ECOLE
POLYTECHNIQUE
DE BRUXELLES

UNIVERSITÉ LIBRE DE BRUXELLES
Ecole Polytechnique de Bruxelles
IRIDIA - Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

A modular approach to the automatic design of control software for robot swarms

From a novel perspective on the reality gap to AutoMoDe

Gianpiero FRANCESCA

Promoteur de Thèse:
Prof. **Mauro BIRATTARI**

Thèse présentée en vue de l'obtention du titre de
Docteur en Sciences de l'Ingénieur

Année académique 2016-2017

to Maria

The thesis

The reality gap problem bears a strong resemblance with the generalization problem experienced in supervised learning.

This resemblance can be exploited to conceive a novel modular approach to the automatic design of control software for robot swarms.

This novel approach appears to be more robust to the reality gap than the classical evolutionary robotics approach.

Summary

The main issue in swarm robotics is to design the behavior of the individual robots so that a desired collective behavior is achieved. A promising alternative to the classical trial-and-error design approach is to rely on automatic design methods. In an automatic design method, the problem of designing the control software for a robot swarm is cast into an optimization problem: the different design choices define a search space that is explored using an optimization algorithm. Most of the automatic design methods proposed so far belong to the framework of evolutionary robotics. Traditionally, in evolutionary robotics the control software is based on artificial neural networks and is optimized automatically via an evolutionary algorithm, following a process inspired by natural evolution. Evolutionary robotics has been successfully adopted to design robot swarms that perform various tasks. The results achieved show that automatic design is a viable and promising approach to designing the control software of robot swarms. Despite these successes, a widely recognized problem of evolutionary robotics is the difficulty to overcome the reality gap, that is, having a seamless transition from simulation to the real world.

In this thesis, we aim at conceiving an effective automatic design approach that is able to deliver robot swarms that have high performance once deployed in the real world. To this, we consider the major problem in the automatic design of robot swarms: the reality gap problem. We analyze the reality gap problem from a machine learning perspective. We show that the reality gap problem bears a strong resemblance to the generalization problem encountered in supervised learning. By casting the reality gap problem into the bias-variance tradeoff, we show that the inability to overcome the reality gap experienced in evolutionary robotics could be explained by the excessive representational power of the control architecture adopted. Consequently, we propose AutoMoDe, a novel automatic design approach that adopts a control architecture with low representational power. AutoMoDe designs software in the form of a probabilistic finite state machine that is composed automatically starting from a number of pre-existing parametric modules.

In the experimental analysis presented in this thesis, we show that adopting a

control architecture that features a low representational power is beneficial: AutoMoDe performs better than an evolutionary approach. Moreover, AutoMoDe is able to design robot swarms that perform better that the ones designed by human designers. AutoMoDe is the first automatic design approach that it is shown to outperform human designers in a controlled experiment.

Original contributions

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

The theoretical framework: We define the theoretical framework in which the reality gap is described and analyzed using a machine learning perspective. In this perspective, the reality gap bears a strong resemblance with the generalization problem experienced in supervised learning and it can be cast in terms of the bias-variance tradeoff.

Critical review of the current practice in the design of robot swarms: We provide an overview of the state of the art in the design of robot swarms that highlights the lack of a common empirical practice. We discuss the challenges to be overcome to establish a proper empirical practice for the domain of the design of robot swarms.

Formal definition of a reference model: We provide the first formal definition of a reference model in the domain of swarm robotics. The reference model formally defines the capabilities, in terms of sensors and actuators, that the robotic platform provides to the control software and it allows for a fair comparison of different design methods or different control software implementations. The concept of reference model is of general relevance for swarm robotics.

Introduction of AutoMoDe: We introduce AutoMoDe, a novel automatic design approach that, following the precepts of the bias-variance tradeoff, features a control architecture with low representational power.

Definition of design methods: We define five design methods: two specializations of AutoMoDe called `Vanilla` and `Chocolate`, an evolutionary design method called `EvoStick`, and two manual methods called `U-Human` and `C-Human`.

Definition of an empirical practice: We define an empirical practice that allows for a fair comparison of a number of design methods on different swarm robotics tasks.

Robot experiments: We present empirical analyses in which robot experiments play a prominent role. The results presented in this thesis have been built upon over

six hundred runs with robot swarms.

Statement

This thesis presents an original work that has never been submitted to Université Libre de Bruxelles or any other institution for the award of a Doctoral degree. Some parts of this thesis are based on a number of peer-reviewed articles that the author, together with other co-workers, has published in the scientific literature.

The review and the critique of the literature presented in Chapter 2 is based on:

> **Francesca, G.** and Birattari, M. (2016). Automatic design of robot swarms: achievements and challenges. *Frontiers in Robotics and AI*, 3(29):1–9.

Chapter 3 (the theoretical framework and the definition of AutoMoDe) and Chapter 4 (the description of `Vanilla` and the experimental results) are based on:

> **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. *Swarm Intelligence*, 8(2):89–112.
>
> 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 Dorigo, M., Birattari, M., Li, X., López-Ibáñez, M., Ohkura, K., Pinciroli, C., and Stützle, T., editors, *Swarm Intelligence, ANTS 2016*, volume 9882 of *LNCS*, pages 149–160. Springer, Berlin, Germany.

The definition of `Chocolate` and the experimental results presented in Chapter 5 are based on:

> **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. (2014b). An experiment in automatic design of robot swarms. In Dorigo, M., Birattari, M., Garnier, S., Hamann, H., Montes de Oca, M., Solnon, C., and Stützle, T., editors, *Swarm Intelligence, ANTS 2014*, volume 8667 of *LNCS*, pages 25–37. Springer, Berlin, Germany.

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. *Swarm Intelligence*, 9(2–3):125–152.

Moreover, the research work presented in this thesis resulted in a number of publications on tools that have been developed, together with co-workers, by the author.

The software infrastructure to use the e-puck robots is described in:

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. *Swarm Intelligence*, 9(1):1–22.

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

The tracking system to calculate the positions of e-puck robots in the experimental arena is described in:

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

Reina, A., Salvaro, M., **Francesca, G.**, Garattoni, L., Pinciroli, C., Dorigo, M., and Birattari, M. (2015b). Augmented reality for robots: Virtual sensing technology applied to a swarm of e-pucks. In *NASA/ESA Conference on Adaptive Hardware and Systems*, pages 1–6.

Finally, implementations of AutoMoDe (`Vanilla` and `Chocolate`) and `EvoStick` have been released as opensource projects. These implementations are improved versions of the ones used for the experiments presented in this thesis. These implementations are described in:

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

Acknowledgments

I acknowledge support from the following projects: Meta-X, founded by the Scientific Research Directorate of the French Community of Belgium; COMEX, (P7/36) funded by the Interuniversity Attraction Poles Programme of the Belgian Science Policy Office; E-SWARM, funded by the European Research Council (ERC) under the European Union's Advanced Grants program (grant agreement No 246939); DEMIURGE, funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 681872).

I wish to thank my supervisor and mentor Dr. Mauro Birattari. It has been an honor to be his first PhD student. Mauro has taught me the love for research: his passion and entusiasm were contagious and they were a source of motivation in the difficult times. I deeply appreciate the time and effort Mauro invested in me. We spent countless hours in discussions and brainstorming. That is what supervisors do. Mauro never stopped there and, along with purely intellectual activities, Mauro supported me in many manual activities such as lasercutting, robot placing, robot repairing, arena building, etc. Mauro has taught me how important is to deliver a clear and coherent message that is tailored for the particular audience to be targeted. Mauro has taught me that a good researcher is also a kind and humble person. For these lessons and many other life lessons that I do not specify here, thank you Mauro!

The research presented in this thesis benefited from the interaction, the contribution, and the help of many researchers. I wish to thank Dr. Vito Trianni for sharing with me his immense knowledge on evolutionary swarm robotics. I wish to thank Prof. Marco Dorigo for his extremely valuable feedback on my research. Marco's comments have been always fruitful: for instance, the idea of comparing automatic and manual design methods originated from a discussion we had in January 2014. I wish to thank Dr. Manuele Brambilla for his support, especially during the initial phases of my PhD studies, and for suggesting the name AutoMoDe. As my research intersected with optimization, I wish to thank Prof. Thomas Stützle for clearing my doubts on many theoretical and technical details about the *irace* package. In this regards, I acknowledge the help (and the patience) of the *irace* team: Dr. Manuel

López-Ibáñez, Dr. Jérémie Dubois-Lacoste and Leslie Perez. Prof. Carlo Pinciroli deserves a special mention for his support on the ARGoS simulator. I wish to thank all my (previously unmentioned) co-authors: Dr. Arne Brutschy, Lorenzo Garattoni, Roman Miletitch, Dr. Gaëtan Podevijn, Dr. Andreagiovanni Reina, Touraj Soleymani, Mattia Salvaro, Dr. Franco Mascia, Prof. Elio Tuci and Dr. Giovanni Pini. I wish to thank Dr. Gabriele Valentini for his friendly advices and for the many philosophical discussions on research and life. In one shot I wish to say a huge "thank you!" to all the professors, researchers, students and friends that I had the honor to meet at IRIDIA. I will not make a list because I am sure that I would forget someone in the rush to submit this thesis. I just say that everyone of you guys enriched my life.

I wish to thank Prof. Antonella Santone who was crazy enough to bet on me, a noisy bachelor student with a lot of enthusiasm and few skills. She supervised me during the research for my first paper. I wish to thank Dr. Paola Pellegrini for her supervision during the research for my second paper. Even though Prof. Santone and Dr. Pellegrini supervised me when I was an undergraduate student, they had a profound impact on my decision to pursue a PhD.

On a more personal basis, a special thanks goes to the long-time friend Federico and to my favourite opponent in soccer games, Lorenzo. In addition, I wish to thank Wakao-san, Isotta, Anja, Mark, Sven and Kobori-san, my colleagues at Toyota Motor Europe, who have been really supportive... and extremely patient with me.

Ringrazio infinitamente la nostra famiglia, tutta. Siete sempre un porto sicuro per noi e rappresentate l'esempio da seguire.

* * *

This thesis is dedicated to my beloved wife Maria that has walked with me in this, and many other adventures. Her contribution to this thesis goes beyond the usual loving support that a wife gives to a husband. Yes, Maria has done all that, but she did not stop there. Because Maria takes actions for her loved ones! Just to make an example above all, I still remember with affection the many weekends and evenings Maria spent with me in the experimental arena, placing 20 e-puck robots in less than 45 seconds! This shows the splendid and lovable person Maria is. For many reasons, that I hope I tell you enough, I love you very much.

Gianpiero

Contents

Chapter 1

Introduction

Robots are an integral part of the modern human society. Robots are everywhere. For example, they raise productivity in factories, they build complex structures under the see, they explore environments out of reach for humans, they assist at the operating table, and they perform tasks that humans cannot perform. The adoption of robots has been pushed forward by important advancements in core technologies needed in robotic systems such as sensors and actuators, manipulators, control systems, materials, batteries, and ultimately artificial intelligence (Yang and McNutt, 2016; Chouard and Venema, 2015; Ford, 2015).

The adoption of robots is radically changing the way tasks are performed. Historically, industrial automation has been the primary domain of application for robotics. Nowadays robots are part of "smart factories" where robots, other machineries, and humans work together and exchange information in order to speed up the production process, reduce costs, and increase quality (Brettel et al., 2014; Radziwon et al., 2014; Lee et al., 2014). Similarly, robots are an integral part of warehouses where they organize, sort, stock, and retrieve a wide variety of items (Letzing, 2012; Basile et al., 2015; Son et al., 2016). Self driving vehicles are already present in factories, mines and even on the same roads used by humans (Sun et al., 2006; Gusikhin et al., 2008; Bengler et al., 2014).

It is to be expected that the presence of robots in our societies will keep increasing and more and more robots will be working together. In this context, robots will be organized in teams or even large groups. One way to coordinating large groups of robots is swarm robotics. Swarm robotics (Şahin, 2005; Dorigo et al., 2014) is an approach to coordinating groups of robots that takes inspiration from the self-organized behaviors of social animals such as ants, bees, etc. These biological systems, despite being composed of individuals with limited capabilities, show extremely complex behaviors. For instance, ants are capable of building nests

featuring tunnels, rooms, and chambers that can reach the complexity of human cities (MacKay, 1981).

In swarm robotics, a large number of robots cooperate and accomplish tasks that single individuals would be unable to accomplish. A robot swarm is a highly redundant system that acts in a self-organized way without the need of any form of centralized coordination. The collective behavior of the swarm is the result of the local interactions that each robot has with its neighboring peers and with the environment.

The self-organized and distributed nature of robot swarms makes them challenging to design. The requirements are typically expressed at the swarm level by specifying the task that the swarm, as a whole, has to perform. However, the swarm is a collective entity and, as such, it is an immaterial concept. In particular, the swarm itself cannot be *programmed*, only the individual robots can. The designer's task is therefore *indirect*: they have to design the individual-level behaviors of the robots that, through a complex set of robot-robot and robot-environment interactions, result in the desired collective behavior of the swarm. At the moment, there is no general approach to the design of robot swarms, even though some preliminary proposals have been made (Hamann and Wörn, 2008; Kazadi et al., 2007; Berman et al., 2011; Brambilla et al., 2012). Currently, most robot swarms are designed by hand using a trial-and-error process: an individual-level behavior is iteratively improved and tested until the desired collective behavior is obtained. This approach is closer to craftsmanship than to engineering: the quality of the result strongly depends on the experience and intuition of the designer. Moreover, this trial-and-error process is time consuming, costly, and lacks repeatability and consistency.

An alternative way to develop robot swarms is to rely on automatic design. In automatic design, the design problem is cast into an optimization problem and then tackled using optimization algorithms. To date, the main automatic design approach that has been adopted in swarm robotics is *evolutionary robotics* (Nolfi and Floreano, 2000). In this approach, an evolutionary algorithm is used to obtain the parameters of a neural network that maps the sensor readings of the individual robot into values fed to its actuators. A large literature shows that evolutionary robotics is able to produce robot swarms that can perform a number of tasks (Bongard, 2013; Doncieux and Mouret, 2014; Trianni, 2014; Silva et al., 2015a).

Nonetheless, evolutionary robotics presents some known limitations. For instance, the current practice of evolutionary robotics lacks of an engineering methodology (Trianni and Nolfi, 2011). The process to obtain a solution involves multiple iterations where the designer tweaks the setup of the automatic method until the

desired solution is generated by the automatic method. This process appears similar to the process adopted by a trial-and-error manual approach: the only difference is that, while in the manual approach the designer acts on the control software directly, in the case of the automatic approach the designer acts on the automatic method that generates the control software. Another limitation of evolutionary robotics is that, being based on neural networks, the instances of control software obtained are *black boxes* that can hardly be analyzed, verified and maintained (Matarić and Cliff, 1996). Most importantly, in the context of swarm robotics, the evolutionary approach has not demonstrated the capability of scaling in complexity and providing solutions for realistic applications (Trianni, 2014). Among the causes, we reckon the difficulty in overcoming the reality gap, that is, having a seamless transition from simulation—the main tool for evolutionary design—to the real world.

In this thesis, we conjecture that the observed limitations of evolutionary robotics result from an uncontrolled *representational power* of the control architecture that is typically adopted in evolutionary robotics. Indeed, one of the tenets of evolutionary robotics is to minimize the assumptions and the bias injected by the designer (Harvey et al., 1997; Nolfi and Floreano, 2000; Bongard, 2013). The idea is to rely on an evolutionary process to fine-tune the dynamics of the interaction between the robot and the environment. To this aim, a common assumption in the literature is the need of a control architecture that features a high representational power—for example, a neural network.

Unfortunately, a high representational power may be counter-productive in the peculiar working conditions faced in swarm robotics, which are highly dynamic and uncertain due to the numerous robot-robot interactions. We claim that such working conditions offer limited regularities to be discovered and exploited by the evolutionary process. As a consequence, it is likely that evolution will produce control software that exploits "idiosyncratic features" (Floreano and Keller, 2010) of the simulation, that is, the differences between simulation and reality, which unavoidably occur. This control software will generalize poorly and will be unable to overcome the reality gap. Because the *a priori* identification of the differences between simulation and reality is in general difficult, an automatic design approach must be as robust as possible to their presence.

