations.

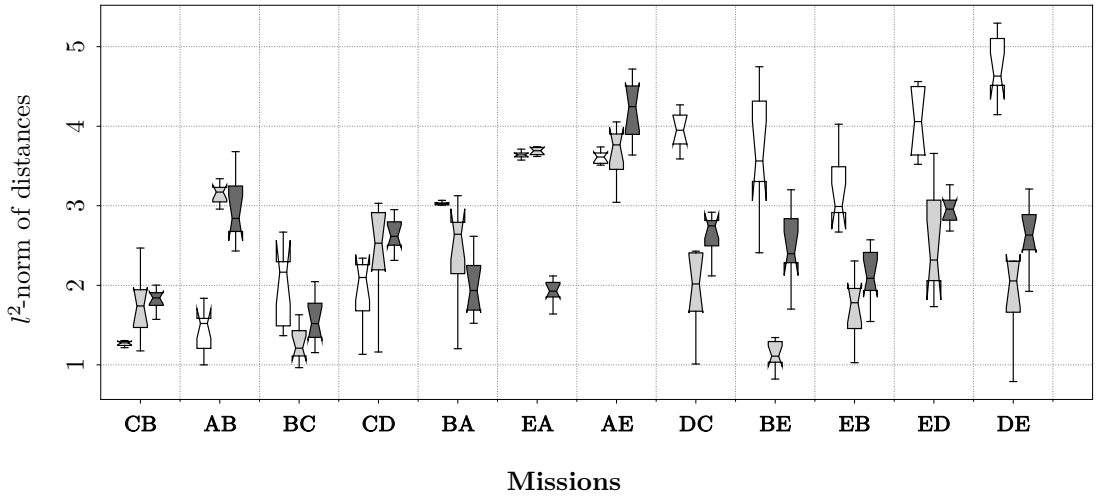


Figure 5.12: l^2-norm of the distance in the feature space between the observed behavior of the robots and the demonstrations. Results per design method are presented with grayscale box-plots, DTF-`MO` (□), DTF-`2SO` (▒), `Manual design` (■).These results have been obtained with simulations in ARGoS3. In the plot, the lower, the better.

5.4.3 Results

Figure 5.12 shows the performance of each method for the twelve sequences of missions. Demonstration videos of the behavior of the robots are provided in the Supplementary Videos of the dissertation (Garzón Ramos 2025).

The performance results presented in Figure 5.12 show that DTF-`2SO` outperforms DTF-`MO` and the manual baseline in half of the missions. Specifically, the l^2-norm is lower for DTF-`2SO` in these missions. Two factors can influence the performance of the methods: (i) the ability to learn the objective functions from the demonstrations, (ii) and the effectiveness of the optimization process in finding suitable control software.

We first investigated the ability of DTF-`MO` and DTF-`2SO` to identify the subset of features that is most relevant for each sub-mission, and properly learn a distribution of weights for w_1 and w_2. The heatmap plots showed that, across all missions, the two methods were able to identify the most relevant features during the learning process—see Szpirer et al. (2024b). Indeed, DTF-`MO` and DTF-`2SO` assigned mission-specific distribution of weights that properly characterized the desired spatial relationship between robots and environment—according to the demonstrations of the two associated sub-missions. We illustrate these results here with the weights assigned in mission $m_{D.E}$ (Figure 5.13).

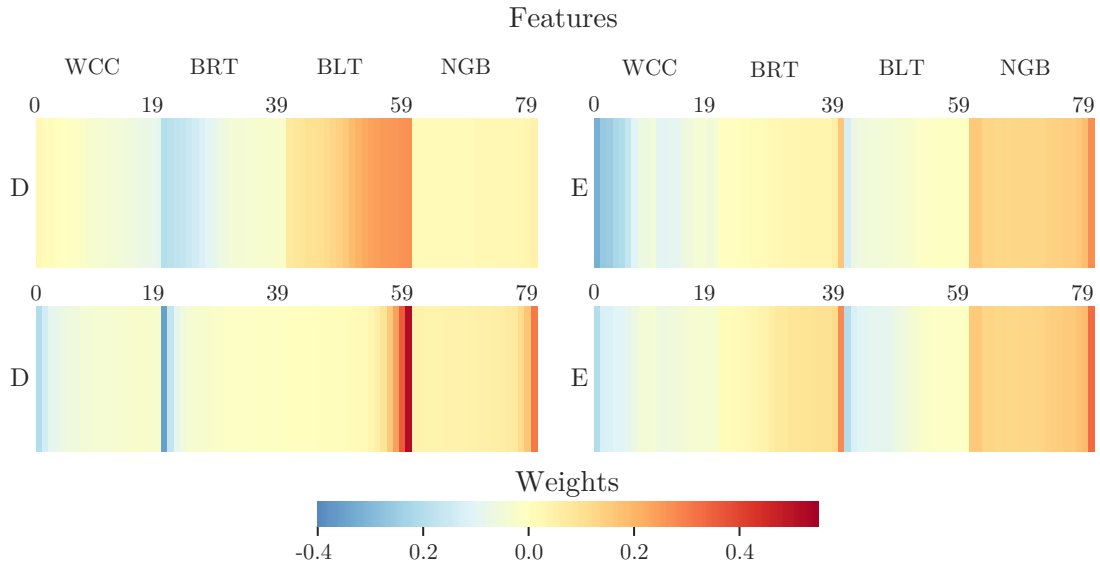


Figure 5.13: Heat-map plots that represent the weight given to each of the 80 features in the objective functions in Mission $m_{D.E}$. The feature vector is organized as follows: positions 0 to 19 are associated to the white circular region (WCC); positions 20 to 39 are associated to the black right triangular region (BRT); positions 40 to 59 are associated to the black left triangular region (BRT); and positions 60 to 79 are associated to the inter-robot distances (NGB). When the weight is near zero, the feature's impact on the objective function is minimal. Negative weights (blue color) require feature minimization, positive weights (orange color) require maximization. As features are inversely proportional to distances, maximizing a feature minimizes its corresponding distance, and vice versa. DTF-`MO` (top) and DTF-`2SO` (bottom) learned for (i) m_{D}, to maximize the distance from the right triangle and minimize it with the left one; (ii) m_{B}, to maximize the distance with the left triangle and the center and minimize it with the right triangle and the peers.

In SUB-MISSION D, the robots were expected to stay close to the left black landmark and far from the right one. Conversely, in SUB-MISSION E, the robots were expected to remain close to the right black landmark and far from the left one. Figure 5.13 shows that both DTF-MO and DTF-2SO set the appropriate emphasis on the related features in each case. These results allowed us to isolate the main factor that caused the performance difference: the optimization process and its ability to produce suitable control software.

Figure 5.12 shows that DTF-MO performed better than the manual baseline and equally or better than DTF-2SO in four missions: $m_{C \cdot B}$, $m_{A \cdot B}$, $m_{C \cdot D}$ and $m_{A \cdot E}$. We visually inspected these missions to determine possible reasons for the performance difference with respect to the general results. We observed that DTF-MO was more effective because it designed behaviors that ease the transition from one sub-mission to the other. By concurrently considering the two sub-missions, it optimized not only each part of the sequence but also the transition phase between them. For example, in SUB-MISSION A of $m_{A \cdot B}$, the robots do not exactly aggregate in the center of the arena—as indicated in the demonstration. Instead, they remain near the borders of the white circle to perceive the walls more easily and react promptly to the signal. Similarly, we observed that in SUB-MISSION D of $m_{C \cdot D}$, the robots not only head toward the left landmark. They also trigger their own cue to signal other robots to follow them—a color-based communication behavior previously observed in TuttiFrutti.

We also inspected two missions in which neither DTF-MO nor DTF-2SO outperformed the manual baseline. We observed that missions $m_{E \cdot A}$ and $m_{B \cdot A}$ are challenging because of a difficult transition phase between the sub-missions. DTF-MO focuses on the second sub-mission and optimizes performance for it, without risking performance loss caused by the time spent in the transition. In DTF-2SO, the optimization process is conducted independently for each sub-mission and the control software is manually assembled afterward, preventing the generation of behaviors that consider the transition between sub-missions. For example, in $m_{E \cdot A}$, the robots effectively aggregate in the right green corner. However, from this corner, some of them are unable to perceive the cue to start performing mission SUB-MISSION A, and fail to aggregate in the center of the arena—as required.

5.4.4 Discussion

The experiments show that DTF-MO is a viable method to automatically design robot swarms that perform sequences of missions, which are specified via demon-

strations. However, the method was challenged by the increased search space of conducting a design process that tracts the sequences as a whole. DTF-MO had to simultaneously design effective control software for two sub-missions, identify the environmental signal, and define the transition rule. Conversely, DTF-2SO benefited from addressing the sub-missions independently during the design process and from a higher degree of human intervention: manually assembling the generated control software and hard-coding the transition rule. We argue that this intervention eased the exploration of the search space for DTF-2SO with respect to DTF-MO, and allowed it to perform better. The visual inspection show that DTF-2SO's advantage is challenged when the demonstrations present conflicting robot positioning, as this can prevent the swarm from properly triggering the manually coded transition rule. Devising an effective transition rule is heavily influenced by the designer's bias, and it can become ineffective without prior experimentation/validation on the mission at hand.

A limitation of our approach to the design of robot swarms by demonstration is that it currently only supports specifying static spatial-organizing behaviors, as noted in previous experiments with Demo-Cho (Gharbi et al. 2023). This restricts the range of behaviors that can be automatically generated. To expand on our previous research, we studied how a robot swarm can switch between behaviors to perform missions in sequence. The experimental protocol we developed for Mandarina turned out to be useful in designing these more complex collective behaviors.

In this study, DTF-MO successfully generated control software based solely on demonstrations of the desired swarm behavior. This shows that some of the multi-criteria design concepts studied with Mandarina can be applied together with inverse reinforcement learning. However, DTF-MO's performance still falls short compared to that achieved with manual intervention by human designers (i.e., DTF-2SO). In DTF-MO, we used a rather simple multi-criteria optimization strategy and we expect that more advanced alternatives (Emmerich and Deutz 2018) could possibly extend the range of missions it can address. In the literature, there are more advanced optimization algorithms available that could bootstrap the design capabilities of DTF-MO, while maintaining the fully automatic nature of its design process. As discussed in Chapters 2 and 4, the selection of appropriate optimization algorithms is an open problem in the automatic design of robot swarms.

5.5 Outreach with the automatic modular design of robot swarms

How to engage the general public in swarm robotics using AutoMoDe? We contend that frontier research must generate knowledge that is accessible to society in all contexts, and for both the specialized and non-specialized public. Throughout the development of this thesis, we were committed to communicating the context, research, and outcome of our work to a wide range of audiences. The goal we set for our efforts was to increase the visibility of swarm robotics and the automatic design of robot swarms.