In this thesis, we consider the reality gap with the aim of conceiving an automatic design method that is able to produce robot swarms that perform effectively in the real word. The intuition of this thesis is that the reality gap problem faced by evolutionary robotics—or any other automatic design method— bears a strong resemblance to the generalization problem faced in supervised learning. Following

this intuition, we cast the reality gap problem in terms of the bias-variance tradeoff formalized in the supervised learning literature (Geman et al., 1992).

The bias-variance tradeoff correlates the generalization capabilities of an approximator with its complexity (or with the amount of computational effort used). Past an optimal level, increasing the complexity of an approximator (or the training effort) is counterproductive because it hinders the generalization abilities of the approximator itself. On the basis of the bias-variance tradeoff, we define two hypotheses in the context of the automatic design of robot swarms. Hypothesis hp1: past an optimal level, increasing the complexity of the control architecture (i.e., its representational power) is counterproductive. Hypothesis hp2: past an optimal level, increasing the design effort is counterproductive. In both cases, past an optimal level, which is unknown *a priori*, the performance in simulation increases while the performance in reality decreases.

In context of this thesis, the two hypotheses represent a tool that we use to understand the implications of the reality gap, with the ultimate goal of conceiving an automatic design method that is able to to produce robot swarms that perform effectively in the real word. The two hypotheses will be corroborated in this thesis. Hypothesis hp1 has a prominent role while hp2 will be corroborated in a proof-of-concept experiment.

Following Hypothesis hp1, we conjecture that the reality gap problem can be tackled through a suitable injection of bias in the control architecture adopted by the automatic design process. We define AutoMoDe, a novel automatic design approach that, while having a control architecture with a low representational power, is able to design robot swarms to tackle many tasks of interest. AutoMoDe generates an individual-level behavior in the form of a probabilistic finite state machine composed of preexisting parametric *modules*. AutoMoDe develops control software by selecting, via an optimization algorithm, the topology of the probabilistic finite state machine, the modules to be included, and the value of their parameters. The set of modules and the rules to compose a probabilistic finite state machine represent the bias injected in the automatic design process. As a result of the injection of bias, we expect that the variance of the behaviors designed by AutoMoDe will be consequently reduced and ultimately the control software will overcome the reality gap. We test AutoMoDe via experiments with a swarm of e-puck robots. The results show that having a low representational power is beneficial: AutoMoDe nicely overcomes the reality gap.

We corroborate Hypothesis hp2 via an experiment involving a swarm of 20 e-puck robots. In the experiment, we compare the performance of robot swarms ob-

tained using increasing design budgets (i.e., increasing computational effort). The results corroborate hp2: past an optimal level of design budget, the performance in simulation increases while the performance in reality decreases.

The rest of the thesis is organized as follows. In Chapter 2, we give an overview of the domain of swarm robotics with a particular focus on the design of control software for robot swarms. We highlight the characteristics of swarm robotics along with possible applications. We review the notable achievements in both manual and automatic design. We analyze the current literature on the automatic design of robot swarms and we highlight that a well-established empirical practice is still missing. Finally, we discuss the challenges to be overcome to establish a proper empirical practice in the domain of the automatic design of robot swarms.

In Chapter 3, we describe the framework that constitutes the theoretical base of this thesis. We introduce the reality gap and, via the bias-variance tradeoff, we describe the intuition that the reality gap bears a strong resemblance with the generalization problem faced in supervised learning. Following this intuition, we describe two hypotheses. Hypothesis hp1, which relates the reality gap to the representational power of the control architecture, is the starting point for the definition of AutoMoDe, an automatic design approach that features a control architecture with low representational power. Hypothesis hp2, which relates the reality gap to the design effort, is corroborated via an experiment with a swarm of e-puck robots.

In Chapter 4, we introduce the first experiment to validate the core ideas of AutoMoDe. As AutoMoDe is an approach that needs to be specialized for a particular robotic platform, we introduce `Vanilla`, a specialization of AutoMoDe for the e-puck robot. We compare `Vanilla` with `EvoStick`, an implementation of evolutionary robotics. We perform an experiment on two tasks.

In Chapter 5, we describe two empirical studies: Study A and Study B. In Study A, we perform an experiment in which `Vanilla` and `EvoStick` are compared with human designers on five swarm robotics tasks. In Study B, we introduce another specialization of AutoMoDe, `Chocolate`. `Chocolate` differs from `Vanilla` only in the optimization algorithm. We compare `Vanilla`, `EvoStick` and `Chocolate` with human designers on five swarm robotics tasks.

In Chapter 6, we conclude the thesis. We summarize the contributions and we highlight their relevance. Finally, we suggest and discuss some future research directions.

Chapter 2

State of the art

This chapter contains a bird-eye view of the domain of swarm robotics with particular focus on the design of control software of robot swarms.

The design of the control software for robot swarms is the main issue in swarm robotics: defining what the individuals should do in order to achieve a desired collective goal is challenging and a general approach to tackle this issue is still missing. Automatic design is a promising approach to the design of control software for robot swarms. In an automatic design method, the design problem is cast into an optimization problem and is addressed using an optimization algorithm.

Although automatic design has been successfully applied to the design of robot swarms, an apparent issue that emerges from the literature is that a solid, well-established, and consistently applied empirical practice is still missing. For example, studies that propose new methods and ideas do not typically provide any comparison with existing ones.

We maintain that the lack of a proper empirical practice hinders the progress of the research on the automatic design of control software for robot swarms. As a consequence, the review of the literature that we propose has two main goals: First, we highlight the notable achievements in the (automatic) design of control software for robot swarms with a particular focus on the empirical approach adopted to assess the performance of the new proposed design methods. Second, we discuss the challenges to be overcome to establish a proper empirical practice for the domain.

This chapter is organized as follows: In Section 2.1, we introduce the domain of swarm robotics. In Section 2.2, we describe the design problem faced in swarm robotics. Section 2.3 and Section 2.4 are devoted to the description of manual and automatic design, respectively. In Section 2.4 we highlight studies that present notable achievements in the domain of the automatic design of control software for robot swarms with a particular attention to the empirical approach adopted to assess new

proposals. In Section 2.5, we discuss some challenges that the research community should overcome to transform the current research on the automatic design of control software for robot swarms into a mature science. In Section 2.6, we conclude with a summary of the analysis of the state of the art performed in this chapter.

2.1 Swarm robotics

In swarm robotics (Şahin, 2005; Dorigo et al., 2014), a large number of robots cooperate and accomplish tasks that a single individual would be unable to accomplish. A robot swarm is a highly redundant system that acts in a self-organized way without the need of any form of centralized coordination. The collective behavior of the swarm is the result of the local interactions that each robot has with its neighboring peers and with the environment. Swarm robotics was born as application of the concepts of swarm intelligence (Beni, 2005) to robotics. Swarm intelligence studies the behavior of systems, either natural or artificial, composed of many agents that interact locally.

Over the years, we can find many reviews of the literature in swarm robotics. Şahin (2005) is the first to survey the literature and to provide the first definition of swarm robotics. For Şahin (2005) swarm robotics is defined as: " the study of how a large number of relatively simple physically embodied agents can be designed such that a desired collective behavior emerges from the local interactions among agents and between the agents and the environment." Two years later, Bayindir and Şahin (2007) organized the literature using five taxonomies: modeling, design, communication, analytical studies and problem. Gazi and Fidan (2007) reviewed the literature focusing on the techniques to model and design swarm robotics systems. This review follows a control-theory perspective. Brambilla et al. (2013) surveyed the literature from an engineering perspective. They classified the contributions into design methods, modeling methods and collective behaviors. Garattoni and Birattari (2016) reviewed the literature following the same classification proposed by Brambilla et al. (2013). This review surveys the most notable swarm robotics systems and the possible future applications of swarm robotics.

This section gives a brief introduction to swarm robotics following the footsteps of Garattoni and Birattari (2016). This section is not to be considered a comprehensive review of the literature in swarm robotics. For comprehensive reviews of the literature in swarm robotics we refer the reader to Brambilla et al. (2013) and Garattoni and Birattari (2016).

2.1.1 Characteristics of a robot swarm

A robot swarm has some peculiar characteristics:

Self-organization. A robot swarm is an autonomous system that behaves in a self-organized manner. In a robot swarm there is no leader that coordinates the individual robots and the collective behavior showed by the swarm is the result of the interactions that each individual robot has with its peers and with the environment.

Redundancy. In a robot swarm, no single robot is indispensable. A robot swarm is composed of a large number of robots and there are multiple robots that can tackle each of the tasks required to accomplish a given mission. When the robot swarm is composed of robots that have identical hardware and control software, the swarm is defined *homogeneous*. On the contrary, when the robot swarm is composed of different classes of robots each one having different hardware and control software, the swarm is defined *heterogeneous*. In both cases, a robot swarm is highly redundant.

Locality. In a robot swarm, the individual robots have local perception and communication abilities. For this reason, the information that the individual robots use to determine their behavior are local: the individual robots interact with their peers in the neighborhood and they are unaware of the total number of robots that globally are taking part in the mission.

Parallelism. A robot swarm tackles multiple tasks at the same time since at any given time, different robots are engaged in different tasks. The individual robots may switch task according to the circumstances. This task allocation follows the same principles of self-organization and locality.

2.1.2 Properties of a robot swarm

The characteristics of self-organization, redundancy, locality and parallelism showed by a robot swarm are appreciated since they are commonly reckoned to promote properties that are desirable in a robotic system such as fault tolerance, scalability and flexibility:

Fault tolerance. A robot swarm is self-organized and it is not managed or coordinated by a single, central entity. Moreover, a robot swarm is composed of a large number of robots that are interchangeable one another. For these two reasons, a robot swarm can cope with the failure of some of its individual robots.

Scalability. Since a robot swarm uses only local communication and perception, its behavior does not rely on a particular number of robots. For this reason, in principle, the behavior of the robot swarm is not qualitatively altered if robots are added or removed.

Flexibility. A robot swarm is parallel and it is composed of robots that can switch tasks according to contingencies. For this reason, a robot swarm can cope with modifications of the environment and changes of the working conditions.

2.1.3 Differences between swarm robotics and traditional approaches

Swarm robotics can be seen as opposed to traditional approaches such as the single-robot approach (Nilsson, 1984) and classical multi-robot approach (Dudek et al., 1996; Parker, 2000; Iocchi et al., 2001).

In a single robot approach, a single robot is in charge of the given mission. The main issue of having a single robot is that the system presents a single point of failure: the damage or the failure of the single robot would jeopardize the entire mission. For this reason, swarm robotics appears more promising for missions in which the probability of damaging the robot is particularly high. In addition, if we consider a robot swarm and a single monolithic robot that should have the same capabilities, the individual robots composing the swarm have, typically, simpler mechanics and feature simpler sensors than the monolithic robot. This should lead to a reduced cost in terms of hardware design in the case of swarm robotics and to a reduced failure probability. On the contrary, the effort to design the control software for the robot swarm is higher that the effort to program a monolithic robot.

In a classical multi-robot approach, a small group of robots is in charge of the given mission (Dudek et al., 1996; Gerkey and Matarić, 2004). The size of the group is relatively small compared to a typical robot swarm. Moreover, the behavior and the communication patterns are tailored at design time for the specific size of the group and each robot has a predefined role in the context of the mission. For this reason, the group of robots of a classical multi-robot approach is in general less scalable than a robot swarm: the addition or the removal of a robot requires a new design of the control software. Similarly, a classical multi-robot approach is less flexible and fault tolerant: changes due to contingencies or failure of some robots have to be considered and taken care of at design time. However, the design of a multi-robot system is easier compared to the design of a robot swarm.

2.1.4 Possible applications

Swarm robotics is a promising approach for those applications that require robotic systems to be fault tolerant, scalable and flexible. Among these applications we find: search and rescue of survivors in disaster areas, humanitarian demining, exploration of planets or of unknown environments, removal of materials like waste or pollutant, surveillance, nano-medicine and nano-surgery. In this section, we follow the same reasoning presented in Garattoni and Birattari (2016) and we cover applications for which some preliminary results have already been achieved:

Patrolling. Christensen et al. (2015) presented a swarm of aquatic-surface robots that aims at performing tasks such as patrolling, intruder detection, and monitoring. A first achievement is presented in Duarte et al. (2016), where a swarm of 10 robots performs monitoring in a waterbody next to the Tagus river in Lisbon, Portugal. Simulated experiments have shown that a swarm composed of 1000 aquatic robots are sufficient to cover 20 km of coast of the Lampedusa Island (Duarte et al., 2014b). More details on these experiments can be found in Section 2.4.1.

Warehouse automation. The automation of warehouses is a promising scenario for robot swarms. In particular, researchers are exploring the use of swarm robotics techniques to create a system that is able to optimize the flow of materials in warehouses while keeping the warehouse organized (Ackerman, 2012). For instance, algorithms inspired by ant's behaviors allow the automatic definition of the optimal paths for the individual robots in order to reduce interferences. Warehouse automation largely benefits from the characteristics of autonomy and decentralization: more robots can be added over time if required by contingencies and the failure of individual robots can be managed by the remaining robots of the swarm that will take the pending tasks. Moreover, the parallelism offered by a robot swarm allows for the reduction of the average order processing time.

Agriculture. Attempts of using robot swarms to automate agricultural processes have been already made both in the industry (Lubin, 2011; Harvest, 2008) and through academic projects (RHEA, 2010; University of Copenhagen, 2011). These projects deal with heterogeneous swarms composed by aerial and ground robots. The aerial robots observe the environment from above in order to provide information to guide the ground robots (e.g., the areas with higher concentration of harvest to crop or the position of the weeds to be removed). The ground robots combine this information with the one they collect locally in order to perform the required activities.

Medicine. Swarm of nano-robots that are meant to be injected in the human body

are currently under study. The main idea is to use a swarm of nano-robot to carry out diagnosis or targeted drug-delivery. In this context, the main challenge is the manufacturing of the robots. To tackle this challenge, two main approaches are present in the literature. The first approach is based on nano-robots that do not have any sensor nor actuator. The behavior of these nano-robots is defined at design time by tuning their physical/chemical properties such as, material, shape, charge and coating (Hauert and Bhatia, 2014; Hauert et al., 2013). The second approach is based on more complex nano-robot that are able to move autonomously, recognize a target and release drugs (Chobotix, 2008; Sarvašová et al., 2015). In both approaches, swarm behaviors are used to allow the nano-robots to perform tasks such as navigation in the blood stream, protection against macrophages, and identification of the targets.

2.2 The design problem in swarm robotics

The design of the behavior of the robot swarm is the core issue in swarm robotics. The self-organized and distributed nature of robot swarms makes them challenging to design. The requirements are typically expressed at the swarm level by specifying the task that the swarm, as a whole, has to perform. However, the swarm is a collective entity and, as such, it is an immaterial concept. In particular, the swarm itself cannot be *programmed*, only the individual robots can. The designer's task is therefore *indirect*: they have to design the individual-level behaviors of the robots that, through a complex set of robot-robot and robot-environment interactions, result in the desired collective behavior of the swarm.

Although the literature describes a number of robot swarms that have been developed and demonstrated, a reliable engineering approach to the design of control software for robot swarms is still at dawn (Brambilla et al., 2013). The most common approach to design robot swarms is the manual design: the designer manually develops the control software of the individual robots. Typically, designers proceed by trial-and-error, guided only by their intuition and experience. Some effort has been made recently to overcome this problem and a number of principled manual methods have been proposed (e.g., see Hamann and Wörn, 2008; Kazadi et al., 2009; Berman et al., 2011; Werfel et al., 2014; Brambilla et al., 2015; Valentini et al., 2016b; Reina et al., 2015c; Lopes et al., 2016). Although interesting and promising results have been obtained, we are far from a generally applicable solution.

Automatic methods are a promising alternative. In an automatic method, the problem of designing the control software for a robot swarm is cast into an opti-

mization problem: the different design choices define a search space that is explored using an optimization algorithm. Most of the automatic methods proposed so far belong to the framework of evolutionary robotics (Nolfi and Floreano, 2000). Traditionally in evolutionary robotics, the control software is based on artificial neural networks and is optimized automatically via an evolutionary algorithm, following a process inspired by natural evolution. As discussed in Section 2.4, evolutionary robotics has been successfully adopted to design robot swarms that perform various tasks. The results achieved show that automatic design is a viable and promising approach to designing the control software of robot swarms.

Unfortunately, the pioneering achievements registered so far are to be considered as somehow isolated contributions, rather than the incremental acquisitions of an established and mature science.[1] With few exceptions, studies that introduce new automatic design methods and ideas do not provide any comparison with previously introduced ones. Indeed, a solid, well-established, and consistently applied practice for the empirical assessment and comparison of automatic design methods is still missing.

2.3 Manual design

In manual design, the designer develops the control software of the individual robots by hand. Typically, the control architecture adopted is based on probabilistic finite state machines (Rabin, 1963). Even though the most commonly used manual approach is the trial-and-error approach, in the recent years principled manual design approaches have been proposed. In the next sections we describe the trial-and-error approach and some promising principled manual design approaches.

2.3.1 Trial-and-error design

Designing robot swarms using trial-and-error is closer to craftsmanship than to engineering: the designer proceeds with the implementation of the behavior in an unstructured way, with little scientific guidance. The designer tries to obtain the desired collective behavior by exploring the space of the individual behaviors of the robots. This exploration is driven only by the ingenuity and the expertise of the designer. The typical design process starts with the designer that develops a first implementation of the individual control software. This first implementation is then tested in order to evaluate the resulting collective behavior of the robot swarm.

[1] We use the notion of *mature science* as defined by Kuhn (1962).

The evaluation is usually performed using computer-based simulations. On the basis of the resulting collective behavior, the designer iteratively modifies and tests the implementation of the control software until the desired collective behavior is achieved.

In trial-and-error design, the implementation follows the principles of *behavior-based robotics* (Brooks, 1990, 1991; Parker, 1996a).

Behavior-based robotics follows the idea that robot do not need full and comprehensive representation of the world in order to act. In Brooks (1991), the author states that:

> Artificial intelligence research has foundered on the issue of representation. When intelligence is approached in an incremental manner, with strict reliance on interfacing to the real world through perception and action, reliance on representation disappears.

In behavior-based robotics, the robot behaves following its perception without trying to plan its actions based on a representation of the environment. This allows the development of robots that show complex behaviors while having a relatively simple control architecture. The control software is typically organized in a modular architecture that is largely inspired by Brooks' *subsumption architecture* (Brooks, 1986). Behavior-based robotics has been successfully applied to single-robot scenarios and then applied to swarm robotics. The application of the behavior-based approach to swarm robotics is straightforward, and has been the most common choice to date (Brambilla et al., 2013). The trial-and-error approach has been used to design robot swarms for various tasks including aggregation (Şahin, 2005), chain formation (Nouyan et al., 2009), and task allocation (Liu and Winfield, 2010).

The behavior-based approach does not address the core design problem that one faces in swarm robotics: it does not provide any guideline to define what the individual robot should do so that the given swarm-level specifications are met.

2.3.2 Principled manual design

Some works have proposed ideas to address the issue of deriving the individual behavior from the desired collective one. Unfortunately, most of them rely on strong assumptions and are not of general applicability as they have been conceived for specific tasks. The following is a brief overview of some of the most promising ideas. For a comprehensive review, we refer the reader to Brambilla et al. (2013).

Martinoli et al. (1999) used rate equations to model a collective clustering behavior and to guide the implementation of the control software of the individual

robots. The method was assessed both in simulation and with up to ten Khepera robots (Mondada et al., 1993). Lerman et al. (2001) and Martinoli et al. (2004) applied rate equations to a cooperative stick pulling task. The control software produced was tested with up to six Kheperas. Lerman and Galstyan (2002) used rate equations to model a foraging behavior under the effect of interference.

Kazadi et al. (2007) used a method based on artificial vector fields to develop a pattern formation behavior. The method is illustrated with simulations and appears to be limited to spatially organizing behaviors. Hsieh et al. (2007) proposed an approach based on artificial potentials to obtain control software for coordinated motion along predefined orbital trajectories. The authors provided convergence proofs and simulated experiments. Similarly, Sartoretti et al. (2014) proposed an approach based on stochastic differential equations driven by white Gaussian noise to tackle coordinated motion. In this case, the orbital trajectory is derived via collective consensus among the robots of the swarm. The approach has been validated with a swarm of eight e-puck robots.

Hamann and Wörn (2008) used Langevin equations to model the behavior of the individual robots, and analytically derived a Fokker-Planck equation that models the collective behavior of the swarm. Berman et al. (2011) adopted a similar approach based on a set of advection-diffusion-reaction partial differential equations to design control software for task allocation. Neither of the two approaches has been assessed in robot experiments yet.

Lopes et al. (2014, 2016) introduced an approach based on supervisory control theory. The approach has been demonstrated by designing a segregation behavior. The assessment has been performed with a swarm of 26 e-pucks and one of 42 kilobots (Rubenstein et al., 2014a). The main drawback of this approach is that it requires extensive domain knowledge.

Brambilla et al. (2012); Brambilla (2014); Brambilla et al. (2015) introduced an approach based on prescriptive modeling and model checking. The approach has been demonstrated by designing control software for two tasks: aggregation and foraging. The assessment has been performed with swarms of up to 20 e-pucks. Also in this case, the approach requires extensive domain knowledge.

Valentini (2016); Valentini et al. (2016a,b, 2014) proposed a modular and model-driven approach to design collective decision-making strategies for the best-of-n problem. They propose a generic control structure as well as a generic macroscopic model that can be instantiated by the designer to create a specific strategy and analyze its performance. The approach has been demonstrated by designing control software for two tasks: collective perception (Valentini et al., 2016a) and site selec-

tion (Valentini et al., 2016b). The assessment has been performed with a swarm of 20 e-pucks and one of 100 kilobots. A limit of the proposed approach is that is confined to the domain of collective decision making.

Another way to solve the collective-to-individual design issue is to define a catalogue of design patters (Babaoglu et al., 2006; Gardelli et al., 2007; Reina et al., 2014, 2015c; Reina, 2016). The definition of a design pattern in the context of swarm robotics is composed of: i) a model that describes the requirements of the desired collective behavior, ii) a description of the individual behavior that once installed on the robots of the swarm leads to the desired collective behavior, and iii) a mapping to obtain the parameters of the individual behavior starting from the parameters of the desired collective behavior. Reina et al. (2015a) demonstrated the use of design patterns on a decision-making task in a collective foraging scenario. The experiments were performed in simulation only.

Another principled manual design method that promotes the decomposition of the design problem has been proposed in order to obtain robot swarms that perform self-assembly (Rubenstein et al., 2014b) and construction (Werfel et al., 2014). This method is based on three steps described in the following considering the case of a construction task. First, the user describes the desired structure to be built in terms of physical properties such as shape and height. Second, the method translates the description of the structure into a series of actions that need to be performed in order to build the structure. Third, the method maps the actions into a set of rules that can be followed by the individual robots. In addition, this method allows for the validation of properties such as correctness and convergence of the construction process. This method can be applied only to a limited range of missions.

In the context of principled manual design, scripting languages that allow the designer to "program" the swarm as a whole entity have been proposed (Bachrach et al., 2010; Pinciroli et al., 2015). These scripting languages feature primitives that model behaviors that are useful in the context of swarm robotics such as local communication or team selection. These primitives are translated approximately into individual behaviors via a runtime library. In addition to swarm behavior primitives, scripting languages offer also primitives that model individual behaviors. Unfortunately, scripting languages are still in their infancy and can not be used in a broad number of applications.

2.4 Automatic design

In automatic design, the problem of designing the control software is cast into an optimization problem. In other terms, an automatic design method uses an optimization algorithm to search the space of all instances of control software that it can possibly produce, with the goal of finding one that maximizes an appropriate performance measure. Automatic design methods can be divided in two classes: *off-line* and *on-line* methods.

2.4.1 Off-line methods

In off-line methods, the design process takes place in a preliminary, dedicated phase: the design phase. The design phase occurs and terminates before the robot swarm is deployed in its operational environment. Within the design process, an off-line method evaluates a relatively large number of different instances of control software. Typically, the evaluation of an instance of control software is performed via a computer-based simulation. On the one hand, simulation enables a faster-than-real-time evaluation and, on the other hand, it prevents that robots are damaged by a possibly low quality instance of the control software.

Evolutionary robotics. Evolutionary robotics (Nolfi and Floreano, 2000) is the most studied automatic design approach in swarm robotics. Typically, in evolutionary robotics an evolutionary algorithm is used to optimize the parameters and possibly the structure of a neural network that takes as an input sensor readings and returns actuation commands. Originally, evolutionary robotics was successfully applied in single robot scenarios and then adopted in swarm robotics.

The adoption of the evolutionary robotics approach in swarm robotics goes under the name of *evolutionary swarm robotics* (Trianni, 2008). In the following, we present a number of notable achievements in evolutionary swarm robotics. For comprehensive reviews of the (single-robot) evolutionary robotics literature, we refer the reader to Bongard (2013); Trianni (2014); Doncieux and Mouret (2014); Silva et al. (2015a).

Most of the works in evolutionary swarm robotics share a set of common characteristics: (1) The swarms produced are behaviorally homogeneous, that is, all the robots of the swarm execute identical copies of the same control software. (2) The objective function that is optimized during the design process is globally and centrally evaluated, that is, it is computed considering the performance of the swarm as a whole. (3) The optimization algorithm adopted within the design process is a classical evolutionary algorithm that features elitism, recombination and mutation.

(4) The size of the population ranges from 50 to 200 individuals, with 100 individuals as the most common value. (5) The number of performance assessments for each instance of the control software ranges from 3 to 100, with 10 as the most common value. Each assessment differs from the others in the initial conditions—i.e., position of the robots and characteristics of the environment. (6) To take into account stochasticity, different independent runs of the evolutionary algorithm are performed. The most common number of independent runs is 10. (7) Unfortunately, the assessment of the obtained control software in reality is not always performed. Many studies present results in simulation only. When robot experiments are performed, they are often only isolated demonstrations rather than a structured empirical analysis aimed at producing statistically significant results.

Since the introduction of evolutionary swarm robotics, most of the research effort aimed at showing the feasibility of the approach and investigating whether a particular collective behavior can be obtained via artificial evolution. Quinn et al. (2003) were the first to adopt the evolutionary approach in the context of swarm robotics. The authors obtained a coordinated motion behavior by using an evolutionary algorithm to optimize control software based on a neural network. Robot experiments were performed with three Khepera robots. More than a decade after the publication of this work, some aspects of the experimental design might appear unusual. For example, the performance of each instance of control software produced within the design process was assessed via 60 evaluations. In the final stage of the design process, the number of evaluations was increased to 100. At the end of the design process, 10 different instances of control software were tested in robot experiments. The very best instance was tested through 100 runs. Following this seminal work, a number of robot swarms designed via evolutionary robotics have been described in the literature. For instance, Christensen and Dorigo (2006) showed how to use evolution to obtain a swarm of robots that is able to perform hole-avoidance and phototaxis at the same time. The study includes a comparison between three different evolutionary algorithms: a classical evolutionary algorithm, evolutionary strategy, and a co-evolutionary genetic algorithm. For each algorithm, 20 independent runs were performed. Evolutionary strategy yielded the best performance. The best instance of control software produced by evolutionary strategy was tested in experiments performed with a group of three s-bot robots. Under a similar setting, Baldassarre et al. (2007) obtained coordinated motion with a swarm of four physically connected s-bot robots. The best instance of control software obtained across 20 independent runs of the design process was tested on a group of four s-bot robots. In the experiments performed by the authors, the control software obtained via computer-based sim-

ulations performed well in reality without the need of any adjustment. Moreover, the same control software performed well also when the robots had to navigate on rough terrains. Hauert et al. (2008) used evolutionary robotics to obtain the control software for a swarm of aerial robots that were required to establish a wireless communication network. Experiments were performed in simulation only. Trianni and Nolfi (2009) studied the design of a self-organizing behavior via evolution. The experimental analysis was performed mostly in simulation and involved 20 independent runs of the design process. The best instance of control software obtained across the 20 runs was tested on groups of two and three s-bot robots. Waibel et al. (2009) investigated the influence on performance of different selective pressures (individual and collective) and different team compositions (homogeneous and heterogeneous swarms). The experimental analysis was conducted on three different variants of a foraging task. For each variant, four different combinations of selective pressures and team compositions were tested via 20 independent runs of the design process. Experiments were conducted both in simulation and with ten Alice robots.

Recently, the research in evolutionary swarm robotics has been influenced by the current trends in evolutionary computation. The studies on *novelty search* and *multi-objective optimization* are worthy of mention. Novelty search (Lehman and Stanley, 2011) is an approach to evolutionary computation that promotes diversity instead of performance. Results indicate that novelty search is robust to issues that affect the classical evolutionary approach, including premature convergence and stagnation. Gomes et al. (2013) introduced novelty search in the context of swarm robotics. They used novelty search to automatically develop control software for aggregation and resource sharing. The study includes a comparison with a classical evolutionary approach and with an hybrid approach that combines a classical approach and novelty search. Experiments were conducted in simulation only.

Multi-objective optimization focuses on problems in which multiple, possibly conflicting objectives are to be achieved. In evolutionary robotics, multi-objective optimization might be of interest in two contexts: (i) The problem is naturally formulated as a multi-objective problem. For example, the task is composite and the robots are required to accomplish multiple sub-tasks, each associated with a performance measure (Capi, 2007). (ii) The designer wishes to promote a specific property that can be characterized via a measurable quantity. For example, the designer wishes to increase the chance that the obtained control software works properly in reality (Koos et al., 2013b), enforce a low complexity of the control software produced by the optimization process (Teo and Abbass, 2004, 2005), or increase the robustness against hardware failures and environmental variability (Lehman et al.,

2013; Koos et al., 2013a). Trianni and López-Ibáñez (2015) were the first to study the application of multi-objective optimization in the context of evolutionary swarm robotics. They presented results on two swarm robotics tasks: flocking and an idealized version of the stick-pulling experiment. Experiments were conducted in simulation only.[2]

Other approaches. A number of notable studies depart from the classical evolutionary robotics tradition by adopting (i) control software architectures other than monolithic neural networks and/or (ii) optimization algorithms other than evolutionary computation. In particular, some authors studied the possibility of using an optimization algorithm to fine-tune a parametric control architecture that features only a small set of parameters. Hecker et al. (2012) obtained a foraging behavior by using artificial evolution to optimize the parameters of a probabilistic finite state machine. Experiments were conducted with a group of three custom-made two-wheeled robots. The objects to be retrieved were emulated via RFID tags placed on the ground. The tags could be sensed by the robots via an onboard RFID reader. Gauci et al. (2014a) obtained an object clustering behavior by using an evolutionary strategy to optimize a simple control architecture that features six parameters. The simplicity of the control architecture arises from the fact that the control architecture controls the speed of the two wheels of the robot exclusively based on the readings of a single three-state *line-of-sight* sensor. The line-of-sight sensor indicates whether the robot faces (i) another robot, (ii) an object to cluster, or (iii) neither of them. On the robot, the line-of-sight sensor was implemented using the frontal camera and a color detector that recognizes robots that are green and objects that are red. Experiments were conducted with a group of five e-puck robots. Ten independent runs of the design process were performed in simulation. For each run, the best instance of control software obtained was tested in reality. Gauci et al. (2014b) optimized the same control architecture via exhaustive search to obtain self-organized aggregation. In this second study, no object is present in the environment and the line-of-sight sensor can be therefore in only two states: it indicates whether the robot faces another robot or not. Due to the even simpler control architecture adopted in this second study, the design space counts only 194 481 different instances of control software that can be possibly generated. The reduced size of the design space allows for exhaustive search in simulation. Experiments were conducted with a group of forty e-puck robots: the best performing instance of control software found via exhaustive search was tested 30 times on the robots. Ferrante et al. (2015) used a method based on

[2]The authors presented results also on a single-robot task.

grammatical evolution (Ferrante et al., 2013) to evolve modular control software to tackle task specialization in a robot swarm. In particular, the authors highlighted the environmental conditions that are necessary for task specialization to emerge. Experiments were performed in simulation only.