We focus our outreach strategy on producing and sharing two types of video communications that highlight our research outcomes.[2] The first type, "Swarm Robotics Capsules," are short, informative clips that introduce fundamental concepts of swarm robotics and showcase parts of our experiments. These were designed to provide an accessible introduction to swarm robotics for enthusiasts. The second type, "Holiday Specials," are seasonal video clips featuring relatable swarm robotics experiments. These were created to reach a broader audience beyond the robotics community, with the aim of making swarm robotics visible to all.

We produced the Holidays' Specials videos using AutoMoDe. To comment on our approach, it is convenient to first illustrate a possible practical application for the automatic off-line design of robot swarms. Meet Franziska (Figure 5.14).

Franziska's swarm of robot actors

Franziska is a young digital content creator who recently gained popularity on a mainstream video platform. She became a trending figure thanks to her unique covers of famous movie scenes, performed by a swarm of robot actors she developed during her PhD studies. As her follower count grows, Franziska has already started accepting sponsored deals. What began as a sporadic hobby—manually programming the robots to recreate movie scenes—has now evolved into a full-time job where she must produce both these recreations and original content that can be monetized.

To maintain momentum on her channel, Franziska now needs to upload high-quality content on a consistent schedule. How can Franziska

[2]This science communication material is an original contribution of this dissertation, produced within the framework of the DEMIURGE project, which supported the development of the thesis. The material is available at `https://demiurge.be/outreach.html`.

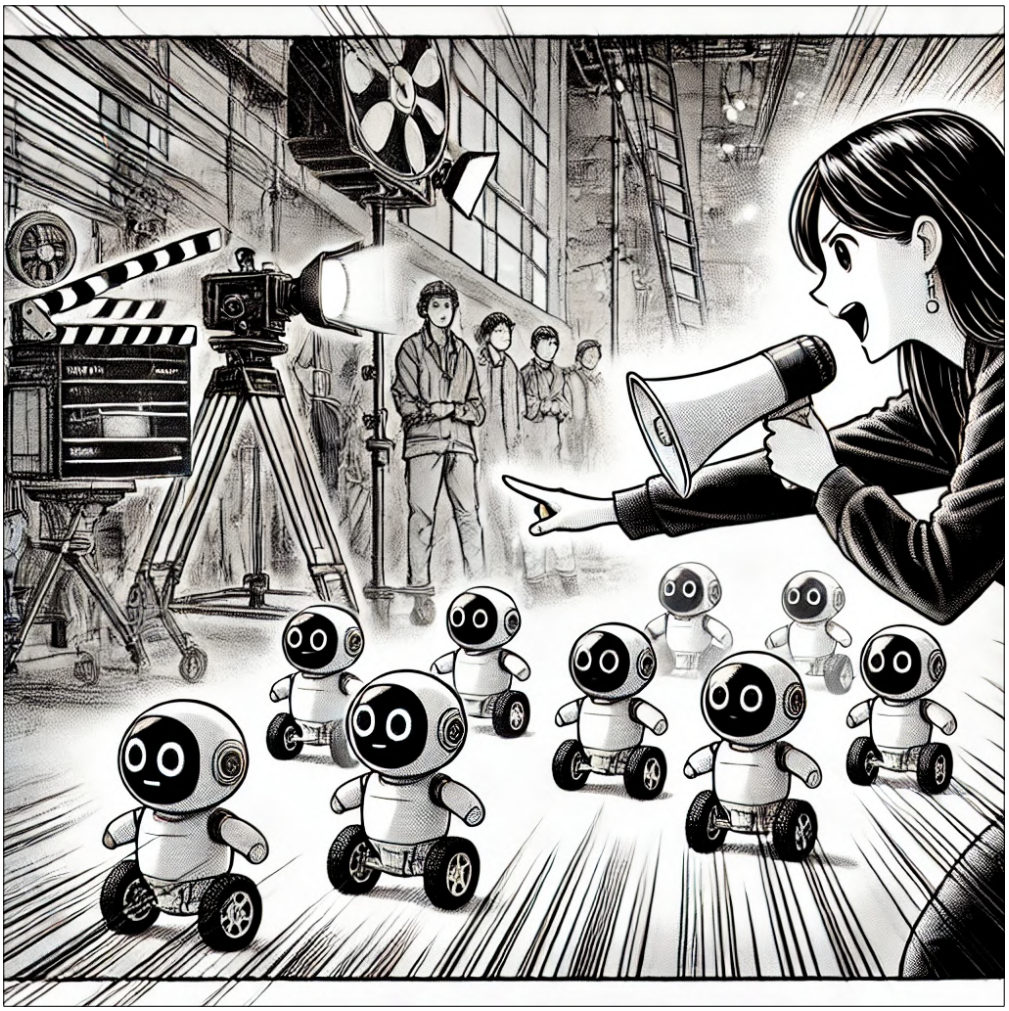


Figure 5.14: Franziska and her swarm of robot actors. Illustration generated with Open AI's DALL·E 3 model.

possibly keep up the pace as an independent creator? After all, her channel is just getting started and she is still running everything on her own. She handles all the production by herself—writing scripts, scouting filming locations, building props, setting up spaces, directing shoots, editing the videos, and sharing them on social media. The most time-consuming task, however, is programming the robot swarm to perform the scenes exactly as scripted. When video creation was just a hobby, she could take one or two months to prepare and publish a single video. But now that she is pursuing this professionally, she needs to release at least two new videos every week, on schedule, to meet the expectations of her growing audience and sponsors.

With little time between productions, Franziska can no longer manually program the robots herself. To keep up, she decided to adopt an automatic off-line design approach to streamline this process. Now, all she needs to do is script the video and sketch the filming location (the mission specification). This information is then given to an automatic design method which generates the control software the robots need to perform the scene. This approach is ideal for Franziska's workflow. Each video has different scripts and locations, making it impossible to create a one-size-fits-all control software instance for her videos. Instead, each video demands custom-tailored control software, which the automatic design method can provide. Renting filming locations is also costly for Franziska, so every minute counts. Thanks to her automatic off-line design approach, the robot swarm is always ready to perform right out of the box—no rehearsals, prior testing, or setting up infrastructure is needed before shooting.

Franziska is also benefitting from recent advances in the automatic design of robot swarms. It would be difficult for Franziska to formulate a precise performance measure (objective function) to evaluate how well the robots follow her script in every automatic design process. Fortunately, she can now rely on a simpler solution: she can start the design process by only demonstrating the desired positions for the robots in a scene. The automatic design process then generates interaction strategies for the swarm, allowing the robots to achieve the demonstrated positions and communicate and coordinate with each other as needed during filming. Additionally, Franziska can now script multiple scenes for a video at once, and the robots will play them

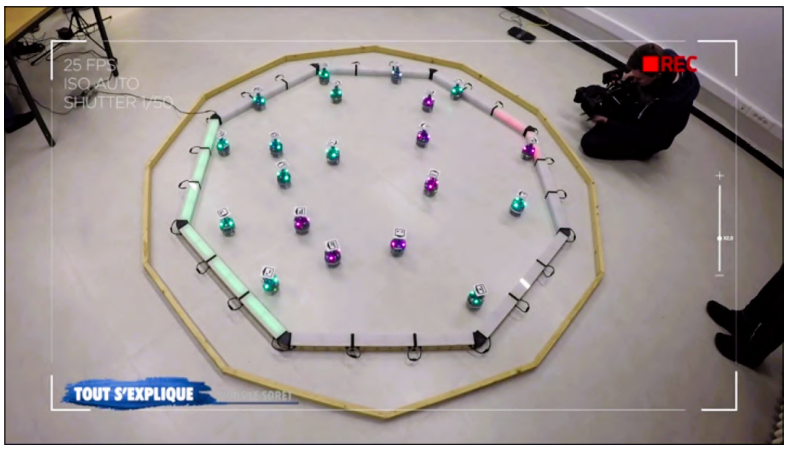
in sequence, responding to her direction cues (signals) for "action" and "cut."

As all robot programming happens off-line with an unattended process, Franziska has more time to focus on the creative aspects of video production. She can now refine her scripts, build better props, create more elaborate sets, improve her video editing, and engage more with her followers and sponsors on social media. This has resulted in higher quality content and better engagement with her community. The automatic off-line design of robot swarms has transformed Franziska from a hobbyist into a successful one-woman enterprise.

Our approach to creating video material for scientific communication with AutoMoDe was not so distant from Franziska's approach. We repurposed the experimental setups and tools described in this dissertation to produce demonstrations for the Holidays' specials videos—see Figure 5.15. The e-pucks were the robot actors we featured in our productions. Like Franziska, we used AutoMoDe to avoid the effort of manually programming the robots for these demonstrations and instead focused on improving the creative aspects of the videos. First, we defined a mission for the robots that could be easily understood by a broad audience. Next, we selected an appropriate method from our repertoire which should be capable of performing the mission. While the design process was executed off-line, we worked on the remaining elements of the creative production. In our videos, we used various cinematographic techniques, storytelling, and relatable soundtracks. We focused on seasonal themes, as they could easily convey visual messages across different countries, cultures, and languages. In these videos, we always only showed what robots are capable of and what we could do with our research. The content we produced kept a rigorous and scientifically sound narrative. At each production step, we curated the content and verified the accuracy of the message we were conveying and the information provided.

V1: Christmas clip: Automatic design of swarms of Santa's little helpers (a.k.a. robot swarms). In this Christmas clip, we use AutoMoDe to design a swarm of Santa's little helpers. The little helpers are e-puck robots, and their mission is to first step into the machines until they turn on, and then transport presents from the machines to the Christmas tree. The collective behavior of the swarm was automatically produced, without the need for human intervention in the design process. We used `TuttiFrutti` to design the robot swarm and developed a Christmas-themed interface in the Unity game engine to monitor the swarm

Demonstration of TuttiFrutti's STOP mission in the media



V1: Christmas-themed video

V2: Easter-themed video

V3: Christmas-themed video

V4: Halloween-themed video

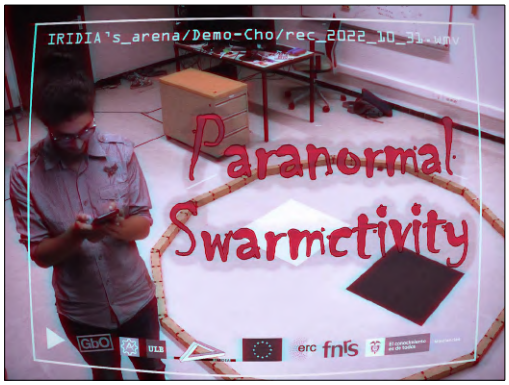


Figure 5.15: Snapshots of swarm robotics popularization videos. These videos are outreach results from the thesis. On top, a demonstration of TuttiFrutti's experiments presented in the Belgian media. The videos V1 to V4 are realizations of robot swarms in seasonally-themed missions, all produced using AutoMoDe. The videos are available in the Supplementary Videos of the dissertation (Garzón Ramos 2025).

during the execution of the mission. The interface for monitoring the robot swarm is part of our ongoing research on the realization of robot swarms using the Robot Operating System (ROS) (Kegeleirs et al. 2024a; Legarda Herranz et al. 2022). A snapshot of the video is shown in Figure 5.15 (mid-left).