Other authors proposed the adoption of a modular control architecture: the control software is obtained by combining modules obtained either via manual or automatic design. Duarte et al. (2014a) presented an approach based on the hierarchical decomposition of complex behaviors into simpler behaviors that can be generated either manually or via evolutionary robotics. The simple behaviors are then combined using a high-level neural networks. The approach was illustrated in simulation only, on an object retrieval task. The results indicate that the approach outperforms the classical, monolithic evolutionary approach. Similarly, Duarte et al. (2014b) used hierarchical decomposition to obtain control software for a patrolling task. The control software comprises low-level behaviors (go to waypoint, patrol, pursue intruder) and one module that arbitrates the low-level behaviors. The low-level behaviors were implemented as neural networks and were obtained via evolution. The arbitrator was manually written. The resulting system was tested in simulation only. The downside of the approach is that hierarchical decomposition is task dependent and has to be performed manually by the designer. In a successive study, the authors designed via artificial evolution control software for a swarm of ten aquatic robots (Duarte et al., 2016; Christensen et al., 2016). The task to be performed by the swarm comprised four different sub-tasks: homing, dispersion, clustering and area monitoring. The aggregated control software was obtained by sequentially combining the instances of control software developed for each sub-task. Experiments were performed in a $330\,\mathrm{m} \times 190\,\mathrm{m}$ waterbody next to the Tagus river in Lisbon, Portugal. The results show that the control software produced via the proposed method crosses the reality gap nicely. Our original method, AutoMoDe, falls in this category. For details on AutoMoDe we refer the reader to Chapter 3.

2.4.2 On-line methods

In on-line methods, the design process takes place when the robot swarm has been already deployed in its operational environment. On-line methods, as opposed to off-line methods, can benefit from the availability of information on the actual operational environment that would be unavailable at off-line design time. Moreover, one could expect that an on-line method can cope with changes in operating conditions and adapt to contingencies. In other words, on-line methods aspire to produce a design that is more tailored to the specific mission than the one produced by an off-

line method. On the downside, on-line methods are likely constrained to a reduced search space with respect to off-line methods because of two main reasons: (i) as the design process is performed by the robots themselves while they are operational, computational resources and time are limiting factors; and (ii) candidate designs that are potentially dangerous for the robots should be *a priori* removed from the search space to avoid jeopardizing the mission. Moreover, in a realistic on-line setting, the design process is fully distributed on the robots and cannot rely on any centralized entity to measure performance and guide the search, as it can in an off-line setting. The design process is therefore constrained to use performance indicator that can be evaluated in a distributed and local way. Whether on-line methods are more or less effective than off-line methods, which are the features of a mission that are better handled by on-line or off-line methods, and whether on-line and off-line methods can be effectively combined are empirical questions that require further research to be answered convincingly.

In the works on the on-line automatic design of control software for robot swarms, we can highlight some typical characteristics: (1) Each robot of the swarm explores a portion of the search space. Typically, each robot evaluates asynchronously a subpopulation of instances of control software and keeps track of their performance. (2) The robots exchange information to co-ordinate the search process. Although the way in which information is exchanged varies from work to work, a typical feature is that best performing instances of control software are shared between the robots. These instances are typically modified using mutation by the sender or by the receiver and become part of the subpopulation of instances explored by the receiver. (3) Robot swarms are *de facto* behaviorally heterogeneous, that is, at any given moment each robot executes a different instance of control software. This does not necessarily exclude that the robots can eventually converge to executing the same instance of control software. (4) As each robot of the swarm is required to evaluate instances of control software, the objective function must be computable relying only on information that is locally available to the robot. This restricts the class of tasks that are tackled in the literature (and that can be possibly tackled) using the on-line approach. The tasks that can be tackled appear to be simpler and less diverse than those that are tackled in the off-line approach. (5) Unfortunately, experiments in reality are not always performed. Some studies present results in simulation only. We can identify four different ways in which experiments are performed: a. experiments are conducted in simulation only; b. the design of the control software is conducted in simulation and then a subset of the best instances of control software are tested on the robots (during the test the robots do not perform on-line design);

c. the design of the control software is conducted partly in simulation and partly on the robots. The obtained control software is tested on the robots; d. the design of the control software is conducted exclusively on the robots. The obtained control software is tested on the robots.

One of the first studies proposing an on-line method for multi-robot systems was published by Parker (1996b, 1997). The author proposed L-ALLIANCE, a behavior-based control architecture that allows the on-line optimization of some internal parameters of the control software. No robot experiment is described in the articles in which the method is introduced. Matarić (2001) surveyed various behavior-based control architectures that allow the on-line optimization of parameters. Lee and Arkin (2003) applied learning momentum (Clark et al., 1992) to a multi-robot system. The task that they considered is an abstraction of capture-the-flag. Experiments were performed in simulation only.

A number of significant studies on on-line automatic design belong to the *embodied evolution* approach (Watson et al., 1999, 2002). In embodied evolution, an evolutionary algorithm that is in charge of optimizing the control software is executed in a distributed way. The computation is performed by the robots of the swarm while they are situated in the operational environment. Watson et al. (1999, 2002) developed a multi-robot system that is able to achieve phototaxis via embodied evolution. In this system, each robot broadcasts a mutated version of the best instance of control software it obtained, with a probability that is a function of its performance. Experiments were conducted with eight custom-made two-wheeled robots. To overcome the limited duration of the batteries and to grant a sufficient amount of time to the design process, robots were connected to a power source using special hardware that makes contact with an electrified floor. Similarly, Usui and Arita (2003) applied embodied evolution to design a simple obstacle avoidance behavior on six Khepera robots. The study includes a comparison between a group of robots that exchange the best performing instances of control software they found and a group that does not. The results indicate that exchanging control software is beneficial. As in Watson et al. (2002), the limited duration of the batteries was overcome using special hardware that provides a continuous power supply, in this case a pantograph that makes constant contact with an overhead power line. Bianco and Nolfi (2004) proposed a method that allows robots to share their internal configuration via self-assembly. The method was tested in simulation only. König and Mostaghim (2009) departed from the classical approach based on neural networks by proposing a control architecture based on finite state machines optimized on-line via embodied evolution. The tasks considered were gate passing and collision avoidance. The experiments

were conducted in simulation only.

A few studies were devoted to the investigation of fundamental scientific questions concerning the relationship between and the combination of learning and evolution. Wischmann et al. (2007) studied the relationship between learning and embodied evolution in a predator-prey scenario. Experiments were performed in simulation only. Elfwing et al. (2011) investigated the combination of reinforcement learning and embodied evolution in a survival scenario. Experiments were conducted in simulation and the control software obtained was then evaluated on two Cyber Rodent robots. The authors followed a hybrid procedure: the design was conducted in simulation using initially a group of four robots, and then a group of two. The resulting control software was eventually tested on the two Cyber Rodent robots.

More recent studies shifted the focus back to the development of effective on-line design methods. Bredeche et al. (2012) proposed MEDEA, a framework for onboard evolution of control software based on neural networks. In MEDEA, robots are able to adapt to environmental conditions. The experimental analysis was performed on two tasks (survival and two suns) using twenty e-puck robots. Some precautions were taken in the definition of the experiments: (i) to reach convergence within the duration of the batteries, experiments were conducted using a small search space; (ii) to allow the exchange of solutions between robots, a reliable local communication was emulated via Wi-Fi and information provided by a tracking system. Haasdijk et al. (2014) introduced a multi-objective variant of MEDEA called MONEE. MONEE allows a swarm of robots to adapt its behavior to the environment and to a given task at the same time. Experiments were performed in simulation only. Silva et al. (2015b) introduced odNEAT, a variant of real-time NEAT (Stanley and Miikkulainen, 2002) for the online and decentralized optimization of robot control software. Experiments on three multi-robot tasks (aggregation, navigation with obstacle avoidance, and phototaxis) were performed in simulation only.

A number of studies presented on-line design methods based on distributed implementations of particle swarm optimization (Pugh and Martinoli, 2009). Here, we mention only two recent studies. We refer the reader to Di Mario et al. (2015) for a more extensive review of the relevant literature. Di Mario and Martinoli (2014) applied distributed particle swarm optimization to the onboard optimization of control software for obstacle avoidance. The method was tested on a swarm of eight Khepera III robots. The experiments were performed in three different settings: (i) the control software was designed in simulation and then tested on the robots; (ii) the control software was designed and tested on the robots; (iii) the control software

was designed partly in simulation and partly on the robots, to be eventually tested on the robots. In a similar study, Di Mario et al. (2015) presented OCBA, a noise-resistant particle swarm optimization algorithm for the on-line automatic design of robot control software. In this case, the on-line adaptation was performed in simulation. The resulting control software was then tested on a swarm of four Khepera III robots.

2.5 Challenges

Although a number of promising methods for the automatic design of control software for robot swarms have been presented and discussed so far, a consolidated literature on the topic is still missing. In some sense, we could (somehow provocatively) argue that the state of the art in the automatic design of control software for robot swarms is undefined and is yet to be properly identified. As we have seen in Section 2.4, with the exception of a few rare cases, published studies do not produce comparisons between different methods. Moreover, the vast majority of the articles in which interesting automatic design methods have been introduced were tailored to answer scientific questions that do not directly belong in automatic design. For example, an important share of the articles devoted to evolutionary robotics were tailored to answer scientific questions related to the plausibility of biological models or the justification of animal behaviors in evolutionary terms. These questions are clearly extremely fascinating and relevant in absolute terms. Nonetheless, in many cases these questions have shadowed the core questions of the research on the automatic design of control software for robot swarms. Indeed, some key questions are hardly addressed in the current literature: *Which automatic method is the best under which conditions? How general is method X? How well does method X perform on different tasks? Which features of a task pose problems to method X? How well does a swarm produced by method X cross the reality gap?* Questions of this nature are in our opinion fundamental for the development of a mature science. In particular, the development of a solid, well-established, and consistently applied empirical practice to assess and compare methods is of paramount importance. Within the field of artificial intelligence, domains that rightfully qualify as mature science greatly invested in the development of a proper empirical practice. Think for example of machine learning and heuristic optimization—for example, evolutionary computation, particle swarm optimization, and ant colony optimization. In these domains, virtually all published articles propose an extensive experimental analysis and all newly proposed ideas are thoroughly compared empirically against the established state of the art. Moreover,

in these domains it is common to share benchmark problems, datasets, implementations, and results.

It is our contention that the research on the automatic design of control software for robot swarms cannot significantly progress further unless the research community endows itself with a strong, common empirical practice. Comparisons between different design methods should play a much more prominent role. At every moment in time, the research community should be mindful of what is the best method for a given problem, what is the relative performance of the available methods, and which are the relative strengths and weaknesses of each existing method. In other words, the state of the art should be precisely defined and every new proposed method should be compared with it.

In the following, we discuss four main issues to be addressed in order to define an empirical practice that is appropriate for the automatic design of control software for robot swarms.

Reference model. To be meaningful, a comparison of design methods must be performed under the same conditions for all methods under analysis. It appears obvious that all methods must be given the same resources: computation time, memory, simulator and simulation models, number and kind of CPUs, operating system and hardware infrastructure, etc. It appears also obvious that the different design methods under analysis must be requested to produce control software for the same robotic platform. This last requirement is less trivial than one might think. Indeed, we became convinced that simply stating which platform is considered in a study is not sufficient to guarantee that all methods under analysis operate under the same conditions. We argue that a more formal approach is needed: a *reference model* for the platform considered should be explicitly defined. The reference model should formally define the sensors and the actuators that the control software can access along with the relative value ranges. Ideally, the control software produced by the automatic design methods under analysis should interact with the platform hardware via a common API. This prevents that the experimenter introduces a bias by allowing a method to access resources or information that is not available to other methods or to use them in a more creative and profitable way.

Precise definition. An automatic design method should be precisely defined so as to enable the reproducibility of its results. In particular, an automatic design method should be univocally identified by a name, should be clearly defined in all its parts, and should properly pinpoint the reference model(s) to which it can be applied. Ideally, an implementation should be made publicly available. This

would guarantee that an automatic design method can be used by researchers other than those who originally proposed it, and can be therefore included in objective comparative studies performed by third parties. By studying the literature presented in Section 2.4, we realized that, to the best of our knowledge, automatic design methods proposed so far have been tested in a single study by authors who introduced them. In each study, researchers either present an original design method or develop a variant of a previously presented one. In the literature, there is no automatic design method that is used "as is" in multiple subsequent studies on different tasks—i.e., without undergoing any *ad hoc*, manual, per-task modification. Moreover, we are not aware of any case in which an automatic design method is included in a study performed by a third party. We find particularly revealing the fact that automatic design methods are not typically given a name by their proponents: it does not make much sense to name a method that is supposed to live within the time span of a single research study. We are quite sure that researchers would name the methods they propose if the practice in the domain were such that newly proposed methods were expected to be included in subsequent studies and applied to a number of different tasks, possibly by a third party. The fact that a method is typically tested on a single task in the original paper in which it is proposed makes it impossible to apprise its qualities as an automatic method. In particular, it makes it impossible to understand whether the method is an *ad hoc* solution for the single task considered or it is general enough and able to address a class of tasks. Clearly, a method that applies to a single task and needs to be manually re-instantiated and/or fine-tuned to be applied to another task cannot legitimately qualify as a automatic method. To serve the purpose of the research on the automatic design of control software for robot swarm, its experimental practice should take the above considerations into account. In particular, it should prescribe that an automatic design method is tested on multiple tasks without undergoing any *ad hoc*, manual, per-task modification. Moreover, each newly proposed method should be compared with those that have been previously proposed so that a clear picture of the state of the art is always available to the community. Independent evaluations performed by third parties would be extremely valuable.

Benchmarks. Another important issue that should be addressed by the community concerns the tasks on which automatic design methods should be tested and compared. A convincing and informative experimental analysis should ideally be based on a large set of different tasks. It should eventually allow the experimenter to make conclusions on the ability of the design methods under analysis

to produce control software for a generic task of interest. Obviously, whether a task can be possibly performed by a swarm of robots depends on the capabilities of the robots, as formally characterized by the reference model. In other terms, a reference model implicitly determines the class of tasks that can be performed by a swarm of robots conforming to the reference model itself. An informative comparison should be based on a representative subset of tasks sampled from this class. We are convinced that the definition of publicly available datasets of benchmark tasks would be highly beneficial for the research on the automatic design of control software for robot swarms and would significantly speed up its progress. Each benchmark task should specify its target platform(s) via a reference model. At each moment in time, the research community should have access to an up-to-date record of the performance yielded by each automatic design method that has been tested on each benchmark task. In this respect, having names that univocally identify design methods is paramount. It would be convenient to conceive programs that automatically generate benchmark tasks within a class. For example, consider a program that generates instances of a search and retrieve task by randomly sampling, according to appropriately defined probability distributions: (i) size and shape of the environment, (ii) number and position of the obstacles, (iii) number and the positions of the targets, (iv) initial placement of the robots, and (v) position of the safe area to which retrieved targets should be carried. Such a program could be used to generate multiple sets of tasks, all sharing the same statistical properties. The experimenter could obtain two disjoint sets: one to be used to automatically design control software, and one to be used to test it.

The definition of benchmark tasks to be used by the whole research community could be based on widely used and commercially available robots. Indeed, some platforms have been considered a *de facto* standard by many research labs for years. For example, the Khepera (Mondada et al., 1999) and the e-puck (Mondada et al., 2009b) have been adopted in a large share of swarm robotics studies. More recently, the kilobot (Rubenstein et al., 2014a) is becoming popular and the Thymio[3] appears to meet all the requirements to become widely adopted in swarm robotics research.

As an alternative, standard benchmark tasks could be implemented via remote controlled labs (e.g., see Zeiger et al., 2009; Kulich et al., 2013; Casini et al., 2014; Casan et al., 2015). For example, imagine a confined area in which a robotic arm can position obstacles and objects as remotely specified by the experimenter. A robot swarm could then act in the area operated by the control software to be

[3]https://www.thymio.org

tested. The robotic arm would be in charge of placing the robots at desired initial positions for each run and of recharging them when needed.

Clearly, we do not wish to go so far as to state that all the research in swarm robotics should be performed on standard benchmarks and be based on the aforementioned platforms or via remote controlled labs. Indeed, research on previously unexplored tasks and the development of original custom-made robots are essential elements for the advancement of the domain. Yet, we are deeply convinced that routinely testing new proposals on standard benchmarks and common platforms is the only way to consolidate results and to enable an ordered and structured accumulation of knowledge.

Robot experiments. A crucial aspect that has been often overlooked in the literature is the ability of a design method to overcome the reality gap, that is, the unavoidable difference between the models used in computer-based simulations and reality. The reality gap is an issue that is particularly relevant when studying off-line automatic design methods. Indeed, one of the major challenges for an off-line design method is to use computer-based simulations to design control software that, once installed on the robots, will behave as expected. Ideally, the robot swarm should behave in reality as its simulated counterpart. Unfortunately, this challenge is far from being overcome. Potentially, the reality gap is an issue also in the study of on-line methods as a large share of the research work on this methods relies on computer-based simulations. Due to the reality gap, automatic design methods (either on-line or off-line methods) cannot be properly assessed only via simulations, at least in this historical moment. We are firmly convinced that robot experiments should have a much more prominent role in the research on the automatic design of control software for robot swarms. They should be the core of the whole empirical practice in the domain. Increasing the prominence of robot experiment has unfortunately a downside: it raises the cost of research. Yet, we are convinced that, at least in this phase of the development of the domain, robot experiments are indispensable and their cost unavoidable. It is common in many domains that researchers need costly resources to perform their studies. For example, research in particle physics and in observational astronomy require having access to a collider and a telescope, respectively. These resources are expensive—much more than a robot swarm. Most research groups do not own them and need to get access to those shared by other groups. Even those that do own the resources, often are not fully satisfied with them and need to get access to alternative resources owned by other groups. Indeed, each collider and telescope is different (energy range, resolution, location, etc.) and researchers often need to

access more than one of them to confirm observations or to calibrate their methods and tools. It is therefore customary that these resources are shared in the context of international cooperation programs. Although the domains of particle physics and observational astronomy are deeply different in many respects from swarm robotics, we are convinced that the mechanisms used to share resources within these two domains could serve as a model also for our community. In particular, we envision international cooperation programs within the research community that investigates the automatic design of control software for robot swarms. Research groups could share their resources with partner groups so that each participant in the cooperation program can test their automatic design methods on different platforms and/or under different experimental conditions. When the research groups own identical robots, they could join their resources and perform experiments with a swarm larger than the one they own. The collaboration would be the ideal context to compare the different automatic design methods developed by the partners and conceive possible cross-fertilizations.

2.6 Summary

In this chapter, we gave a general introduction to swarm robotics with the goal of introducing the problem of automatic design of robot swarms. Swarm robotics is an appealing domain of robotics in which a given mission is tackled by a large group of robots, the swarm. The swarm is a distributed system based on individual autonomous robots that collaborate and interact without any kind of central control or knowledge. Notwithstanding the simplicity of the individual robot, the swarm is able to show quite complex collective behaviors. Swarm robotics is appealing because it promotes the development of systems that are fault tolerant, scalable and flexible. Such properties are desirable in many applications such as patrolling, warehouse automation, agriculture and medicine.

The main problem in swarm robotics is the design of the behavior of the individual robots: robot swarms are immaterial entities, and as such, cannot be programmed directly. This problem can be tackled using manual or (semi-)automatic design methods.

In this chapter, we reviewed the most notable achievements in the design of control software for robot swarms with a particular focus on automatic design methods. These achievements show that automatic methods are a viable and promising approach to the design of control software for robot swarms. Unfortunately, the literature on the automatic design of control software for robot swarms appears to be

scattered and composed of isolated contributions: no comparisons between design methods are provided and new ideas and methods are not properly assessed against a well-established state of the art. It is our contention that the lack of an empirical practice hinders the progress of the domain.

In the body of the chapter, we highlighted four issues that need to be addressed to establish a proper empirical practice for the automatic design of control software for robot swarms: (i) Every study that proposes or applies an automatic design method should clearly define a reference model for the robotic platform considered. (ii) Every automatic design method should be precisely defined in all its parts and parameters, and univocally identified by a name. (iii) Libraries of standard benchmarks should be defined and adopted by the community for assessing newly proposed methods and ideas. (iv) Robot experiments should be the ultimate way to assess methods for the automatic design of control software for robot swarms and should be an essential element of any research study in the domain.

We are convinced that a solid, well-established, and consistently applied empirical practice would allow the community to promote the best ideas proposed so far, to focus on promising directions, and to attract further researches and investments to the domain of automatic design of control software for robot swarms.

The achievements and the flaws found in the literature have a strong influence on this thesis. In particular, overcoming these flaws constitutes an important share of the original contribution of this thesis: For instance, in this thesis, we stress the importance of the definition of a reference model and we officially define what is, to the best of our knowledge, the first reference model in the swarm robotics literature. The original method proposed in this thesis is precisely defined and it is univocally identified with the name AutoMoDe. We define a proto-library of benchmarks that allows the performance assessment of design methods on different and clearly-defined tasks. For the performance assessment of the design methods under analysis we use an experimental protocol that relies heavily on robot experiments.

Chapter 3

From a novel perspective on the reality gap to AutoMoDe

In this chapter, we describe the intuition that is the starting point of the thesis: the reality gap bears a strong resemblance to the generalization problem faced in supervised learning. Following this intuition, we cast the reality gap in terms of the bias-variance tradeoff as formalized in the machine learning literature: past an optimal level, increasing the complexity of the approximator (or the design effort) is counterproductive because it hiders the generalization abilities of the approximator itself. We believe that the reality gap experienced in the automatic design of robot swarms is a similar phenomenon and we formulate two working hypotheses that will be corroborated in this thesis: hp1) past an optimal level, increasing the representational power of the control architecture is counterproductive; hp2) past an optimal level, increasing the design effort is counterproductive. In both cases, past an *a priori* unknown optimal level, the performance obtained in simulation and the one obtained in reality diverge: the performance in simulation increases while the one in reality decreases.

In this thesis, we will corroborate the two hypotheses. Hypothesis hp1 occupies a more prominent role as it is the inspiration for the introduction of AutoMoDe, the core contribution of this thesis. AutoMoDe is an automatic design method that, while having a control architecture with low representational power, is able to tackle tasks of interest in swarm robotics. We corroborate hypothesis hp2 in this chapter via an experiment with a swarm of e-puck robots.

The rest of the chapter is organized as follows: In Section 3.1, we describe the reality gap problem. In Section 3.2 we describe the theoretical framework and the hypotheses. In Section 3.3, we introduce AutoMoDe. In Section 3.4, we present the experimental study to corroborate hypothesis hp2. In Section 3.5, we conclude with

a summary of the chapter.

3.1 The reality gap problem

The *reality gap problem* (Brooks, 1992; Jakobi et al., 1995) is one of the major issues to be faced in evolutionary swarm robotics—and in all automatic design methods that rely on simulation. The reality gap is the intrinsic and unavoidable difference between reality and simulation. As a consequence of the reality gap, differences should be expected between how an instance of control software designed in simulation behaves in reality.

Several techniques have been proposed to mitigate this problem. For example, Miglino et al. (1995) increased the realism of simulation using samples of the responses of the robot's sensors and actuators; Jakobi (1997) suggested the inclusion of noise in the simulation of sensors and actuators and in the conditions experienced by the robots during the design process; more recently, Bongard et al. (2006) and Koos et al. (2013b) alternated simulation with tests on the physical system to correct the simulator nuisances. The reality gap problem is a specific instance of a wider problem related to the generalization abilities of evolutionary robotics—and of any automatic design approach—that is, the overfitting of the solution to the particular conditions encountered during the design process. By continuously refining the control software in a subset of the possible operating conditions, the control software is optimized "to match the specificities of the simulation, which differ from the real world." (Floreano et al., 2008).

Our contention is that the inability to generalize to unexperienced working conditions in the automatic design of robot swarms bears a similarity with the generalization problem faced in supervised learning.

3.2 Facts and Hypotheses

Neural networks have been studied for over seven decades, with alternating fortune —e.g., McCulloch and Pitts (1943); Rosenblatt (1958); Minsky and Papert (1969); Werbos (1974); Cybenko (1989). Around the year 2000, neural networks appeared to be superseded by other learning methods. They regained the general attention of researchers and practitioner in the last decade, thanks to the major success of deep learning—e.g., see Schmidhuber (2015). In the context of our reasoning, we are interested in scientific facts about neural networks and their generalization capabilities that where established mostly in the 1990's. In particular, we are interested in the

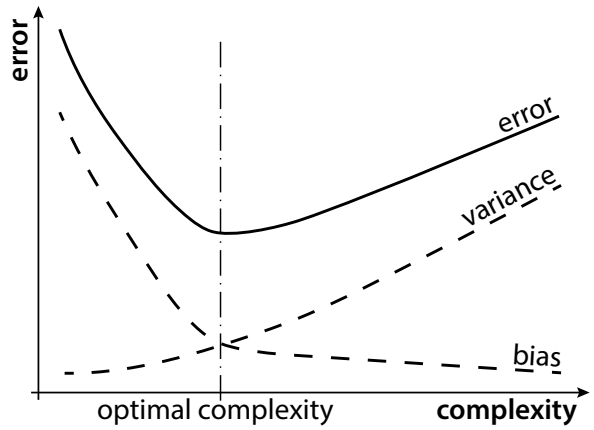


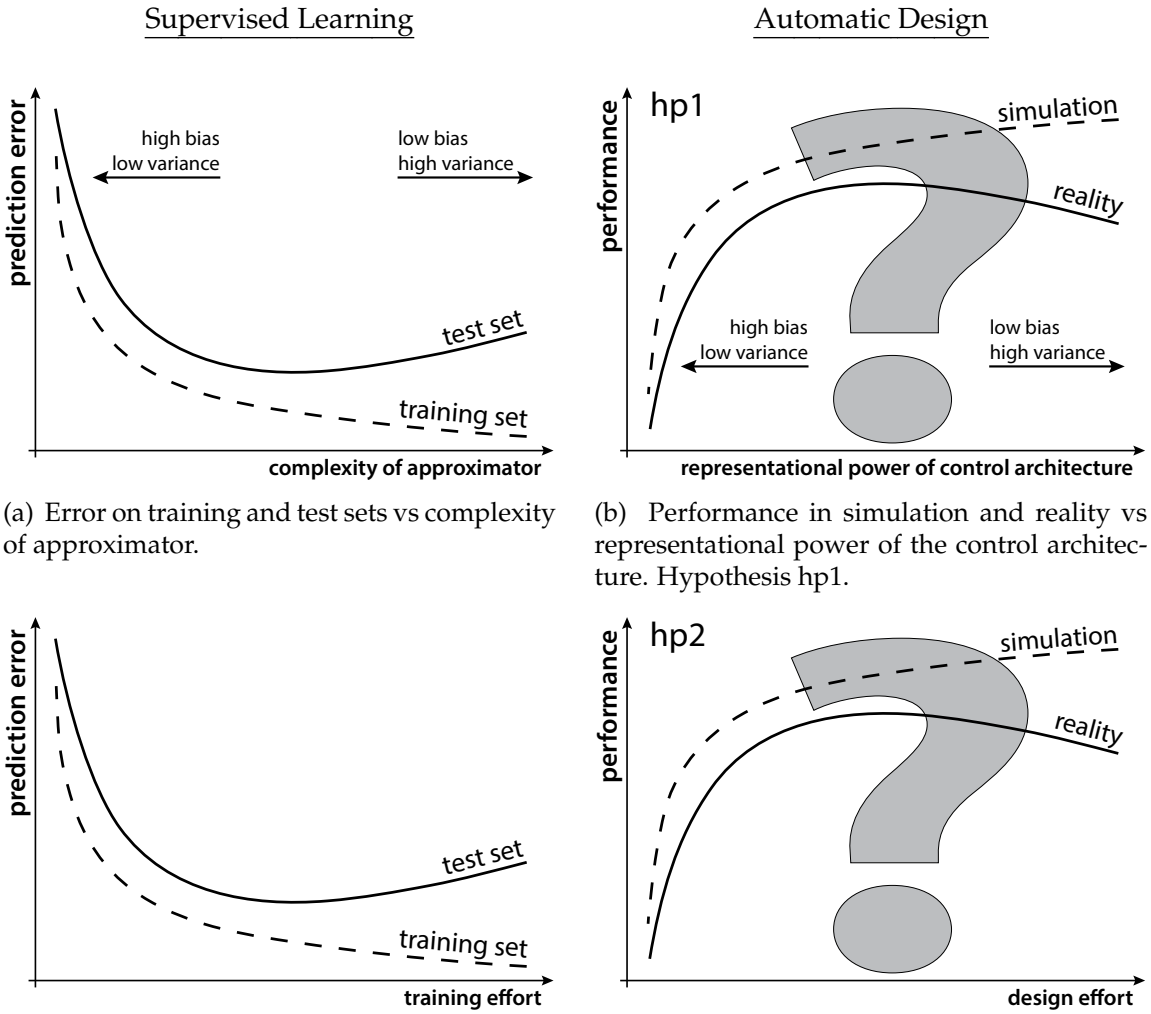
Figure 3.1: Decomposition of the error into a bias and a variance component.

relationship between prediction error and two characteristics: (1) the complexity of the neural network; and (2) the amount of training effort.

A fundamental result for understanding the relationship between error and complexity is the so called *bias/variance decomposition* (Geman et al., 1992).[1] It has been proved that the prediction error can be decomposed into a bias and a variance component. Low-complexity neural networks—i.e., those with a small number of hidden neurons and therefore low representational power—present a high bias and a low variance. Conversely, high-complexity neural networks—i.e., those with a large number of hidden neurons and therefore a high representational power—present a low bias and a high variance. As the bias and variance components combine additively, the error presents a U shape: for an increasingly large level of complexity, the error first decreases and then increases again. This implies that high complexity (i.e., high representational power and low bias) is not necessarily a positive characteristic: indeed an optimal value of the complexity exists. Beyond that value, prediction error increases. See Figure 3.1 for a graphical illustration of the concept. In other terms, a complex network (i.e., high number of neurons and therefore high representational power) is able to learn complex functions but then generalizes poorly. Indeed, it is an established fact that the higher the complexity of a neural network (as of any functional approximator), the lower is the error on the training set and the higher is the error on a previously unseen test set—provided that we are beyond the optimal complexity. This fact is graphically represented in Figure 3.2a: past the optimal level of complexity, the errors on training set and test set diverge.

Concerning the relationship between prediction error and training effort, a sec-

[1] In statistics, the bias of an estimator is the difference between estimator's expected value and the true value of the parameter being estimated. The variance represents the variability of the values estimated by the estimator. For a more advanced and general treatment of the issue, see also Wolpert (1997).



(a) Error on training and test sets vs complexity of approximator.

(b) Performance in simulation and reality vs representational power of the control architecture. Hypothesis hp1.

(c) Error on training and test sets vs training effort.

(d) Performance in simulation and reality vs design effort. Hypothesis hp2.

Figure 3.2: Conceptual relationship between the bias-variance tradeoff in supervised learning and in automatic design (a/b) and between *overfitting* in supervised learning and *overdesign* in automatic design (c/d).

ond important fact has been established, which goes under the name of *overfitting*—or alternatively overtraining. Overfitting is the tendency of a neural network (as of any functional approximator) to overspecialize to the examples used for training, which impairs its generalization capabilities. As a result of overfitting, one can observe that if the learning process is protracted beyond a given level, the error on the training and test sets diverge. Indeed, past an optimal level of the training effort, which is typically unknown *a priori*, the error on a previously unseen test set increases, while the one on the training set keeps decreasing. This fact is graphically represented in Figure 3.2c.

It should be noted that the two facts illustrated in Figures 3.2a and 3.2c are strictly related. The former considers the case in which the level of training effort is fixed and the complexity of the approximator is varied; the latter considers the dual case in which the complexity of the approximator is fixed and the amount of training effort is varied. In both cases, past an *a priori* unknown level of the independent variable, the error on the training and test sets diverge.