V2: Automatic design of collective (collecting) behaviors for robot (rabbit) swarms: Easter egg hunt. In this Easter clip, we use AutoMoDe to design a swarm of Easter egg hunters. The "rabbits" are e-puck robots, and their mission is to gather in a basket the Easter eggs that randomly appear in the garden. The collective behavior of the swarm was automatically produced, without the need for human intervention in the design process. We used `TuttiFrutti` to design the robot swarm and developed an Easter-themed interface to monitor the swarm during the execution of the mission—as in V1. A snapshot of the video is shown in Figure 5.15 (mid-right).

V3: Swarming lights on winter nights–AutoMoDe edition: a robot swarm ballet. In this Christmas clip, we use AutoMoDe to design the collective behavior for a swarm of shepherd robots that drive their peers towards a Christmas tree. The mission of the shepherd robots (in red) is to position as many of the yellow robots in the tree. The collective behavior of the swarm of shepherd robots was produced automatically, without human intervention in the design process. We used `Pistacchio` to design the robot swarm and developed a new Christmas-themed interface to monitor the swarm during the execution of the mission—as in V2 and V3. A snapshot of the video is shown in Figure 5.15 (bottom-left).

V4: Paranormal swarmctivity: a ghostly robot swarm that learns from demonstrations! In this Halloween clip, we use an extension to the AutoMoDe approach, Demo-`Cho`, to produce an action-live video. Starting from demonstrations, Demo-`Cho` designs the collective behavior of a robot swarm. In the video, the researcher demonstrates three collective behaviors that are imitated by the robot swarm. The collective behaviors of the swarm were produced automatically, without the need for human intervention in the design process. The original idea for developing DTF-`MO` resulted from using Demo-`Cho` in the sequential demonstrations shown in this video. A snapshot of the video is shown in Figure 5.15 (bottom-right).

The video material we created using AutoMoDe, along with our methodology for producing it, became a key tool for communicating the scientific outcome of our research during presentations, invited talks, and media interactions—see Figure 5.15 (top). Through our efforts to produce content for science communication, we learned how outreach and public engagement not only foster our creativity but

also enrich our scientific work. So far, in this dissertation, we have focused on demonstrating the versatility of AutoMoDe in generating collective behaviors for robot swarms through extensive experimentation. The final contribution of this thesis, presented in this section, aims to motivate further application of AutoMoDe in creating scientific and educational content to popularize swarm robotics. With this contribution, we join ongoing efforts to engage the general public in shaping the way we understand, develop, use, deploy, and teach swarm robotics (Hamann et al. 2018; Carrillo-Zapata et al. 2020; Garzón Ramos et al. 2021a; Alhafnawi et al. 2022).

6. Conclusions

In this dissertation, we investigated the design of collective behaviors for robot swarms via automatic modular design. We introduced novel experimental scenarios where robots use various forms of signaling to collectively perform their mission. Additionally, we addressed design problems where robot swarms must be realized under concurrent design criteria. In our research, we collected sufficient empirical evidence to assert that AutoMoDe can leverage environmental and inter-robot signaling to tackle missions where a robot swarm is required to communicate, react to events, and perform missions sequentially—the thesis developed in this dissertation. We provide the swarm robotics community with a suite of AutoMoDe methods for generating control software for robot swarms. These address state-of-the-art design problems by enabling the coordination of robots through visual signaling.

In this final chapter, we summarize the major contributions of the thesis, highlight key findings, and propose directions for future efforts.

Automatic design, both fully- and semi-automatic, has become a viable approach to the realization of robot swarms. Our review of the literature in Chapter 2 showed that there is a growing scientific community contributing to the development of new methods and approaches in the field. However, the proposed methods are rarely evaluated for general applicability, and there is no well-established state of the art or benchmark problems to assess their effectiveness. Comparisons between methods are also rare, as are systematic evaluations across robot platforms with varying capabilities. We argue that the main body of literature needs to shift from proving feasibility to thoroughly investigating the strengths and limitations of the design approaches.

The research presented in this dissertation was conducted using a systematic and empirical approach. We developed a series of methods from the AutoMoDe family, each building on previous ones. For each method, we identified its unique and/or inherited elements, evaluated its ability to design collective behaviors across different missions and performance metrics, and compared it to alternative solutions—including neuroevolutionary and manual design approaches. We conducted experiments in both simulation and with physical robots, using statistical tools to draw clear conclusions about the relative performance of the methods. This effort resulted in what is, to the best of our knowledge, the most diverse compendium of collective behaviors achieved through the automatic design of robot swarms, whether using AutoMoDe or other approaches. We consider this to be a significant contribution to the state of the art in swarm robotics. The core factor in achieving this result was the versatility and flexibility of AutoMoDe, which allowed us to leverage various forms of inter-robot and environmental signaling to address a wide range of design problems in the automatic design of robot swarms.

In Chapter 3, we presented `TuttiFrutti` and our first study on the design of collective behaviors for robots that can display and perceive color signals. We showed that `TuttiFrutti` was capable of establishing an appropriate relationship between the colors that robots perceive and the behaviors they must adopt. Experiments with `TuttiFrutti` showed that AutoMoDe can design collective behaviors with three desirable capabilities if color signaling is enabled in the modules on which it operates. First, robots can use the colors that they perceive as a reference to navigate the environment. Second, the swarm can collectively change its behavior when a specific color signal appears. Third, the swarm can exhibit communication behaviors in which robots pair the color signals they emit and the colors to which they react. In our experiments, the relationship between the environmental and inter-robot signaling and the behavior change was established on a per-mission basis,

and responded to the specifications of the mission at hand. With `TuttiFrutti`, we have greatly increased the variety of collective behaviors that AutoMoDe can generate. This is an original contribution from the thesis.

From Chapter 3, we identified a key question to be addressed in future work: *How can AutoMoDe and the robots' signaling capabilities be used to support ongoing human-swarm interaction research?* During the realization of all experiments presented in this dissertation, the color signaling mechanism helped monitor the state of the robots and interpret the behavior of the swarm. A promising research direction is to explore how the swarm can intentionally and automatically use this signaling to communicate with human operators. AutoMoDe's ability to discover effective interaction dynamics could facilitate the development of suitable human-swarm interaction strategies, using simple visual signaling that operators can directly perceive. In the context of human-swarm interaction research, AutoMoDe could allow the swarm to actively define a communication protocol with a human, finding ways to be understood rather than relying on the human to program when and how the swarm communicates information.

In Chapter 4, we built on the ideas introduced with `TuttiFrutti` and presented `Mandarina` along with our research on the design of robot swarms under concurrent design criteria. We used `Mandarina` to address design problems in which the mission the swarm must perform is specified as a set of independent objective functions to be optimized concurrently. In particular, we investigated the design of robot swarms that use environmental signals to transition between behaviors and perform missions sequentially. The traditional approach to addressing multi-criteria design problems in swarm robotics is to aggregate the design criteria into a single performance measure. In `Mandarina`, on the contrary, we used Iterated F-race to conduct the design process without aggregating the design criteria. Our experiments showed that this approach outperforms the traditional approach, including methods that use the weighted sum, the hypervolume, and the l^2-norm.

The literature revised in Chapter 2 shows that multi-criteria design problems are common and interesting to the swarm robotics community. In fact, researchers often unintentionally specify missions that simultaneously express varied preferences and design criteria related to mission outcomes, robot behavior, or the design process itself. However, in the related studies, the researchers neglect the multi-criteria nature of the design problem and the possible benefit of using multi-objective optimization methods to address it. Our study on bi-criteria design presents elements that can bootstrap further research into the broader multi-criteria design of robot swarms using optimization-based techniques. More precisely, we provided

here (i) an experimental framework for specifying bi-criteria missions, which can be extended to consider additional criteria; (ii) a set of baseline approaches—-the weighted sum, the hypervolume, and the l^2-norm; (iii) empirical results regarding popular automatic design approaches—modular design and neuroevolution; and (iii) we demonstrate the feasibility of this automatic design processes with both simulations and physical robots. This is also an original contribution of the thesis.

From Chapter 4, we identified two key questions to be addressed in future work: *What is the most suitable way to compare design methods effectively?* Our study with `Mandarina` considered design methods that rely on optimization processes driven by different performance measures. We noticed that the expected performance of a method compared to others can vary depending on the metric used to assess the generated control software instances. For this reason, we decided to analyze and compare the methods with respect to all performance measures used in the optimization processes. However, a better protocol should be defined to compare the methods in a fair way. *To which extent `Mandarina`'s capability to address multi-criteria design problems generalize beyond the bi-objective case?* In this dissertation, we focused on missions specified by two concurrent design criteria. However, in practice, `Mandarina` could handle additional criteria as long as each is provided to Iterated F-race as an independent problem instance to be considered in the optimization process. Considering more concurrent criteria during the design process can lead to a greater number of performance trade-offs and potentially make Iterated F-race's statistical comparison of control software less effective. Future research should expand the experimental setup to investigate this issue.

In Chapter 5, we introduced four design methods—`Mate`, `Habanero`, `Pistacchio`, and DTF-`MO`—demonstrating how AutoMoDe can leverage inter-robot and environmental signaling across a variety of design problems. With `Mate`, we showed that a simple single-bit signaling protocol embedded in a specialized module for AutoMoDe was sufficient to overcome previous limitations in designing spatially organizing behaviors for robot swarms. With `Habanero`, we showed that the direct communication capabilities developed with `TuttiFrutti` using color signals could also be applied to indirect communication using pheromone-based stigmergy. With `Pistacchio`, we showed that AutoMoDe's capabilities to leverage signaling not only facilitate communication within the swarm but also enable interactions with other active agents that populate the swarm's workspace. DTF-`MO` expanded on our previous research by removing the need for a preexisting performance measure to guide the design process, showing that AutoMoDe can handle problems with concurrent design criteria based solely on demonstrations. Our approach to

developing and evaluating these methods followed the practices originally defined by Francesca et al. (2014b, 2015) in the seminal work on AutoMoDe. We addressed these diverse design problems with AutoMoDe methods that had no fundamental differences from each other. An original contribution of the thesis is to have demonstrated this flexibility and versatility of AutoMoDe.

From Chapter 5, we also identified two key questions to be addressed in future work: *Can AutoMoDe design robot swarms that use both direct and indirect communication, either alternately or simultaneously?* With `TuttiFrutti`, we showed that direct communication can emerge from an automatic design process, and with `Habanero`, we showed that indirect communication can also be designed automatically. An interesting direction for future research is to explore whether AutoMoDe can design collective behaviors where robots use a combination of these two communication methods, or whether it can select between them on a per-mission basis. *Can AutoMoDe design collective behaviors that enable swarms to identify and interact with cooperative, non-cooperative, or neutral external agents?* In the heterogeneous setup we studied with `Pistacchio`, the sheep neither acted with intent nor aimed to maximize their own performance measure. A more realistic scenario would involve external agents driven by inference and planning. Future work should focus on how AutoMoDe can discover and design appropriate interaction dynamics between the swarm and such more complex agents.