Several ideas have been proposed to deal with these facts and produce so called *robust* learning methods. The most notable ones are cross-validation and regularization techniques—e.g., see Stone (1974); Bauer et al. (2007). In the context of this chapter, it is worth mentioning a technique known as *early stopping*, which consists in halting the learning process before the error on training and test set start to diverge—e.g., see Morgan and Bourlard (1990); Prechelt (1998); Caruana et al. (2001); Raskutti et al. (2014).

As stated in Section 3.1, it is our contention that the reality gap problem faced in automatic design of robot control software is reminiscent of the generalization problem faced in supervised learning. If the two problems are indeed sufficiently similar, one should be able to observe in the automatic design the counterparts of the facts illustrated in Figures 3.2a and 3.2c. In particular, one should observe that the performance in simulation and reality diverge: (hp1) for an increasing level of representational power of the control architecture—Figure 3.2b; and (hp2) for an increasing level of the design effort[2]—Figure 3.2d. The only difference between Figures 3.2a and 3.2b (and between Figures 3.2c and 3.2d) is that the former concerns the *minimization* of error, while the latter the *maximization* of performance. On Figures 3.2b and 3.2d, we superimposed a large question mark to signify that these plots represent hypotheses, as opposed to the plots appearing on their left, which represent established scientific facts supported by a vast literature.

Hypothesis hp1 depicted in Figures 3.2b inspired us the definition of AutoMoDe, an automatic design method that has a lower representational power (i.e., lower complexity) than the neural networks typically adopted in evolutionary robotics. The expected result is that when comparing AutoMoDe with a control architecture with a higher representational power (i.e., a neural network), we expect the neural network to perform better than automode in simulation, but worse in reality. AutoMoDe, described in Section 3.3, represents the core contribution of this thesis.

Hypothesis hp2 depicted in Figure 3.2d inspired us in the definition of the experiment presented in Section 3.4. The goal of the experiment is to verify whether, for a sufficiently large design effort, the performance in simulation and reality of auto-

[2]the term *design effort* is the counterpart in the context of the automatic design of robot swarm of the term training effort used in machine learning.

matically designed control software tend to diverge. As this phenomenon would be the automatic design counterpart of overfitting, we shall call it *overdesign*.

3.3 Performance vs Representational Power: Introducing AutoMoDe

AutoMoDe (automatic modular design) is an automatic approach inspired by hypothesis hp1, which claims that high representational power of the control architecture can be counterproductive. AutoMoDe adopts a control architecture that features a lower representational power compared to the neural networks that are typically used in evolutionary robotics.

AutoMoDe generates control software in the form of a probabilistic finite state machine. We chose probabilistic finite state machines as a control architecture because they are commonly used in the manual design of robot swarms due to their modularity and readability. Probabilistic finite state machines are composed of *states* and *transitions*. In AutoMoDe, states are chosen among a set of preexisting *constituent behaviors* and transitions are defined on the basis of a set of preexisting *conditions*. In the following, we will collectively refer to constituent behaviors and conditions as *modules*. AutoMoDe automatically searches for the best combination of modules to perform a given task.

Each constituent behavior is an activity that the robot can perform. Constituent behaviors have a set of parameters that regulate their internal functioning. Parameters allow AutoMoDe to fine-tune constituent behaviors and fit different situations. Different instances of the same constituent behavior can be obtained by assigning different values to the parameters and can coexist in the same probabilistic finite state machine.

Conditions are used to trigger transitions from a constituent behavior to another one in response to a particular event. Similarly to constituent behaviors, conditions can be fine-tuned through a set of parameters and can be instantiated multiple times in the same probabilistic finite state machine.

The output of AutoMoDe is thus a combination of specific instances of modules where the parameters of the modules and the topology of the connections are optimized for the task at hand. AutoMoDe explores a search space that is represented by all the possible probabilistic finite state machines that can be obtained by instantiating and combining the given modules. Within AutoMoDe, the exploration of the search space can be performed using any optimization algorithm that is deemed appropriate.

In AutoMoDe, the fact that the control software is obtained by selecting, assem-

bling, and fine-tuning some given modules introduces a bias and reduces the representational power: the control software produced is *a priori* constrained to belong to the space of the finite state machines that can be composed out of the given modules. This limits the possibility to fine-tune the dynamics of the robot-robot and robot-environment interaction. As we will show in the experimental results presented in Chapter 4, if the set of modules is appropriately defined, the bias that is introduced reduces the variance and increases the generalization capabilities of the obtained control software.

3.3.1 The specialization of AutoMoDe

AutoMoDe is a general framework that needs to be specialized: i) AutoMoDe has to be adapted to the given robotic platform, and ii) the optimization process has to be defined. In the following, we will refer to the person that performs the specialization of AutoMoDe for a specific platform as the *expert*.

The expert specializes AutoMoDe for a specific platform on the basis of a *reference model*, an abstraction of the robotic platform that specifies in formal terms its characteristics and capabilities. The reference model defines the way in which we think of the robots and the way in which we intend the interaction of a robot with the environment and with other robots. In particular, the reference model defines an interface between the hardware layer and the logic layer represented by the control software. As an example, the reference model of a platform featuring an ambient light sensor could include the capability to distinguish between night and day. This capability could take the form of a Boolean variable that is updated by the hardware every, say, $100\,\mathrm{ms}$ and that can be read by the control software.

The reference model implicitly defines the tasks that can be performed using a swarm composed of robots of instances of the given robotic platform. For example, a task that requires separating blue objects from green objects cannot be performed by robots that are unable to distinguish green from blue. In order to undertand the relationship between tasks and reference model, we define the classes T_{SR} and B_{RM}: T_{SR} is the class of behaviors that allow the swarm to perform tasks relevant to swarm robotics; B_{RM} is the class of all the behaviors that can be produced by a robot swarm conforming to the reference model. The intersection of T_{SR} and B_{RM} gives the class T of behaviors that perform tasks relevant to swarm robotics and that can be produced by robots conforming to the given reference model. The class T is a proper subset of B_{RM}: indeed many behaviors contained in B_{RM} do not have any practical relevance to swarm robotics and therefore do not belong to T_{SR}.

On the basis of the reference model, the expert must produce the set of modules

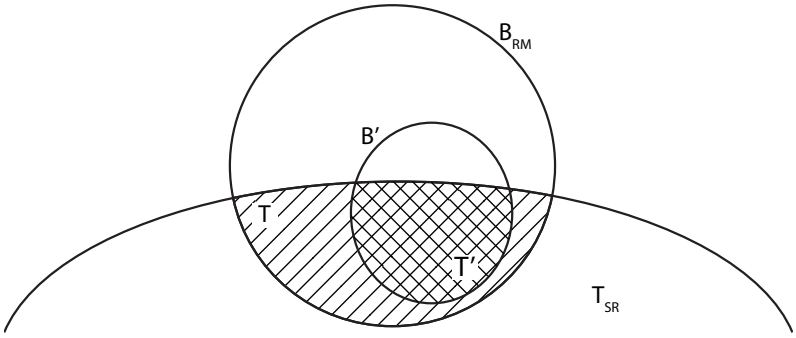


Figure 3.3: Graphical representation of the classes of behaviors and the corresponding tasks. T_{SR} is the class of behaviors that perform tasks of interest in swarm robotics. B_{RM} is the class of behaviors that can be produced by a swarm of robots conforming to the reference model. $T = T_{SR} \cap B_{RM}$ is the class behaviors that perform tasks relevant to swarm robotics and that can be produced by robots conforming to the reference model. B' is the class of behaviors that can be produced by the set of modules of the specialization of AutoMoDe. Assuming that the expert implements the modules correctly, $B' \subset B_{BM}$ and the specialization of AutoMoDe has a relatively low representational power. $T' = T \cap B'$ is the class of behaviors that perform relevant tasks and that can be obtained by combining the set of modules. In the ideal case, $T \equiv T' \equiv B'$, the specialization of AutoMoDe has the lowest representational power that allows to cover all the possible tasks of interest, given the reference model at hand.

that will be used by the specialization of AutoMoDe that is intended to produce control software for the corresponding platform. The set of modules implicitly defines the class B' of behaviors that can be produced by a swarm of robots whose control software is obtained by assembling the modules themselves. The relationship between the classes B_{RM} and B' defines the representational power of the obtained control architecture: assuming that the expert implements the modules correctly, it has to be expected that many behaviors belonging to B_{RM} that do not have a practical relevance will not belong to B'. In this case, B' is a proper subset of B_{RM} and consequently, the obtained control architecture features a reduced representational power. The intersection between the classes B' and T reppresents the class T' of behaviors that can be designed by the specialization of AutoMoDe and that perform tasks of interest. Figure 3.3 gives a graphical representation of the different classes.

In the ideal case, the set of modules perfectly and exhaustively exploits all the capabilities provided by the reference model and $T' \equiv T$. In practice, it has to be expected that the set of modules produced by the expert fails to suitably exploit some of the capabilities provided by the reference model, with the result that T' will be a proper subset of T. In this process of specialization, the experience of the expert plays an important role. Indeed, the expert defines the constituent behaviors and

the conditions by taking inspiration from those that have been previously presented in the literature and is guided by their personal understanding of what tasks are relevant in swarm robotics.

It has to be noticed that the specialization of AutoMoDe to a given reference model is task independent and has to be done only once: the same set of modules will be then used to design the control software for any task that one will subsequently wish to tackle with the given platform. It will be clearly unrealistic to expect that a specialization of AutoMoDe for a reference model is able to perform a task $t \notin T$. On the other hand, given the current understanding of swarm robotics, whether any task $t \in T$ can be performed via a set of modules produced by the expert—that is, whether $T' \equiv T$—is an empirical question.

Concerning the definition of the optimization process, a number of elements have to be selected including: the optimization algorithm to span the space of possible control software; a way to initialize the optimization algorithm; possible constraints on the finite state machine to be produced—e.g., the maximum number of states and of outgoing transitions for each state; and a way to assess the performance of a candidate control software. We foresee that, to assess the performance of control software candidates, the optimization process will typically rely on computer-based simulations. The specialization of AutoMoDe for a specific robotic platform involves therefore also the selection of an appropriate simulator of the robotic platform at hand.

In the thesis, we present two specializations of AutoMoDe. We present `Vanilla` and `Chocolate` in Chapter 4 and Chapter 5, respectively. Both `Vanilla` and `Chocolate` are specialization of AutoMoDe for the e-puck robot. The results presented in Chapters 4 and 5 corroborate hypothesis hp1 that claims that high representational power can be counterproductive: AutoMoDe, which adopts a control architecture with relatively low representational power, performs better than design methods that adopt control architectures with higher representational power.

3.4 Performance vs Training Effort: A proof-of-concept experiment

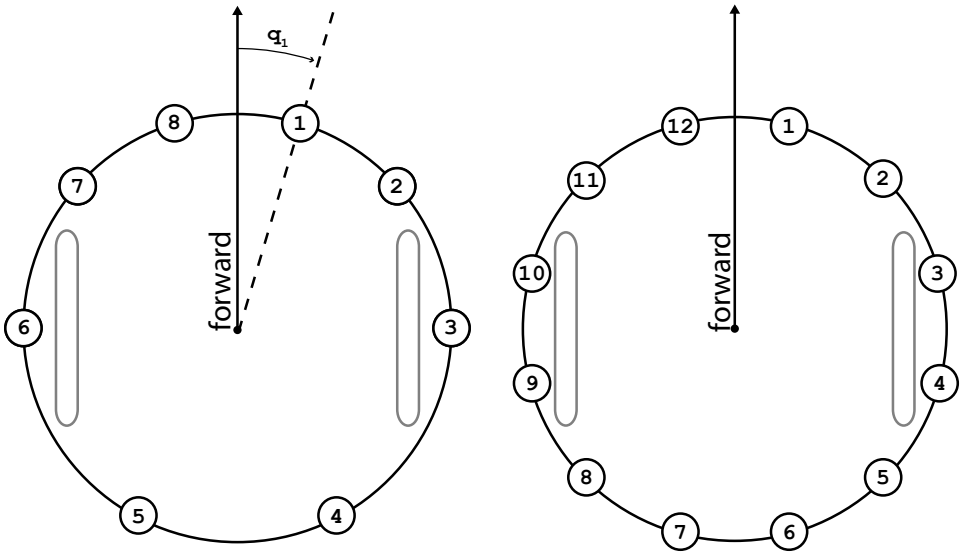
The aim of the experiment presented in this section is to corroborate hypothesis hp2 depicted in Figure 3.2d: past an unknown level of training effort the performances in simulation and in reality diverge. In this section, we present the material adopted in the experiment, the automatic design method, the task, the protocol, and the results.

Figure 3.4: Front and side view of an e-puck robot and a tag used to localize the robot via a ceiling-mounted camera. The epuck robot features an omnidirectional camera that is not used in the study—the reference model does not include it.

3.4.1 Robot platform and reference model

The experiment presented in this chapter is performed with a swarm of *e-puck* robots extended with the Overo Gumstick, a ground sensor, and a *range-and-bearing* board—see Figure 3.4 for a picture of the platform. The e-puck robot is a small wheeled robot designed for research and education (Mondada et al., 2009b,a). It is equipped with 8 IR transceivers that can be used as both light and proximity sensors. The IR transceivers are distributed around the body of the robot as shown in Figure 3.5. The Overo Gumstick is a single-board computer that allows the e-puck to run Linux. The ground sensor comprises 3 IR transceivers positioned in the front of the robot and pointed downward to measure the reflectivity of the ground. The range-and-bearing board (Gutiérrez et al., 2009) comprises 12 IR emitters and 12 receivers that are equally distributed along the perimeter of the board and pointed radially and outwards, on the horizontal plane. The range-and-bearing board allows the e-puck to send and receive messages within a range of about $0.7\,\mathrm{m}$. When an e-puck receives a message via the range-and-bearing board, it also obtains information about the relative position of the sender.

The reference model that we adopt for the platform described above is given in Table 3.1. We call this reference model $\mathrm{RM1}$. According to the reference model $\mathrm{RM1}$, the control software has a control cycle of $100\,\mathrm{ms}$. At each control step, the control software makes decisions based on the variables $prox_i$, $light_i$, gnd_i, n, r_m, and $\angle b_m$, which abstract the proximity, light, ground sensors, and range-and-bearing read-



(a) IR transceivers. (b) Range-and-bearing receivers.

Figure 3.5: Position of the IR transceivers and of the range-and-bearing receivers around the body of the e-puck. The e-puck features four IR transceivers in the front, two in the back, and two on the sides. The angle q_1 represents the angle at which the IR transceiver 1 is positioned with respect to the head of the e-puck. The e-puck features range-and-bearing receivers that are equally distributed around the body of the e-puck.

Table 3.1: Reference model RM1—$prox_i$ is the reading of the i-th proximity sensor and $\angle q_i$ is the angle at which the i-th proximity sensor is positioned with respect to the head of the robot; $light_i$ is the reading of the i-th light sensor and $\angle q_i$ is the angle at which the i-th light sensor is positioned with respect to the head of the robot; gnd_i is the reading of the i-th ground sensor; n is the number of robots perceived in the neighborhood; r_m and $\angle b_m$ are respectively the range and bearing of the m-th neighbor; v_l and v_r are respectively the speed of the left and right wheel; and $\bar{v}$ is the maximum speed of the robot. Sensors and actuators are updated with a period of 100 ms.