The dissertation presented here, along with that of Francesca (2017), Ligot (2023), Hasselmann (2023), Kuckling (2023a), and Salman (2024), has shown that a consistent empirical framework can be used to systematically develop, test, and compare new automatic design methods across different design problems and platforms. We demonstrated this in the context of AutoMoDe, building a cumulative knowledge base that resulted from collaboration between multiple researchers. We believe that the growing scientific community in the automatic design of robot swarms can benefit from similar coordinated work. Future efforts should also be devoted to organizing the scientific community to collectively review current approaches, explore new directions, and establish well-defined engineering methodologies to design robot swarms via optimization. These efforts could be structured around working groups in professional societies or possibly through the establishment of an interinstitutional doctoral network.

A robot swarm is a group of robots that, by working together, can collectively perform missions beyond their individual capabilities. We, as swarm roboticists, must also fully leverage this powerful approach to build a mature engineering discipline for the automatic design of robot swarms. In this thesis, we show the

swarm robotics community how to use AutoMoDe to advance the design of robot swarms by fully automating the generation of communication-based collective behaviors.

Bibliography

Abbeel, P. and Ng, A. Y. (2004). "Apprenticeship learning via inverse reinforcement learning". In: *ICML'04: Proceedings of the 21th Annual International Conference on Machine Learning*. ACM, p. 1.

Abbeel, P. and Ng, A. Y. (2010). "Inverse reinforcement learning". In: *Encyclopedia of Machine Learning*. Ed. by C. Sammut and G. I. Webb. Springer Handbooks. Springer, pp. 554–558.

Albrecht, S. V., Christianos, F., and Schäfer, L. (2024). *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press.

Alfeo, A. L., Castelló Ferrer, E., Lizarribar Carrillo, Y., Grignard, A., Alonso Pastor, L., Sleeper, D. T., Cimino, M. G. C. A., Lepri, B., Vaglini, G., Larson, K., Dorigo, M., and Pentland, A. (2019). "Urban swarms: a new approach for autonomous waste management". In: *2019 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 4233–4240.

Alhafnawi, M., Hunt, E. R., Lemaignan, S., O'Dowd, P., and Hauert, S. (2022). "MOSAIX: a swarm of robot tiles for social human-swarm interaction". In: *2022 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 6882–6888.

Alharthi, K., Abdallah, Z. S., and Hauert, S. (2022). "Automatic extraction of understandable controllers from video observations of swarm behaviors". In: *Swarm Intelligence: 13th International Conference, ANTS 2022*. Ed. by M. Dorigo, H. Hamann, M. López-Ibáñez, J. García-Nieto, A. P. Engelbrecht, C. Pinciroli, V. Strobel, and C. L. Camacho Villalón. Vol. 13491. Springer, pp. 41–53.

Allwright, M., Bhalla, N., El-faham, H., Antoun, A., Pinciroli, C., and Dorigo, M. (2014). "SRoCS: leveraging stigmergy on a multi-robot construction platform for unknown environments". In: *Swarm Intelligence: 9th International Conference,*

ANTS 2014. Ed. by M. Dorigo, M. Birattari, S. Garnier, H. Hamann, M. Montes de Oca, C. Solnon, and T. Stützle. Vol. 8667. Lecture Notes in Computer Science. Springer, pp. 158–169.

Allwright, M., Zhu, W., and Dorigo, M. (2019). "An open-source multi-robot construction system". In: *HardwareX* 5, e00050.

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.

Antoun, A., Valentini, G., Hocquard, E., Wiandt, B., Trianni, V., and Dorigo, M. (2016). "Kilogrid: a modular virtualization environment for the Kilobot robot". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 3809–3814.

Aswale, A. and Pinciroli, C. (2023). "Heterogeneous coalition formation and scheduling with multi-skilled robots". In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 5402–5409.

Baker, M. (2016). "1,500 scientists lift the lid on reproducibility". In: *Nature* 533.7604, pp. 452–454.

Balaprakash, P., Birattari, M., and Stützle, T. (2007). "Improvement strategies for the F-Race algorithm: sampling design and iterative refinement". In: *Hybrid Metaheuristics: 4th International Workshop, HM 2007*. Ed. by T. Bartz-Beielstein, M. J. Blesa, C. Blum, B. Naujoks, A. Roli, G. Rudolph, and M. Sampels. Vol. 4771. Lecture Notes in Computer Science. Springer, pp. 108–122.

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*. Ed. by M. Marjan. IGI Global, pp. 436–501.

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

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.

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.

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*. Ed. by W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. K. Burke, and N. Jonoska. Morgan Kaufmann Publishers, pp. 11–18.

Birattari, M., Yuan, Z., Balaprakash, P., and Stützle, T. (2010). "F-Race and Iterated F-Race: an overview". In: *Experimental Methods for the Analysis of Optimization Algorithms*. Ed. by T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss. Springer, pp. 311–336.

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*. Ed. by M. Dorigo, M. Birattari, X. Li, M. López-Ibáñez, K. Ohkura, C. Pinciroli, and T. Stützle. 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 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., 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*. Ed. by N. Pillay and R. Qu. Natural Computing Series. Springer, pp. 73–90.

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

Bloom, J., Paliwal PranjalMukherjee, A., and Pinciroli, C. (2023). "Decentralized multi-agent reinforcement learning with global state prediction". In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 8854–8861.

Borenstein, J. and Koren, Y. (1989). "Real-time obstacle avoidance for fast mobile robots". In: *IEEE Transactions on Systems, Man, and Cybernetics* 19.5, pp. 1179–1187.

Bozhinoski, D. and Birattari, M. (2018). "Designing control software for robot swarms: software engineering for the development of automatic design methods". In: *RoSE'18: Proceedings of the 1st International Workshop on Robotics Software Engineering*. ACM, pp. 33–35.

Bozhinoski, D. and Birattari, M. (2022). "Towards an integrated automatic design process for robot swarms". In: *Open Research Europe* 1, p. 112.

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.

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.

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

Brutschy, A., Pini, G., and Decugnière, A. (2012). *Grippable objects for the foot-bot*. Tech. rep. TR/IRIDIA/2012-001. IRIDIA, Université Libre de Bruxelles.

Brutschy, A., Garattoni, L., Brambilla, M., Francesca, G., Pini, G., Dorigo, M., and Birattari, M. (2015). "The TAM: abstracting complex tasks in swarm robotics research". In: *Swarm Intelligence* 9.1, pp. 1–22.

Campo, A., Gutiérrez, Á., Nouyan, S., Pinciroli, C., Longchamp, V., Garnier, S., and Dorigo, M. (2010). "Artificial pheromone for path selection by a foraging swarm of robots". In: *Biological Cybernetics* 103.5, pp. 339–352.

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*. Ed. by J. E. Fieldsend. ACM, pp. 100–103.

Carrillo-Zapata, D., Milner, E., Hird, J., Tzoumas GeorgiosVardanega, P. J., Sooriyabandara, M., Giuliani, M., Winfield, A., and Hauert, S. (2020). "Mutual shaping in swarm robotics: user studies in fire and rescue, storage organization, and bridge inspection". In: *Frontiers in Robotics and AI* 7, p. 53.

Castelló Ferrer, E., Yamamoto, T., Dalla Libera, F., Liu, W., Winfield, A., Nakamura, Y., and Ishiguro, H. (2016). "Adaptive foraging for simulated and real robotic swarms: the dynamical response threshold approach". In: *Swarm Intelligence* 10.1, pp. 1–31.

Castelló Ferrer, E., Hardjono, T., Pentland, A., and Dorigo, M. (2021). "Secure and secret cooperation in robot swarms". In: *Science Robotics* 6.56, eabf1538.

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

Chen, J., Gauci, M., Li, W., Kolling, A., and Groß, R. (2015). "Occlusion-based cooperative transport with a swarm of miniature mobile robots". In: *IEEE Transactions on Robotics* 31.2, pp. 307–321.

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*. Ed. by L. M. Rocha, L. S. Yaeger, M. A. Bedau, D. Floreano, R. L. Goldstone, and A. Vespignani. MIT Press, pp. 248–254.

Christensen, A. L., O'Grady, R., and Dorigo, M. (2009). "From fireflies to fault-tolerant swarms of robots". In: *IEEE Transactions on Evolutionary Computation* 13.4, pp. 754–766.

Colledanchise, M. and Ögren, P. (2018). *Behavior Trees in Robotics and AI: An Introduction*. Chapman & Hall/CRC Artificial Intelligence and Robotics Series. CRC Press.

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

Das, A., Gervet, T., Romoff, J., Batra, D., Parikh, D., Rabbat, M., and Pineau, J. (2019). "TarMAC: targeted multi-agent communication". In: *ICML 2019: Proceedings of the 36th Annual International Conference on Machine Learning*. ACM, pp. 1538–1546.

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.

Doncieux, S., Bredeche, N., Mouret, J.-B., and Eiben, A. (2015). "Evolutionary robotics: what, why, and where to". In: *Frontiers in Robotics and AI* 2, p. 4.

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.

Dorigo, M. and Birattari, M. (2007). "Swarm intelligence". In: *Scholarpedia* 2.9, p. 1462.

Dorigo, M. et al. (2013). "Swarmanoid: a novel concept for the study of heterogeneous robotic swarms". In: *IEEE Robotics & Automation Magazine* 20.4, pp. 60–71.

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.

Dosieah, G. Y., Özdemir, A., Gauci, M., and Groß, R. (2022). "Moving mixtures of active and passive elements with robots that do not compute". In: *Swarm Intelligence: 13th International Conference, ANTS 2022*. Vol. 13491. Lecture Notes in Computer Science. Springer, pp. 183–195.

Duarte, M., Oliveira, S. M., and Christensen, A. L. (2014). "Hybrid control for large swarms of aquatic drones". In: *ALIFE 14: The Fourteenth International Conference on the Synthesis and Simulation of Living Systems*. Ed. by H. Sayama, J. Rieffel, S. Risi, R. Doursat, and H. Lipson. MIT Press, pp. 785–792.

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

Ducatelle, F., Di Caro, G. A., Pinciroli, C., and Gambardella, L. M. (2011). "Self-organized cooperation between robotic swarms". In: *Swarm Intelligence* 5, pp. 73–96.

Emmerich, M. T. M. and Deutz, A. H. (2018). "A tutorial on multiobjective optimization: fundamentals and evolutionary methods". In: *Natural Computing* 17.3, pp. 585–609.

Endo, W., Baumann, C., Asama, H., and Martinoli, A. (2023). "Automatic multi-robot control design and optimization leveraging multi-level modeling: an exploration case study". In: *IFAC-PapersOnLine* 56.2, pp. 11462–11469.

Feola, L., Reina, A., Talamali, M. S., and Trianni, V. (2023). "Multi-swarm interaction through augmented reality for Kilobots". In: *IEEE Robotics and Automation Letters* 8.11, pp. 6907–6914.

Ferrante, E., Turgut, A. E., Mathews, N., Birattari, M., and Dorigo, M. (2010). "Flocking in stationary and non-stationary environments: a novel communication strategy for heading alignment". In: *Parallel Problem Solving from Nature – PPSN XI: 11th International Conference*. Ed. by R. Schaefer, C. Cotta, J. Kołodziej, and G. Rudolph. Vol. 6239. Lecture Notes in Computer Science. Springer, pp. 331–340.

Ferrante, E., Turgut, A. E., Huepe, C., Stranieri, A., Pinciroli, C., and Dorigo, M. (2012). "Self-organized flocking with a mobile robot swarm: a novel motion control method". In: *Adaptive Behavior* 20.6, pp. 460–477.

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*. Ed. by C. Blum. 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.

Fishburn, P. C. (1970). *Utility Theory for Decision Making*. Publications in Operations Research. John Wiley & Sons.

Floreano, D., Mitri, S., Magnenat, S., and Keller, L. (2007). "Evolutionary conditions for the emergence of communication in robots". In: *Current Biology* 17.6, pp. 514–519.

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

Foerster, J., Assael, I. A., De Freitas, N., and Whiteson, S. (2016). "Learning to communicate with deep multi-agent reinforcement learning". In: *NeurIPS'16: Proceedings of the 29th International Conference on Neural Information Processing Systems*. Ed. by D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett. Vol. 29. Curran Associates, Inc.

Fonseca, C. M., Paquete, L., and López-Ibáñez, M. (2006). "An improved dimension-sweep algorithm for the hypervolume indicator". In: *2006 IEEE Congress on Evolutionary Computation*. IEEE, pp. 1157–1163.

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*. Ed. by M. Dorigo, M. Birattari, S. Garnier, H. Hamann, M. Montes de Oca, C. Solnon, and T. Stützle. 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., 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. 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. (2017). "A modular approach to the automatic design of control software for robot swarms: from a novel perspective on the reality gap to AutoMoDe". PhD thesis. Université Libre de Bruxelles.

Fujisawa, R., Dobata, S., Sugawara, K., and Matsuno, F. (2014). "Designing pheromone communication in swarm robotics: group foraging behavior mediated by chemical substance". In: *Swarm Intelligence* 8.3, pp. 227–246.

Garnier, S., Combe, M., Jost, C., and Theraulaz, G. (2013). "Do ants need to estimate the geometrical properties of trail bifurcations to find an efficient route?: a swarm robotics test bed". In: *PLOS Computational Biology* 9.3, e1002903.

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.

Garattoni, L. and Birattari, M. (2016). "Swarm robotics". In: *Wiley Encyclopedia of Electrical and Electronics Engineering*. Ed. by J. G. Webster. 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.

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., Bolaños, J. P., Diaz, J., Pachajoa, G., and Birattari, M. (2021a). "Introduciendo la robótica de enjambres a entusiastas de la robótica: experiencias y resultados de una colaboración académica". In: *I Congreso Internacional de la Sociedad de Doctores e Investigadores de Colombia (SOPHIC 2021): la ciencia al servicio de la sociedad*. Ed. by E. Ramírez Vargas, L. A. Pedraza Herrera, V. Otero Jiménez, and S. L. Leiva Maldonado. Editorial SoPhIC, pp. 46–48.

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. (2021b). "The automatic off-line design of robot swarms: recent advances and perspectives". In: *R2T2: Robotics Research for Tomorrow's Technology*. Ed. by G. De Masi, E. Ferrante, and P. Dario.

Garzón Ramos, D., Salman, M., Ubeda Arriaza, K., Hasselmann, K., and Birattari, M. (2022). *MoCA: a modular RGB color arena for swarm robotics experiments*. Tech. rep. TR/IRIDIA/2022-014. IRIDIA, Université Libre de Bruxelles.

Garzón Ramos, D. and Birattari, M. (2024). "Automatically designing robot swarms in environments populated by other robots: an experiment in robot shepherding". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 12240–12247.

Garzón Ramos, D., Pagnozzi, F., Stützle, T., and Birattari, M. (2024). "Automatic design of robot swarms under concurrent design criteria: a study based on Iterated F-Race". In: *Advanced Intelligent Systems*, p. 2400332.

Garzón Ramos, D. (2025). *Leveraging environmental and inter-robot signaling in the automatic modular design of robot swarms: communication, reaction to events, and sequential missions – Supplementary Videos*. `https://dgarzonramos.com/posts/Gar2025phd.html`.

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.

Genter, K. and Stone, P. (2014). "Influencing a flock via ad hoc teamwork". In: *Swarm Intelligence: 9th International Conference, ANTS 2014*. Ed. by M. Dorigo, M. Birattari, S. Garnier, H. Hamann, M. Montes de Oca, C. Solnon, and T. Stützle. Vol. 8667. Lecture Notes in Computer Science. Springer International Publishing, pp. 110–121.

Genter, K. and Stone, P. (2016). "Adding influencing agents to a flock". In: *AAMAS '16: Proceedings of the 2016 International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), pp. 615–623.

Gendreau, M. and Potvin, J.-Y., eds. (2019). *Handbook of Metaheuristics*. International Series in Operations Research & Management Science. Springer.

Gharbi, I., Kuckling, J., Garzón Ramos, D., and Birattari, M. (2023). "Show me what you want: inverse reinforcement learning to automatically design robot swarms by demonstration". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 5063–5070.

Giusti, A., Nagi, J., Gambardella, L. M., and Di Caro, G. A. (2012). "Distributed consensus for interaction between humans and mobile robot swarms (demonstration)". In: *AAMAS '12: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems – Volume 3*. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), pp. 1503–1504.

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

Gomes, J. and Christensen, A. L. (2018). "Task-agnostic evolution of diverse repertoires of swarm behaviours". In: *Swarm Intelligence: 11th International Conference, ANTS 2018*. Ed. by M. Dorigo, M. Birattari, C. Blum, A. L. Christensen, A. Reina, and V. Trianni. Vol. 11172. Lecture Notes in Computer Science. Springer, pp. 225–238.

Grassé, P.-P. (1959). "La reconstruction du nid et les coordinations interindividuelles chez Bellicositermes natalensis et Cubitermes sp. la théorie de la stigmergie: essai d'interprétation du comportement des termites constructeurs". In: *Insectes Sociaux: International Journal for the Study of Social Arthropods* 6.1, pp. 41–80.

Guerreiro, A. P., Fonseca, C. M., and Paquete, L. (2021). "The hypervolume indicator: computational problems and algorithms". In: *ACM Computing Surveys* 54.6.

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)*. Ed. by K. Kosuge. IEEE, pp. 3111–3116.

Hamann, H., Pinciroli, C., and Mammen, S. von (2018). "A gamification concept for teaching swarm robotics". In: *2018 12th European Workshop on Microelectronics Education (EWME)*. Ed. by J. Haase. IEEE, pp. 83–88.

Hamann, H., Schranz, M., Elmenreich, W., Trianni, V., Pinciroli, C., Bredeche, N., and Ferrante, E. (2020). "Editorial: designing self-organization in the physical realm". In: *Frontiers in Robotics and AI* 7, p. 164.

Hamann, H. and Reina, A. (2021). "Scalability in computing and robotics". In: *IEEE Transactions on Computers*.

Hamann, H. (2012). "Towards swarm calculus: universal properties of swarm performance and collective decisions". In: *Swarm Intelligence: 8th International Conference, ANTS 2012*. Ed. by M. Birattari, C. Blum, A. L. Christensen, A. P. Engelbrecht, R. Groß, M. Dorigo, and T. Stützle. Vol. 7461. Lecture Notes in Computer Science. Springer, pp. 168–179.

Hamann, H. (2014). "Evolution of collective behaviors by minimizing surprise". In: *ALIFE 14: The Fourteenth International Conference on the Synthesis and Simulation of Living Systems*. Ed. by H. Sayama, J. Rieffel, S. Risi, R. Doursat, and H. Lipson. MIT Press, pp. 344–351.

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., 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., Robert, F., and Birattari, M. (2018b). "Automatic design of communication-based behaviors for robot swarms". In: *Swarm Intelligence: 11th International Conference, ANTS 2018*. Ed. by M. Dorigo, M. Birattari, S. Garnier, H. Hamann, M. Montes de Oca, C. Solnon, and T. Stützle. Vol. 11172. Lecture Notes in Computer Science. Springer, pp. 16–29.

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., 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., Ligot, A., and Birattari, M. (2023). "Automatic modular design of robots swarms based on repertoires of behaviors generated via novelty search". In: *Swarm and Evolutionary Computation* 83, p. 101395.

Hasselmann, K. (2023). "Advances in the automatic modular design of control software for robot swarms: using neuroevolution to generate modules". PhD thesis. Université Libre de Bruxelles.

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.

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*. Ed. by M. Dorigo, M. Birattari, C. Blum, A. L. Christensen, A. P. Engelbrecht, R. Groß, and T. Stützle. Vol. 7461. Lecture Notes in Computer Science. Springer, pp. 252–259.

Hettiarachchi, S. and Spears, W. M. (2005). "Moving swarm formations through obstacle fields". In: *Proceedings of the 2005 International Conference on Artificial Intelligence, ICAI 2005*. Vol. 1. CSREA Press, pp. 97–103.

Hettiarachchi, S. and Spears, W. M. (2006). "DAEDALUS for agents with obstructed perception". In: *2006 IEEE Mountain Workshop on Adaptive and Learning Systems*. IEEE, pp. 195–200.

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.

Heuthe, V.-L., Panizon, E., Gu, H., and Bechinger, C. (2024). "Counterfactual rewards promote collective transport using individually controlled swarm microrobots". In: *Science Robotics* 9.97, eado5888.

Heylighen, F. (2016a). "Stigmergy as a universal coordination mechanism I: definition and components". In: *Cognitive Systems Research* 38, pp. 4–13.

Heylighen, F. (2016b). "Stigmergy as a universal coordination mechanism II: varieties and evolution". In: *Cognitive Systems Research* 38, pp. 50–59.

Howard, A., Matarić, M. J., and Sukhatme, G. S. (2002). "Mobile sensor network deployment using potential fields: a distributed, scalable solution to the area coverage problem". In: *Distributed Autonomous Robotic Systems 5*. Ed. by H. Asama, T. Fukuda, and T. Hasegawa. Springer, pp. 299–308.

Hu, J., Turgut, A. E., Krajník, T., Lennox, B., and Arvin, F. (2020). "Occlusion-based coordination protocol design for autonomous robotic shepherding tasks". In: *IEEE Transactions on Cognitive and Developmental Systems*, p. 1.