Sensors/Actuators	Variables
Proximity	$prox_i \in [0,1]$, $\angle q_i$, with $i \in \{1,2,\ldots,8\}$
Light	$light_i \in [0,1]$, $\angle q_i$, with $i \in \{1,2,\ldots,8\}$
Ground	$gnd_i \in \{0, 0.5, 1\}$, with $i \in \{1,2,3\}$
Range and bearing	$n \in \mathbb{N}$ and $r_m, \angle b_m$, with $m \in \{1,2,\ldots,n\}$
Wheels	$v_l, v_r \in [-\bar{v}, \bar{v}]$, with $\bar{v} = 0.16\,\mathrm{m/s}$

Period of the control cycle: 100 ms

ings. On the basis of the decision made, the control software can set the variables v_l and v_r, which abstract the actuators that operate on the wheels. Specifically, $prox_i$ can assume values in the range $[0, 1]$. It is equal to 0 when the i-th proximity sensor does not perceive obstacles within a $0.03\,\mathrm{m}$ range, while it is equal to 1 when the obstacle is closer than $0.01\,\mathrm{m}$. Variable $light_i$ can assume values in $[0, 1]$. It is equal to 0 if the i-th light sensor perceives only the ambient light, while it is equal to 1 when the sensor saturates.[3] Variable gnd_i can assume only three values. It is equal to 0, 0.5 or 1 when the i-th ground sensor detects, respectively, a black, a gray, or a white floor. Variable n is the number of robots in the neighborhood, as perceived via the range-and-bearing board. Variables r_m and $\angle b_m$ are the range and bearing of each robot m in the neighborhood. Variables v_l and v_r define the speed of the wheels. They are constrained in $[-\bar{v}, \bar{v}]$, with $\bar{v} = 0.16\,\mathrm{m/s}$ being the maximum speed of the e-puck.

3.4.2 Design Method

The design method used in the experiment presented in this chapter is `EvoStick`. We defined `EvoStick` as a typical evolutionary robotics method that implements the current best practice in the automatic design of robot swarms. As in the literature there is no definition of a "standard" method of evolutionary robotics, we defined it ourself. We called it `EvoStick` because our intention is to use it as a yardstick for the evaluation of other design methods. In particular, we will use `EvoStick` in Chapters 4 and 5 to evaluate the performance of `Vanilla` and `Chocolate`. `EvoStick` is the same method that we have already successfully used in the experiments presented in Francesca et al. (2012).

`EvoStick` is based on the reference model $\mathrm{RM1}$. Each robot is controlled by a fully connected, feed-forward neural network whose control cycle has a period of $100\,\mathrm{ms}$, as specified by the reference model. The neural network has 24 inputs, 2 outputs and no hidden units. The inputs are based on the capabilities defined by the reference model: 8 proximity sensors $prox_i \in [0, 1]$, $i \in \{1, 2, \ldots, 8\}$; 8 light sensors $light_i \in [0, 1]$, $i \in \{1, 2, \ldots, 8\}$; 3 ground sensors $gnd_i \in \{0, 0.5, 1\}$, $i \in \{1, 2, 3\}$; and 5 aggregated inputs from the range-and-bearing board. The aggregated inputs from the range-and-bearing board are: $\tilde{z}(n) = 1 - 2/(1 + e^n)$, where n is the number of robots perceived; and the scalar projections of $\vec{w}_{r\&b} = \sum_{m=1}^{n} (1/r_m, \angle b_m)$ on the four unit vectors that point at $45°, 135°, 225°$, and $315°$ with respect to the head of the robot. The activation of the output neurons is computed as the weighted sum of all input units plus a bias term, filtered through a standard logistic function. The outputs of

[3] $\mathrm{RM1}$ assumes that, besides the ambient light, at most one light source is present in the environment.

the neural network are scaled in $[-v_m, v_m]$, with $v_m = 0.16\,\mathrm{m/s}$, as specified by the reference model, and are used to set the speed of the two wheels.

The neural network is characterized by a set of 50 parameters. Each parameter is a real value in $[-5, 5]$. These parameters are optimized via a standard evolutionary algorithm. The cardinality of the population is 100. The initial population is randomly generated. At each iteration, each individual in the population is evaluated through 10 simulations performed using ARGoS' 2D dynamic physics engine (Pinciroli et al., 2012). The following population is generated via elitism and mutation. The elite composed of the 20 best individuals is included unchanged. The rest of the population is obtained from the elite via mutation: parameters are modified by adding a random value drawn from a normal distribution with mean 0 and variance 1. The evolutionary algorithm stops after a predefined number of iterations.

`EvoStick` implements the current best practice to mitigate the reality gap problem: A simulated uniform noise of 5% is added on the proximity, ground and light sensors and on the wheels actuator—as suggested by Jakobi et al. (1995). The noise of the range-and-bearing board follows a model defined using empirical data—as suggested by Miglino et al. (1995).

3.4.3 Task

A swarm of $N = 20$ e-pucks must perform an aggregation task. The environment is a dodecagonal arena of $4.91\,\mathrm{m}^2$ surrounded by walls—see Figure 3.6. The floor is gray, except two circular black areas, a and b. These areas have the same radius of $0.35\,\mathrm{m}$ and are centered at $0.60\,\mathrm{m}$ from the center of the arena. The swarm must aggregate on either a or b. At the beginning of the run, each robot is randomly positioned in the arena. The run lasts for $T = 240\,\mathrm{s}$ during which, the robots move in the arena according to their control software. At the end of a run, the performance of the swarm is computed using the objective function

$$F = max(N_a, N_b)/N, \tag{3.1}$$

where N_a and N_b are the number of e-pucks that, at the end of the run, are on a and b, respectively, and N is the total number of e-pucks. The objective function ranges from 0, when no e-puck is either on a or b, to 1, when all e-pucks are either on a or b.

3.4.4 Protocol

The experiment comprises two phases. In Phase 1, `EvoStick` is run 30 times for 256 iterations each. In order to evaluate the performance of the swarm at different levels

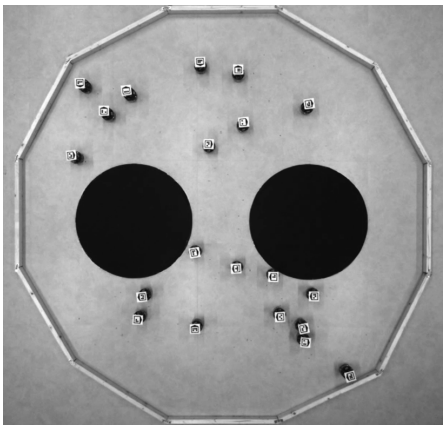


Figure 3.6: Arena and twenty e-pucks.

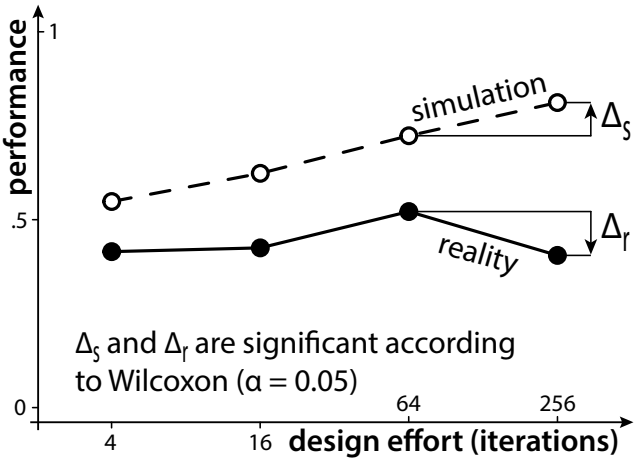


Figure 3.7: Results of the experiment.

of the design effort, for each run of `EvoStick` we collect the best neural network produced at four different stages: iteration 4, 16, 64, and 256. In Phase 2, we evaluate the neural networks collected in Phase 1. Each neural network is evaluated once in simulation and once in reality. The evaluation is performed under the same experimental conditions of Phase 1. Concerning the evaluation in reality, we reduced human intervention as much as possible to avoid biasing the results: (1) the control software is automatically uploaded to each e-puck via the infrastructure described in Garattoni et al. (2015); (2) the performance of the swarm is formally evaluated using the objective function defined in Equation 3.1 and is computed automatically via the tracking system described in Stranieri et al. (2013); (3) the tracking system is also used to automatically drive the robots to random initial positions at the beginning of each evaluation (Stranieri et al., 2013).

3.4.5 Results

Figure 3.7 summarizes the results. Visually, the two curves representing the average performance in simulation and reality closely resemble the hypothetical ones that we sketched in Figure 3.2d. In particular, between iteration 64 and 256 of the evolutionary algorithm, the performance in simulation increased while the one in reality decreased. To confirm that the observed trends are a genuine phenomenon rather than simply random fluctuations, we used the paired Wilcoxon signed rank test (with 95% confidence level) to analyze the performance difference between iteration 64 and 256. We did this for both curves. In both cases, the null hypothesis we tested is that the performance at iteration 64 and 256 is the same and that the ob-

served differences are the result of random fluctuations. As alternative hypotheses we used those suggested by Figure 3.2d: from iteration 64 to 256, the performance in simulation increases while the one in reality decreases. In both cases, the observations reject the null hypothesis in favor of the alternative.

3.4.6 Discussion

The results corroborate our hypothesis that the reality gap problem bears strong resemblance to the generalization problem faced in supervised learning. In particular, the results highlight a phenomenon that we shall call *overdesign*: as the training effort increases, past an optimal value, the performance that an automatically designed swarm obtains in reality diverges from the one it obtains in simulation.

The results presented in this study are preliminary, as they concern a single automatic design method and a single task. To establish overdesign as a scientific fact, further experimental work is needed and should involve a sufficiently large number of automatic design methods and tasks. Nonetheless, the results presented here are in line with our expectations and corroborate our hypothesis hp2. Moreover, they are in line also with similar results previously obtained in the automatic fine-tuning of the parameters of metaheuristics. Within that context, Birattari (2009) devised an experiment in which an iterated local search algorithm is fine-tuned on an instance of the quadratic assignment problem and is then tested on another instance of the same problem. The author recorded the cost of the best solution found by the algorithm on the two instances as a function of the tuning effort. The results show that, past an optimal value of the tuning effort, the costs diverge: on the tuning instance the cost keeps decreasing, while on the test instance it starts increasing. In the context of the automatic fine-tuning of metaheuristics, the phenomenon observed has been named *overtuning*.

These results suggest that one should check whether the control software obtained upon convergence of the design process is indeed the one that perform the best in reality. Moreover, these results suggest that a form of early stopping could be beneficial.

3.5 Summary

In this chapter, we presented the core intuition of the thesis: the reality gap problem bears a strong resemblance to the generalization problem of machine learning. To exploit this resemblance, we tackle the reality gap using tools that are similar to

those used when dealing with the generalization problem in machine learning. Taking inspiration from the concept of bias-variance tradeoff, we defined two working hypotheses, hp1 and hp2. Hypothesis hp1 relates the representational power of the control architecture with the performance of the robot swarm; hypothesis hp2 relates the design effort with the performance of the robot swarm. In both cases, past an optimal and *a priori* unknown level, increasing the representational power of control architecture (or the design effort) is counterproductive: while the performance in simulation increases, the performance in reality decreases.

We were inspired by hypothesis hp1 to devise AutoMoDe. AutoMoDe is an automatic design approach that aims at mitigating the effects of the reality gap by adopting a control architecture with limited representational power. In AutoMoDe, the control architecture is a probabilistic finite state machine where states and transitions are pre-existing parametric modules. AutoMoDe designs control software by automatically combining and instantiating such parametric modules. In Chapters 4 and 5 we will present two specializations of AutoMoDe and we will assess them empirically.

We corroborated hypothesis hp2 via a proof-of-concept experiment with a swarm of e-puck robots. In this experiment, `EvoStick`, a design method based on evolutionary robotics, designed control software for a swarm of 20 e-pucks in order to tackle an aggregation task. We observed that, past an optimal level of the design effort, the longer the design process is protracted, the better the performance of the swarm became in simulation and the worst in reality. We called this phenomenon *overdesign*. Overdesign is the automatic design counterpart of the well known overfitting problem encountered in machine learning.

The contribution of this chapter goes beyond the introduction of AutoMoDe and the experiment we devised to corroborate hypothesis hp2. In Section 3.4, we addressed some of the challenges presented in Section 2.5: We defined the reference model RM1 that formally describes the platform at hand. Moreover, we precisely defined a design method, `EvoStick`; we set its parameters; and we identified it univocally for future reference by giving it a name. Finally, we assessed `EvoStick` empirically with a robot experiment that comprises 120 runs with 20 e-pucks.

Chapter 4

AutoMoDe-Vanilla

This chapter describes the first study devoted to the empirical assessment of the core ideas of AutoMoDe. We perform this assessment using AutoMoDe-Vanilla (hereafter Vanilla), a proof-of-concept specialization of AutoMoDe for the e-puck robot. More precisely, Vanilla is specialized for a particular *reference model* of the e-puck. This reference model formally describes the characteristics of the version of the platform we consider and the functionalities that are made available to the control software.

We evaluate Vanilla using two tasks commonly studied in the swarm robotics literature: aggregation and foraging. The results obtained show that Vanilla automatically designs control software that allows the swarm to successfully accomplish the two tasks. Moreover, the control software designed by Vanilla is readable and understandable for a human.

The rest of the chapter is organized as follows: In Section 4.1, we introduce Vanilla. In Section 4.2, we describe the experimental protocol and the setup we use to evaluate Vanilla. In Section 4.3, we present the results and, in Section 4.4, we discuss them. Finally, in Section 4.5, we conclude with a summary of the chapter.

4.1 Proof of concept: AutoMoDe-Vanilla

Vanilla is a proof-of-concept specialization of AutoMoDe. Our goal in this chapter is not to define the ultimate automatic design method, but to show that the core ideas of AutoMoDe are valid. For this reason, Vanilla is unsophisticated in many respects such as the way in which probabilistic finite state machines are represented and optimized.

4.1.1 Robot platform and reference model

Vanilla is specialized for a swarm of e-puck robots extended with the Overo Gumstick, the ground sensor, and the *range-and-bearing* board—see Figure 3.4 for a picture of the platform. This platform is the same used in the experiments presented in Section 3.4 and it was formally defined in Section 3.4.1 via the reference model $\mathrm{RM1}$. The reference model $\mathrm{RM1}$ describes the set of sensors and actuators exposed to the control software. For convenience of the reader, we provide a brief sketch of the reference model $\mathrm{RM1}$ here and we refer the reader to Section 3.4.1 for the details.

According to the reference model $\mathrm{RM1}$, the control cycle has a period of $100\,\mathrm{ms}$. At each time step, the control software receives the readings through the variables $prox_i$, $light_i$, gnd_i, r_m, and $\angle b_m$ that abstract respectively, proximity, light, ground sensors, and the readings of the range-and-bearing board. Based on these variables, the control software decides the command values v_l and v_r to be applied to the wheel motors. We summarize all the variables of the reference model $\mathrm{RM1}$ in Table 3.1.

4.1.2 Module set

In Vanilla, the set of modules comprises six constituent behaviors and six conditions. Some of the modules have tunable parameters that are optimized by Vanilla. Vanilla selects, combines, and fine-tunes these modules to produce a finite state machine. This finite state machine operates with a period of $100\,\mathrm{ms}$, which is the same period at which sensors and actuators are updated, as specified by the reference model $\mathrm{RM1}$. At each control cycle, the constituent behavior associated with the current state is performed. Subsequently, each outgoing transition of the current state is considered and the corresponding condition is evaluated to decide whether the transition is enabled or not. In case no transition is enabled, no state transition occurs. If at least a transition is enabled, one of them is randomly selected and the current state is updated accordingly.

In the following, we describe the modules of Vanilla. We adopt the convention that tunable parameters are denoted by letters of the Greek alphabet.

Constituent behaviors

Exploration: the robot moves straight. If any of the proximity sensors positioned in front senses an obstacle, that is, if $prox_i \geq 0.1$ for any $i \in \{1, 2, 7, 8\}$, the robot turns on itself for a random number of control cycles chosen in $\{0, 1, \ldots, \tau\}$, where τ is an integer parameter in $\{1, 2, \ldots, 100\}$. The robot turns away from the direction faced by the proximity sensor that returned the highest value.

Stop: the robot stays still.