Hunt, E. R., Jones, S., and Hauert, S. (2019). "Testing the limits of pheromone stigmergy in high-density robot swarms". In: *Royal Society Open Science* 6.11, p. 190225.

Hunt, E. R. and Hauert, S. (2020). "A checklist for safe robot swarms". In: *Nature Machine Intelligence* 2, pp. 420–422.

Hussein, A., Gaber, M. M., Elyan, E., and Jayne, C. (2017). "Imitation learning: a survey of learning methods". In: *ACM Compututing Surveys* 50.2, p. 21.

Hüttenrauch, M., Šošić, A., and Neumann, G. (2019). "Deep reinforcement learning for swarm systems". In: *Journal of Machine Learning Research* 20.1, pp. 1966–1996.

Ijspeert, A. J., Martinoli, A., Billard, A., and Gambardella, L. M. (2001). "Collaboration through the exploitation of local interactions in autonomous collective robotics: the stick pulling experiment". In: *Autonomous Robots* 11.2, pp. 149–171.

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*. Ed. by F. Morán, A. Moreno, J. J. Merelo, and P. Chacón. Vol. 929. Lecture Notes in Artificial Intelligence. Springer, pp. 704–720.

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

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*. Ed. by R. Groß, A. Kolling, S. Berman, E. Frazzoli, A. Martinoli, F. Matsuno, and M. Gauci. 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.

Jones, S., Milner, E., Sooriyabandara, M., and Hauert, S. (2020). "Distributed situational awareness in robot swarms". In: *Advanced Intelligent Systems* 2.11, p. 2000110.

Jones, S., Milner, E., Sooriyabandara, M., and Hauert, S. (2022). *DOTS: an open testbed for industrial swarm robotic solutions*. `https://arxiv.org/abs/2203.13809`.

Jones, J. E. (1924). "On the determination of molecular fields". In: *Proceedings of the Royal Society of London* 106, pp. 463–477.

Kaiser, T. K. and Hamann, H. (2019). "Engineered self-organization for resilient robot self-assembly with minimal surprise". In: *Robotics and Autonomous Systems* 122, p. 103293.

Kaiser, T. K. and Hamann, H. (2022). "Innate motivation for robot swarms by minimizing surprise: from simple simulations to real-world experiments". In: *IEEE Transactions on Robotics* 38.6, pp. 3582–3601.

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

Kegeleirs, M., Garzón Ramos, D., and Birattari, M. (2019). "Random walk exploration for swarm mapping". In: *Towards Autonomous Robotic Systems: 20th Annual Conference, TAROS 2019*. Ed. by K. Althoefer, J. Konstantinova, and K. Zhang. Vol. 11650. Lecture Notes in Computer Science. Springer, pp. 211–222.

Kegeleirs, M., Todesco, R., Garzón Ramos, D., Legarda Herranz, G., and Birattari, M. (2022). *Mercator: hardware and software architecture for experiments in swarm SLAM*. Tech. rep. TR/IRIDIA/2022-012. IRIDIA, Université Libre de Bruxelles.

Kegeleirs, M., Garzón Ramos, D., Garattoni, L., Francesca, G., and Birattari, M. (2023). "Automatic off-line design of robot swarms: exploring the transferability of control software and design methods across different platforms". In: *ICRA 2023 Transferability in Robotics Workshop*. EU Horizon project euRobin.

Kegeleirs, M., Garzón Ramos, D., and Birattari, M. (2024a). "DeimOS: a ROS-ready operating system for the e-puck". In: *Journal of Open Research Software*, (accepted).

Kegeleirs, M., Garzón Ramos, D., Hasselmann, K., Garattoni, L., Francesca, G., and Birattari, M. (2024b). "Transferability in the automatic off-line design of robot swarms: from sim-to-real to embodiment and design-method transfer across different platforms". In: *IEEE Robotics and Automation Letters* 9.3, pp. 2758–2765.

Kegeleirs, M., Garzón Ramos, D., Legarda Herranz, G., Gharbi, I., Szpirer, J., Debeir, O., Garattoni, L., Francesca, G., and Birattari, M. (2024c). "Collective perception for tracking people with a robot swarm". In: *40th Anniversary of the IEEE Conference on Robotics and Automation (ICRA@40)*, (accepted).

Kegeleirs, M., Garzón Ramos, D., Legarda Herranz, G., Gharbi, I., Szpirer, J., Hasselmann, K., Garattoni, L., Francesca, G., and Birattari, M. (2024d). "Leveraging swarm capabilities to assist other systems". In: *ICRA 2024 Breaking Swarm Stereotypes Workshop*. EU EIC project EMERGE.

Kellogg, J., Bovais, C., Dahlburg, J., Foch, R. J., Gardner, J. H., Gordon, D. F., Hartley, R. L., Kamgar-Parsi, B., Mcfarlane, H., Pipitone, F., Ramamurti, R., Sciambi, A., Spears, W. M., Srull, D., and Sullivan, C. (2002). "The NRL micro tactical expendable (MITE) air vehicle". In: *The Aeronautical Journal*, pp. 431–441.

Khaliq, A. A., Di Rocco, M., and Saffiotti, A. (2014). "Stigmergic algorithms for multiple minimalistic robots on an RFID floor". In: *Swarm Intelligence* 8.3, pp. 199–225.

Khatib, O. (1986). "Real-time obstacle avoidance for manipulators and mobile robots". In: *The International Journal of Robotics Research* 5.1, pp. 90–98.

King, A. J., Portugal, S. J., Strömbom, D., Mann, R. P., Carrillo, J. A., Kalise, D., Croon, G. de, Barnett, H., Scerri, P., Groß, R., Chadwick, D. R., and Papadopoulou, M. (2023). "Biologically inspired herding of animal groups by robots". In: *Methods in Ecology and Evolution* 14.2, pp. 478–486.

Kirkpatrick, S., Gelatt Jr., C. D., and Vecchi, M. P. (1983). "Optimization by simulated annealing". In: *Science* 220.4598, pp. 671–680.

Kolling, A., Walker, P., Chakraborty, N., Sycara, K., and Lewis, M. (2016). "Human interaction with robot swarms: a survey". In: *IEEE Transactions on Human-Machine Systems* 46.1, pp. 9–26.

Koza, J. R. (1992). *Genetic programming: on the programming of computers by means of natural selection*. MIT Press.

Krieger, M. J. B., Billeter, J.-B., and Keller, L. (2000). "Ant-like task allocation and recruitment in cooperative robots". In: *Nature* 406, pp. 992–995.

Kuckling, J., Ligot, A., Bozhinoski, D., and Birattari, M. (2018a). "Behavior trees as a control architecture in the automatic modular design of robot swarms". In: *Swarm Intelligence: 11th International Conference, ANTS 2018*. Ed. by M. Dorigo, M. Birattari, C. Blum, A. L. Christensen, A. Reina, and V. Trianni. Vol. 11172. Lecture Notes in Computer Science. Springer, pp. 30–43.

Kuckling, J., Ligot, A., Bozhinoski, D., and Birattari, M. (2018b). *Search space for AutoMoDe-Chocolate and AutoMoDe-Maple*. Tech. rep. TR/IRIDIA/2018-012. IRIDIA, Université Libre de Bruxelles.

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*. Ed. by K. Beuls, B. Bogaerts, G. Bontempi, P. Geurts, N. Harley, B. Lebichot, T. Lenaerts, G. Louppe, and P. Van Eecke. Vol. 2491. CEUR Workshop Proceedings.

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. (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*. Ed. by B. Bogaerts, G. Bontempi, P. Geurts, N. Harley, B. Lebichot, T. Lenaerts, and G. Louppe. Vol. 1196. Communications in Computer and Information Science. Springer, pp. 3–17.

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

Kuckling, J., Pelt, V. van, and Birattari, M. (2021b). "Automatic modular design of behavior trees for robot swarms with communication capabilities". In: *Applications of Evolutionary Computation: 24th International Conference, EvoApplications 2021*. Ed. by P. A. Castillo and J. L. Jiménez Laredo. Vol. 12694. Lecture Notes in Computer Science. Springer, pp. 130–145.

Kuckling, J., Pelt, V. van, and Birattari, M. (2022). "AutoMoDe-Cedrata: automatic design of behavior trees for controlling a swarm of robots with communication capabilities". In: *SN Computer Science* 3, p. 136.

Kuckling, J., Luckey, R., Avrutin, V., Vardy, A., Reina, A., and Hamann, H. (2024). "Do we run large-scale multi-Robot systems on the edge? More evidence for two-phase performance in system size scaling". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 4562–4568.

Kuckling, J. (2023a). "Optimization in the automatic modular design ofcontrol software for robot swarms". PhD thesis. Université Libre de Bruxelles.

Kuckling, J. (2023b). "Recent trends in robot learning and evolution for swarm robotics". In: *Frontiers in Robotics and AI* 10, p. 1134841.

Labella, T. H., Dorigo, M., and Deneubourg, J.-L. (2006). "Division of labor in a group of robots inspired by ants' foraging behavior". In: *ACM Transactions on Autonomous Adaptive Systems*, pp. 4–25.

Lee, S., Milner, E., and Hauert, S. (2022). "A data-driven method for metric extraction to detect faults in robot swarms". In: *IEEE Robotics and Automation Letters* 7.4, pp. 10746–10753.

Legarda Herranz, G., Garzón Ramos, D., Kuckling, J., Kegeleirs, M., and Birattari, M. (2022). *Tycho: a robust, ROS-based tracking system for robot swarms*. Tech. rep. TR/IRIDIA/2022-009. IRIDIA, Université Libre de Bruxelles.

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

Lerman, K., Galstyan, A., Martinoli, A., and Ijspeert, A. J. (2001). "A macroscopic analytical model of collaboration in distributed robotic systems". In: *Artificial Life* 7.4, pp. 375–393.

Lerman, K. and Galstyan, A. (2002). "Mathematical model of foraging in a group of robots: effect of interference". In: *Autonomous Robots* 13.2, pp. 127–141.

Li, W., Gauci, M., and Groß, R. (2016). "Turing learning: a metric-free approach to inferring behavior and its application to swarms". In: *Swarm Intelligence* 10, pp. 211–243.

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.

Licitra, R. A., Bell, Z. I., and Dixon, W. E. (2019). "Single-agent indirect herding of multiple targets with uncertain dynamics". In: *IEEE Transactions on Robotics* 35.4, pp. 847–860.

Lien, J.-M., Bayazit, O. B., Sowell, R. T., Rodríguez, S., and Amato, N. M. (2004). "Shepherding behaviors". In: *2004 IEEE International Conference on Robotics and Automation (ICRA)*. Vol. 4. IEEE, pp. 4159–4164.

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., 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*. Ed. by M. Dorigo, T. Stützle, M. J. Blesa, C. Blum, H. Hamann, M. K. Heinrich, and V. Strobel. Vol. 12421. Lecture Notes in Computer Science. Springer, pp. 109–122.

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.

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

Ligot, A. (2023). "Assessing and forecasting the performance of optimization-based design methods for robot swarms: experimental protocol & pseudo-reality predictors". PhD thesis. Université Libre de Bruxelles.

Liu, W., Winfield, A., Sa, J., Chen, J., and Dou, L. (2007). "Towards energy optimization: emergent task allocation in a swarm of foraging robots". In: *Adaptive Behavior* 15.3, pp. 289–305.

Lochmatter, T., Aydın Göl, E., Navarro, I., and Martinoli, A. (2013). "A plume tracking algorithm based on crosswind formations". In: *Distributed Autonomous Robotic Systems: The 10th International Symposium*. Ed. by A. Martinoli, F. Mondada, N. Correll, G. Mermoud, M. Egerstedt, M. A. Hsieh, L. E. Parker, and K. Støy. Springer Tracts in Advanced Robotics. Springer, pp. 91–102.

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*. Ed. by M. Dorigo, M. Birattari, S. Garnier, H. Hamann, M. Montes de Oca, C. Solnon, and T. Stützle. 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.

Lopes, Y. K., Trenkwalder, S. M., Leal, A. B., Dodd, T. J., and Groß, R. (2017). "Probabilistic supervisory control theory (pSCT) applied to swarm robotics". In: *AAMAS '17: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), pp. 1395–1403.

López-Ibáñez, M., Pérez Cáceres, L., Dubois-Lacoste, J., Stützle, T., and Birattari, M. (2020). *The irace package: user guide*. IRIDIA, Université Libre de Bruxelles, Brussels, Belgium.

Marocco, D. and Nolfi, S. (2007). "Emergence of communication in embodied agents evolved for the ability to solve a collective navigation problem". In: *Connection Science* 19.1, pp. 53–74.

Marler, R. T. and Arora, J. S. (2010). "The weighted sum method for multi-objective optimization: new insights". In: *Structural and Multidisciplinary Optimization* 41.6, pp. 853–862.

Martinoli, A., Ijspeert, A. J., and Mondada, F. (1999). "Understanding collective aggregation mechanisms: from probabilistic modelling to experiments with real robots". In: *Robotics and Autonomous Systems* 29, pp. 51–63.

Mason, K. and Hauert, S. (2023). "Evolving multi-objective neural network controllers for robot swarms". In: *2023 Autonomous Robots and Multirobot Systems Workshop*. Ed. by N. Basilico, M. Sridharan, N. Agmon, F. Amigoni, J. Biswas, A. Farinelli, M. Gini, G. A. Kaminka, and D. Nardi.

Mathews, N., Christensen, A. L., O'Grady, R., Mondada, F., and Dorigo, M. (2017). "Mergeable nervous systems for robots". In: *Nature Communications* 8.1, p. 439.

Mathews, N., Christensen, A. L., Stranieri, A., Scheidler, A., and Dorigo, M. (2019). "Supervised morphogenesis: exploiting morphological flexibility of self-assembling multirobot systems through cooperation with aerial robots". In: *Robotics and Autonomous Systems* 112, pp. 154–167.

Matarić, M. J. (1997). "Reinforcement learning in the multi-robot domain". In: *Autonomous Robots* 4.1, pp. 73–83.

Maxim, P. M., Spears, W. M., and Spears, D. (2009). "Robotic chain formations". In: *IFAC Proceedings Volumes* 42.22, pp. 19–24.

Mayet, R., Roberz, J., Schmickl, T., and Crailsheim, K. (2010). "Antbots: a feasible visual emulation of pheromone trails for swarm robots". In: *Swarm Intelligence: 7th International Conference, ANTS 2010*. Ed. by M. Dorigo, M. Birattari, G. A. Di Caro, R. Doursat, A. P. Engelbrecht, D. Floreano, L. M. Gambardella, R. Groß, E. Şahin, H. Sayama, and T. Stützle. Vol. 6234. Lecture Notes in Computer Science. Springer, pp. 89–94.

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.

Miettinen, K. (1998). *Nonlinear Multiobjective Optimization*. Springer.

Milner, E., Sooriyabandara, M., and Hauert, S. (2023). *Swarm Performance Indicators: metrics for robustness, fault tolerance, scalability and adaptability*. `https://arxiv.org/abs/2311.01944`.

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*. Ed. by P. Gonçalves, P. Torres, and C. Alves. Instituto Politécnico de Castelo Branco, pp. 59–65.

Mondada, F., Franzi, E., and Ienne, P. (1997). "Mobile robot miniaturisation: a tool for investigation in control algorithms". In: *Experimental Robotics III:*

3rd International Symposium. Ed. by T. Yoshikawa and F. Miyazaki. Vol. 200. Lecture Notes in Control and Information Sciences. Springer, pp. 501–513.

Na, S., Raoufi, M., Turgut, A. E., Krajník, T., and Arvin, F. (2019). "Extended artificial pheromone system for swarm robotic applications". In: *ALIFE 2019: The 2019 Conference on Artificial Life*. MIT Press, pp. 608–615.

Na, S., Qiu, Y., Turgut, A. E., Ulrich, J., Krajník, T., Yue, S., Lennox, B., and Arvin, F. (2020). "Bio-inspired artificial pheromone system for swarm robotics applications". In: *Adaptive Behavior* 29.4, pp. 395–415.

Na, S., Niu, H., Lennox, B., and Arvin, F. (2022). "Bio-inspired collision avoidance in swarm systems via deep reinforcement learning". In: *IEEE Transactions on Vehicular Technology* 71.3, pp. 251–2526.

Nedjah, N. and Silva Junior, L. (2019). "Review of methodologies and tasks in swarm robotics towards standardization". In: *Swarm and Evolutionary Computation* 50, p. 100565.

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

Nouyan, S., Campo, A., and Dorigo, M. (2008). "Path formation in a robot swarm: self-organized strategies to find your way home". In: *Swarm Intelligence* 2.1, pp. 1–23.

Nouyan, S., Groß, R., Bonani, M., Mondada, F., and Dorigo, M. (2009). "Teamwork in self-organized robot colonies". In: *IEEE Transactions on Evolutionary Computation* 13.4, pp. 695–711.

O'Grady, R., Christensen, A. L., and Dorigo, M. (2009). "SWARMORPH: multirobot morphogenesis using directional self-assembly". In: *IEEE Transactions on Robotics* 25.3, pp. 738–743.

O'Grady, R., Groß, R., Christensen, A. L., and Dorigo, M. (2010). "Self-assembly strategies in a group of autonomous mobile robots". In: *Autonomous Robots* 28.4, pp. 439–455.

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

Osa, T., Pajarinen, J., Neumann, G., Bagnell, J. A., Abbeel, P., and Peters, J. (2018). "An algorithmic perspective on imitation learning". In: *Foundations and Trends in Robotics* 7.1–2, pp. 1–179.

Özdemir, A., Gauci, M., and Groß, R. (2017). "Shepherding with robots that do not compute". In: *ECAL 2017, the Fourteenth European Conference on Artificial Life*. MIT Press, pp. 332–339.

Pagnozzi, F. and Birattari, M. (2021). "Off-policy evaluation of the performance of a robot swarm: importance sampling to assess potential modifications to the finite-state machine that controls the robots". In: *Frontiers in Robotics and AI* 8, p. 55.

Parker, L. E., Rus, D., and Sukhatme, G. S. (2016). "Multiple mobile robot systems". In: *Springer Handbook of Robotics*. Ed. by B. Siciliano and O. Khatib. Springer Handbooks. Springer, pp. 1335–1384.

Payton, D., Daily, M., Estowski, R., Howard, M., and Lee, C. (2001). "Pheromone robotics". In: *Autonomous Robots* 11, pp. 319–324.

Pérez-Dattari, R., Della Santina, C., and Kober, J. (2024). "PUMA: deep metric imitation learning for stable motion primitives". In: *Advanced Intelligent Systems* 6.11, p. 2400144.

Pierson, A. and Schwager, M. (2018). "Controlling noncooperative herds with robotic herders". In: *IEEE Transactions on Robotics* 34.2, pp. 517–525.

Pinciroli, C., Birattari, M., Tuci, E., Dorigo, M., Rey Zapatero, M. del, Vinko, T., and Izzo, D. (2008a). "Lattice formation in space for a swarm of pico satellites". In: *Ant Colony Optimization and Swarm Intelligence: 6th International Conference, ANTS 2008*. Ed. by M. Dorigo, M. Birattari, C. Blum, M. Clerc, T. Stützle, and A. Winfield. Springer, pp. 347–354.

Pinciroli, C., Birattari, M., Tuci, E., Dorigo, M., Rey Zapatero, M. del, Vinko, T., and Izzo, D. (2008b). "Self-organizing and scalable shape formation for a swarm of pico satellites". In: *2008 NASA/ESA Conference on Adaptive Hardware and Systems*. IEEE, pp. 57–61.

Pinciroli, C., O'Grady, R., Christensen, A. L., and Dorigo, M. (2009). "Self-organised recruitment in a heteregeneous swarm". In: *2009 International Conference on Advanced Robotics*. IEEE, pp. 1–8.

Pini, G., Brutschy, A., Birattari, M., and Dorigo, M. (2011a). "Task partitioning in swarms of robots: reducing performance losses due to interference at shared resources". In: *Informatics in Control Automation and Robotics: Revised and Selected Papers from the International Conference on Informatics in Control Automation and Robotics 2009*. Ed. by J. Andrade Cetto, J. Filipe, and J.-L. Ferrier. Vol. 85. Lecture Notes in Electrical Engineering. Springer, pp. 217–228.

Pini, G., Brutschy, A., Frison, M., Roli, A., Dorigo, M., and Birattari, M. (2011b). "Task partitioning in swarms of robots: an adaptive method for strategy selection". In: *Swarm Intelligence* 5.3–4, pp. 283–304.

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.

Pini, G., Brutschy, A., Scheidler, A., Dorigo, M., and Birattari, M. (2014). "Task partitioning in a robot swarm: object retrieval as a sequence of subtasks with direct object transfer". In: *Artificial Life* 20.3, pp. 291–317.

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

Pitonakova, L., Giuliani, M., Pipe, A., and Winfield, A. (2018). "Feature and performance comparison of the V-REP, Gazebo and ARGoS robot simulators". In: *Towards Autonomous Robotic Systems: 19th Annual Conference, TAROS 2018*. Ed. by M. Giuliani, T. Assaf, and M. E. Giannaccini. Lecture Notes in Computer Science. Springer, pp. 357–368.

Podevijn, G., O'Grady, R., and Dorigo, M. (2012). "Self-organised feedback in human swarm interaction". In: *Proceedings of the workshop on robot feedback in human-robot interaction: how to make a robot readable for a human interaction partner, Ro-Man 2012*.

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.

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.

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.

Reina, A., Cope, A. J., Nikolaidis, E., Marshall, J. A. R., and Sabo, C. (2017). "ARK: augmented reality for Kilobots". In: *IEEE Robotics and Automation Letters* 2.3, pp. 1755–1761.

Reina, A., Zakir, R., De Masi, G., and Ferrante, E. (2023). "Cross-inhibition leads to group consensus despite the presence of strongly opinionated minorities and asocial behaviour". In: *Communications Physics* 6, p. 236.

Riedo, F., Chevalier MorganeMagnenat, S., and Mondada, F. (2013). "Thymio II: a robot that grows wiser with children". In: *2013 IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO)*. IEEE, pp. 187–193.

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

Russell, R. A. (1997). "Heat trails as short-lived navigational markers for mobile robots". In: *1997 IEEE International Conference on Robotics and Automation (ICRA97)*. Vol. 4. IEEE, pp. 3534–3539.

Russell, R. A. (1999). "Ant trails – an example for robots to follow?" In: *1999 IEEE International Conference on Robotics and Automation, ICRA'99*. Vol. 4. IEEE, pp. 2698–2703.

Sadeghi Amjadi, A., Bilaloğlu, C., Turgut, A. E., Na, S., Şahin, E., Krajník, T., and Arvin, F. (2024). "Reinforcement learning-based aggregation for robot swarms". In: *Adaptive Behavior* 32.3, pp. 265–281.

Şahin, E. (2005). "Swarm robotics: from sources of inspiration to domains of application". In: *Swarm Robotics: SAB 2004 International Workshop*. Ed. by E. Şahin and W. M. Spears. Vol. 3342. Lecture Notes in Computer Science. Springer, pp. 10–20.

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

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

Salman, R., Alzaatreh, A., and Sulieman, H. (2022). "The stability of different aggregation techniques in ensemble feature selection". In: *Journal of Big Data* 9, p. 51.

Salman, M., Garzón Ramos, D., and Birattari, M. (2024). "Automatic design of stigmergy-based behaviours for robot swarms". In: *Communications Engineering* 3, p. 30.

Salman, M. (2024). "Automatic design of pheromone-based stigmergy in robot swarms". PhD thesis. Université Libre de Bruxelles.

Schmickl, T. and Hamann, H. (2011). "BEECLUST: a swarm algorithm derived from honeybees". In: *Bio-inspired Computing and Networking*. Ed. by Y. Xiao. CRC Press, pp. 95–137.

Schmickl, T., Thenius, R., Moslinger, C., Timmis, J., Tyrrell, A., Read, M., Hilder, J., Halloy, J., Campo, A., Stefanini, C., Manfredi, L., Orofino, S., Kernbach, S.,

Dipper, T., and Sutantyo, D. (2011). "CoCoRo – the self-aware underwater swarm". In: *2011 Fifth IEEE Conference on Self-Adaptive and Self-Organizing Systems Workshops (SASOW)*. Ed. by A. Lomuscio, P. Scerri, A. Bazzan, and M. Huhns. IEEE, pp. 120–126.

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.

Sebastián, E., Montijano, E., and Sagüés, C. (2022). "Adaptive multirobot implicit control of heterogeneous herds". In: *IEEE Transactions on Robotics* 38.6, pp. 3622–3635.

Shucker, B. and Bennett, J. K. (2007). "Scalable control of distributed robotic macrosensors". In: *Distributed Autonomous Robotic Systems 6*. Ed. by R. Alami, R. Chatila, and H. Asama. Springer, pp. 379–388.

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.

Soleymani, T., Trianni, V., Bonani, M., Mondada, F., and Dorigo, M. (2015). "Bio-inspired construction with mobile robots and compliant pockets". In: *Robotics and Autonomous Systems* 74, pp. 340–350.

Šošić, A., Khuda Bukhsh, W. R., Zoubir, A. M., and Koeppl, H. (2017). "Inverse reinforcement learning in swarm systems". In: *AAMAS '17: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), pp. 1413–1421.

Soysal, O. and Şahin, E. (2005). "Probabilistic aggregation strategies in swarm robotic systems". In: *2005 IEEE Swarm Intelligence Symposium, SIS 2005*. IEEE, pp. 325–332.

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

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*. Ed. by B. Bogaerts, G. Bontempi, P. Geurts, N. Harley, B.

Lebichot, T. Lenaerts, and G. Louppe. 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.

Sperati, V., Trianni, V., and Nolfi, S. (2008). "Evolving coordinated group behaviours through maximisation of mean mutual information". In: *Swarm Intelligence* 2.2–4, pp. 73–95.

Sperati, V., Trianni, V., and Nolfi, S. (2011). "Self-organised path formation in a swarm of robots". In: *Swarm Intelligence* 5.2, pp. 97–119.

Spears, W. M. and Gordon, D. F. (1999). "Using artificial physics to control agents". In: *1999 International Conference on Information Intelligence and Systems*. IEEE Computer Society, pp. 281–288.

St-Onge, D., Varadharajan, V. S., Ivan, Š., and Beltrame, G. (2020). "From design to deployment: decentralized coordination of heterogeneous robotic teams". In: *Frontiers in Robotics and AI* 7, p. 51.

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.

Strobel, V., Castelló Ferrer, E., and Dorigo, M. (2018). "Managing byzantine robots via blockchain technology in a swarm robotics collective decision making scenario". In: *AAMAS '18: Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), pp. 541–549.

Strobel, V., Castelló Ferrer, E., and Dorigo, M. (2020). "Blockchain technology secures robot swarms: a comparison of consensus protocols and their resilience to byzantine robots". In: *Frontiers in Robotics and AI* 7, p. 54.

Strobel, V., Pachecho, A., and Dorigo, M. (2023). "Robot swarms neutralize harmful Byzantine robots using a blockchain-based token economy". In: *Science Robotics* 8.79, eabm4636.

Strobel, V., Dorigo, M., and Fritz, M. (2024). *LLM2Swarm: robot swarms that responsively reason, plan, and collaborate through LLMs*. `https://arxiv.org/abs/2410.11387`.

Szpirer, J., Garzón Ramos, D., and Birattari, M. (2024a). "Automatic design of robot swarms that perform composite missions: an approach based on in-

verse reinforcement learning". In: *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 5791–5798.

Szpirer, J., Garzón Ramos, D., and Birattari, M. (2024b). *Automatic design of robot swarms that perform composite missions: an approach based on inverse reinforcement learning*. https://iridia.ulb.ac.be/supp/IridiaSupp2023-003.

Talamali, M. S., Bose, T., Haire, M., Xu, X., Marshall, J. A. R., and Reina, A. (2020). "Sophisticated collective foraging with minimalist agents: a swarm robotics test". In: *Swarm Intelligence* 14.1, pp. 25–56.

Talamali, M. S., Saha, A., Marshall, J. A. R., and Reina, A. (2021). "When less is more: robot swarms adapt better to changes with constrained communication". In: *Science Robotics* 6.56, eabf1416.

Tan, M. (1993). "Multi-agent reinforcement learning: independent vs. cooperative agents". In: *ICML'93: Proceedings of the 10th Annual International Conference on Machine Learning*. ACM, pp. 330–337.

Tomczak, J. M., Wundefinedglarz-Tomczak, E., and Eiben, A. (2020). "Differential evolution with reversible linear transformations". In: *GECCO'20: Proceedings of the Genetic and Evolutionary Computation Conference*. Ed. by J. A. Lozano. ACM, pp. 205–206.

Trianni, V., Labella Thomas, H., and Dorigo, M. (2004). "Evolution of direct communication for a swarm-bot performing hole avoidance". In: *Ant Colony Optimization and Swarm Intelligence: 4th International Workshop, ANTS 2004*. Ed. by M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, and T. Stützle. Vol. 3172. Lecture Notes in Computer Science. Springer, pp. 130–141.

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.

Trianni, V. and López-Ibáñez, M. (2015). "Advantages of task-specific multi-objective optimisation in evolutionary robotics". In: *PLOS ONE* 10.8, e0136406.

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

Turgut, A. E., Çelikkanat, H., Gökçe, F., and Şahin, E. (2008). "Self-organized flocking in mobile robot swarms". In: *Swarm Intelligence* 2.2, pp. 97–120.

Valentini, G., Hamann, H., and Dorigo, M. (2015). "Efficient decision-making in a self-organizing robot swarm: on the speed versus accuracy trade-off". In: *AAMAS '15: Proceedings of the 2015 International Conference on Autonomous*

Agents and Multiagent Systems. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), pp. 1305–1314.

Valentini, G., Ferrante, E., and Dorigo, M. (2017). "The best-of-n problem in robot swarms: formalization, state of the art, and novel perspectives". In: *Frontiers in Robotics and AI* 4, p. 9.

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.

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.

Winfield, A., Harper, C. J., and Nembrini, J. (2005). "Towards dependable swarms and a new discipline of swarm engineering". In: *Swarm Robotics: SAB 2004 International Workshop*. Ed. by E. Şahin and W. M. Spears. Vol. 3342. Lecture Notes in Computer Science. Springer, pp. 126–142.

Winfield, A., Liu, W., Nembrini, J., and Martinoli, A. (2008). "Modelling a wireless connected swarm of mobile robots". In: *Swarm Intelligence* 2.2, pp. 241–266.

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.

Xie, T., Jiang, N., Wang, H., Xiong, C., and Bai, Y. (2021). "Policy finetuning: bridging sample-efficient offline and online reinforcement learning". In: *NeurIPS'21: Proceedings of the 34th International Conference on Neural Information Processing Systems*. Ed. by M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan. Vol. 34. Curran Associates, Inc., pp. 27395–27407.

Yang, G.-Z., Bellingham, J., Dupont, P. E., Fischer, P., Floridi, L., Full, R., Jacobstein, N., Kumar, V., McNutt, M., Merrifield, R., Nelson, B. J., Scassellati, B., Taddeo, M., Taylor, R., Veloso, M., Wang, Z. L., and Wood, R. J. (2018). "The grand challenges of Science Robotics". In: *Science Robotics* 3.14, eaar7650.

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.

Zhu, W., Oğuz, S., Heinrich, M. K., Allwright, M., Wahby, M., Christensen, A. L., Garone, E., and Dorigo, M. (2024). "Self-organizing nervous systems for robot swarms". In: *Science Robotics* 9.96, eadl5161.

Zitzler, E. and Thiele, L. (1998). "Multiobjective optimization using evolutionary algorithms – a comparative case study". In: *Parallel Problem Solving from Nature – PPSN V: 5th International Conference*. Ed. by A. Eiben, T. Bäck, M.

Schoenauer, and H.-P. Schwefel. Vol. 1498. Lecture Notes in Computer Science. Springer, pp. 292–301.

Zopounidis, C. and Doumpos, M., eds. (2017). *Multiple Criteria Decision Making: Applications in Management and Engineering*. Multiple Criteria Decision Making. Springer.