Phototaxis: the robot moves towards the light source, if perceived; otherwise, it moves straight. Obstacle avoidance is embedded: the robot follows the vector $\vec{w} = \vec{w}' - k\vec{w}_o$, where k is a hard-coded parameter whose value has been *a priori* fixed to 5 and $\vec{w}'$ and $\vec{w}_o$ are vectors defined as:

$$\begin{aligned} \vec{w}' &= \begin{cases} \vec{w}_l = \sum_{i=1}^{8} \left(light_i, \angle q_i\right), & \text{if light is perceived,} \\ \left(1, \angle 0\right), & \text{otherwise;} \end{cases} \\ \vec{w}_o &= \sum_{i=1}^{8} \left(prox_i, \angle q_i\right), \end{aligned} \tag{4.1}$$

where $\angle q_i$ is the angle at which sensor i is positioned with respect to the head of the robot.

Anti-phototaxis: the robot moves away from the light source, if perceived; otherwise, it moves straight. Obstacle avoidance is embedded: the robot follows the vector $\vec{w} = \vec{w}' - k\vec{w}_o$, where

$$\vec{w}' = \begin{cases} -\vec{w}_l, & \text{if light is perceived,} \\ \left(1, \angle 0\right), & \text{otherwise;} \end{cases}$$

and k, $\vec{w}_l$, and $\vec{w}_o$ are defined in phototaxis.

Attraction: the robot uses the range-and-bearing board to go in the direction of the robots in neighborhood, if any; otherwise, it moves straight. Obstacle avoidance is embedded: the robot follows the vector $\vec{w} = \vec{w}' - k\vec{w}_o$, where

$$\vec{w}' = \begin{cases} \vec{w}_{r\&b} = \sum_{m=1}^{n} \left(\frac{\alpha}{r_m}, \angle b_m\right), & \text{if robots are perceived,} \\ \left(1, \angle 0\right), & \text{otherwise;} \end{cases} \tag{4.2}$$

and α is a real-valued parameter in $[1, 5]$, and where $\vec{w}_o$ and k are defined in phototaxis.

Repulsion: the robot moves away from the other robots in its neighborhood, if any; otherwise, it moves straight. Obstacle avoidance is embedded: the robot follows the vector $\vec{w} = \vec{w}' - k\vec{w}_o$, where

$$\vec{w}' = \begin{cases} -\vec{w}_{r\&b}, & \text{if robots are perceived,} \\ \left(1, \angle 0\right), & \text{otherwise;} \end{cases}$$

and $\vec{w}_{r\&b}$ is defined in attraction, while $\vec{w}_o$ and k are defined in phototaxis.

Conditions

Black-floor: if $gnd_i = 0$, for any $i \in \{1, 2, 3\}$, the transition is enabled with probability β, where β is a parameter.

Gray-floor: same as black-floor but the prerequisite is that $gnd_i = 0.5$, for any $i \in \{1, 2, 3\}$.

White-floor: same as black-floor but the prerequisite is that $gnd_i = 1$, for any $i \in \{1, 2, 3\}$.

Neighbor-count: the transition is enabled with probability:

$$z(n) = \frac{1}{1 + e^{\eta(\xi - n)}}, \tag{4.3}$$

where n is the number of robots in the neighborhood, $\eta \in [0, 20]$ is a real-valued parameter and $\xi \in \{0, 1, \ldots, 10\}$ is an integer parameter. The transition is enabled with probability 0.5 if $n = \xi$. The parameter η regulates the steepness of the function $z(n)$ at $n = \xi$.

Inverted-neighbor-count: the transition is enabled with probability $1 - z(n)$, where $z(n)$ is defined in Equation 4.3.

Fixed-probability: the transition is enabled with probability β, where β is a parameter.

4.1.3 Optimization process

Concerning the optimization algorithm, `Vanilla` adopts F-Race (Birattari et al., 2002; Birattari, 2009), a racing algorithm originally developed for tuning metaheuristics. F-Race is able to select the most promising parameter setting out of an initial pool of candidates. F-race has been designed to handle stochasticity in the evaluation of candidates: in the case of swarm robotics, the performance of a control software candidate is highly stochastic and the ability of F-race to handle stochasticity appears to be appropriate in this context. Moreover, F-race is an extremely simple algorithm and in the context of this chapter we wish to keep the focus on the control architecture rather than on the optimization process.

Within the optimization process, control software candidates are evaluated via a computer-based simulation performed using ARGoS (Pinciroli et al., 2012), a multi-engine simulator of swarm robotics systems. In particular, we use ARGoS' 2D dynamic physics engine to model the robots and the environment.

Vanilla adopts the implementation of F-race provided by the *irace* package (López-Ibáñez et al., 2011) for R (R Development Core Team, 2008). F-Race iteratively evaluates a set of control software candidates, all generated randomly at the beginning of the optimization process, and discards the candidates that have a low expected performance, until a stopping criterion is met.

At iteration i, the candidates that have not been discarded in the previous $i-1$ iterations are evaluated on a test case. A test case is characterized by the specific initial condition—e.g., the initial position of the robots in the arena. To evaluate a control software candidate on a test case, F-Race performs a simulation run. After all candidates are evaluated, F-Race discards those candidates whose expected performance, as estimated on the i test cases considered so far, is statistically dominated by at least another candidate. The surviving candidates enter iteration $i+1$. The process stops either when a single candidate remains or when a predefined maximum number of evaluations has been performed. The maximum number of evaluations is the available *design budget* and is part of the specifications of an automatic design problem. It measures the computational resources available to produce the desired design.

In order to limit the representational power of the control software produced, we limit the number of states and conditions included: Vanilla can generate probabilistic finite state machines with up to 4 states, where each state can have up to 4 outgoing transitions. The candidates are generated at the beginning of the optimization process using the built-in sampling procedure provided by the *irace* package. This procedure samples the space defined as:

$$\left\langle \#S,\ S_i,\ S_i^p,\ \#N_i,\ N_{i,j_i},\ C_{i,j_i},\ C_{i,j_i}^p \right\rangle_{\substack{i=1,2\ldots,\#S \\ j_i=1,2,\ldots,\#N_i}} \tag{4.4}$$

where $\#S \in \{1, 2 \ldots, 4\}$ is the number of states of the probabilistic finite state machine; $S_i \in \{1, 2 \ldots, 6\}$ is the constituent behavior of state i;[1] S_i^p are the parameters of the constituent behavior S_i, if any; $\#N_i \in \{1, 2, \ldots, 4\}$ is the outdegree of state i, that is, the number of transitions outgoing state i; j_i is an index spanning the $\#N_i$ successors of state i; $N_{i,j_i} \in \{1, 2, \ldots, \#S\}$ is the j_i-th successor of state i; $C_{i,j_i} \in \{1, 2, \ldots, 6\}$ is the condition associated to the transition that connects state i to its j_i-th successor N_{i,j_i}; and C_{i,j_i}^p are the parameters of the condition C_{i,j_i}, if any. Figure 4.1 provide an illustrative example of a finite state machine sampled from the space defined in Equation 4.4.

The cardinality of the initial set of candidates is one sixth of the available design budget.

[1] State $i = 1$ is the initial state of the probabilistic finite state machine.

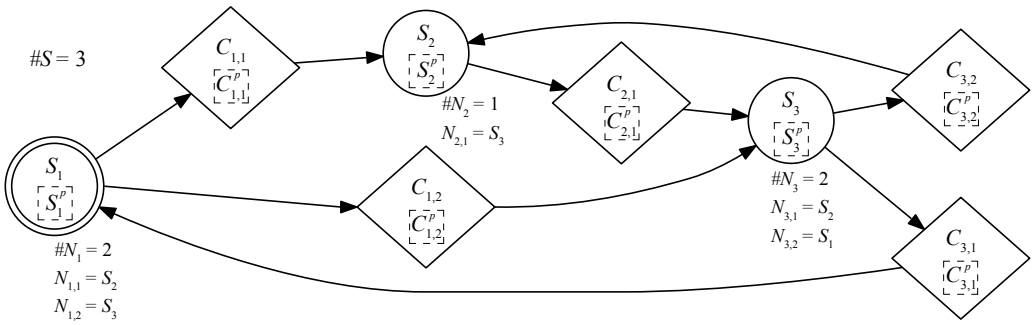


Figure 4.1: Example of a finite state machine generated by Vanilla. The notation adopted is the one defined in Equation 4.4. The circles represent the states and the double circle represents the initial state. The label S_i indicates the constituent behavior of the state i. The diamonds represent the transitions. The label C_i indicates the condition associated with the transition i. The labels contained in the dashed squares indicate the parameter set of the constituent behaviors or the transitions.

Vanilla implements the best practice commonly followed in automatic design to obtain control software that has the highest chance to overcome the reality gap: A simulated uniform noise of 5% is added on the proximity, ground and light sensors and on the wheels actuator—as suggested by Jakobi et al. (1995). The noise of the range-and-bearing board follows a model defined using empirical data—as suggested by Miglino et al. (1995). Moreover, the initial position and orientation of the robots at each iteration of the F-Race algorithm are randomly set by sampling from a uniform distribution.

4.2 Experimental setup

To assess the capabilities of Vanilla, we conduct a series of experiments in which Vanilla is used to automatically design the control software for robot swarms that are intended to perform two different tasks: *aggregation* and *foraging*. We selected these two tasks because they are common benchmarks in swarm robotics and it appears that they can be performed by a swarm of robots characterized by the reference model RM1. Regarding the aggregation task, the same task was performed in Francesca et al. (2012) by a swarm of robots characterized by a reference model that featured a subset of the capabilities of the reference model RM1. For this reason, we assume that the task can be performed by a robot swarm adopting the considered

reference model $\mathrm{RM1}$[2]. Regarding the foraging task, even though we do not have prior studies that show a swarm of robots that performs it, we assume that it can be performed by the considered robot swarm as it is similar to the aggregation task considered in this study. In particular, the main difference between the foraging task and the aggregation task is the presence of a light source in the foraging task. This light source, that can be used by the robots to orientate themselves, is made available to the robots via the variables of the reference model $light_i$. For this reason, we assume that also this task can be performed by a robot swarm adopting the reference model $\mathrm{RM1}$.

The experiments presented in this chapter adhere to a hands-off experimental protocol: we do not allow any human intervention in the automatic design process. The aim of the experiments is to assess the expected performance of `Vanilla` in designing control software for a robot swarm.[3] We run three sets of experiments that differ in the design budget, that is, the total number of simulation runs that `Vanilla` can use to design the control software. The three design budgets are: 10 000, 50 000, and 200 000 simulation runs. For each design budget, we execute 20 independent runs of `Vanilla` and we therefore obtain 20 instances of control software; we then assess the performance of these instances on the robots by performing a single run of each of them.[4]

The experimental protocol we adopt provides for a number of elements that reduce the intervention of the human experimenter during the evaluation of the control software produced by `Vanilla`: The control software obtained in simulation is automatically cross-compiled by ARGoS and copied on the e-pucks without any modification. The initial position and orientation of the robots is obtained by running the constituent behavior exploration—see Section 4.1—for a random number of seconds in $\{1, 2, \ldots, 20\}$. The performance of the robots is automatically computed by a tracking system (Stranieri et al., 2013) on the basis of data gathered via a ceiling camera.

[2]The reference model adopted in Francesca et al. (2012) does not include the variables r_m and $\angle b_m$.

[3]Because `Vanilla` is stochastic, reporting and discussing its expected performance appears to be the appropriate choice (Birattari and Dorigo, 2007).

[4]The reader might wonder why, in order to estimate the expected performance of `Vanilla` on each design budget, we repeat the design process 20 times and we test the resulting design on the robots only once. Due to time constraints, we have decided to run 20 robot experiments per design budget. Having fixed to 20 the total number of robot experiments, one might be tempted to consider alternative protocols: repeat the design 20 times and evaluate each resulting design 1 time; repeat the design 10 times and evaluate each resulting design 2 times; repeat the design 5 times and evaluate each resulting design 4 times; or even performing the design once and evaluate the result 20 times. Although all these protocols would produce an unbiased estimate of the expected performance of `Vanilla`, the one implemented in this study is the one that minimizes the variance of the estimate. A similar issue has been formally studied in the context of the assessment of stochastic optimization algorithms (Birattari, 2004, 2009).

Table 4.1: EvoStick—Partition of the available design budget.

budget	evolutionary algorithm	(iterations)	post-evaluation	(per individual)
10 000	8 000	(8)	2 000	(20)
50 000	40 000	(40)	10 000	(100)
200 000	150 000	(150)	50 000	(500)

With the aim of quantifying the effects of the reality gap, we perform a further independent assessment in simulation of the instances of control software produced by Vanilla. Also in simulation, each instance is assessed by performing a single run.

4.2.1 A yardstick: EvoStick

To put Vanilla into perspective, we compare Vanilla to EvoStick. EvoStick is the same method used in the experiments described in Section 3.4. In the following, we provide a brief description of EvoStick and we refer the reader to Section 3.4.2 for details.

EvoStick is based on the reference model RM1 that is the same one adopted by Vanilla. EvoStick is an implementation of the classical evolutionary swarm robotics approach: an evolutionary algorithm optimizes the feed-forward neural network that controls each robot. Inputs and outputs of the neural networks are defined on the basis of the reference model RM1. In particular, the neural network has 24 inputs: 8 readings from the proximity sensors, 8 from the light sensors, 3 from the ground sensors and 5 that are obtained by aggregating the range-and-bearing readings. The outputs are the commands to the two wheel motors.

The neural network has 50 real-valued parameters that are optimized by an evolutionary algorithm that features mutation and elitism. The evolutionary algorithm operates on populations of 100 neural networks. At a given iteration, each neural network is tested 10 times in simulation. The population to be tested at the subsequent iteration is created as follows: the 20 best performing neural networks (the elite), are included unchanged; 80 further neural networks are generated from the elite via mutation. Simulations are performed using ARGoS (Pinciroli et al., 2012).

The available design budget is partitioned in two parts: one for the evolutionary algorithm and one for the post-evaluation. See Table 4.1 for the details.

EvoStick implements exactly the same precautions adopted in Vanilla to reduce the risk of obtaining control software that does not overcome the reality gap—see Section 4.1.3. In the definition of EvoStick, we do not employ any further technique to overcome the reality gap. The adoption of other techniques is not *a pri-*

ori justified, and would possibly become apparent only *a posteriori*, when looking at the results obtained on each specific task to be performed. If additional techniques to overcome the reality gap are adopted on a per-task basis and *a posteriori* on the base of the results obtained, the overall design process would end up into a human-driven trial-and-error search and this would defeat the purpose of our experimental protocol and of our research as a whole.

We assess the performance of `EvoStick` with the same criteria and under the same experimental conditions that we adopt for the assessment of `Vanilla`. In particular, the two methods use the same simulator, design control software for the same robotic platform under the same reference model, and optimize the same objective function under the same environmental conditions. Concerning the robot experiments, to limit spurious effects of battery level and of other possible unforeseen ambiental contingencies, the order of the experiments is randomly generated and runs of instances of control software produced by `Vanilla` and by `EvoStick` are interleaved.

4.2.2 Tasks

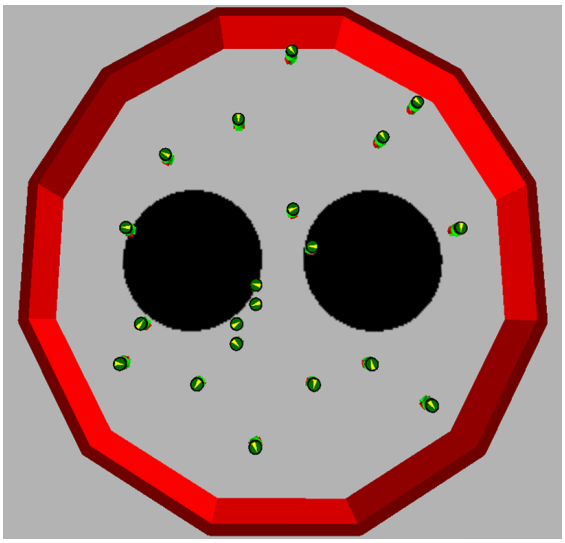
In all the experiments that we perform on the aggregation and the foraging tasks, the swarm comprises 20 e-puck robots. The available time to complete the task is $250\,\mathrm{s}$. The robots operate in a dodecagonal arena of $4.91\,\mathrm{m}^2$, surrounded by walls. For future reference, we define a coordinate system with origin in the center of the arena and x axis parallel to one of the sides. Coordinates in this system are given in meters.

Aggregation

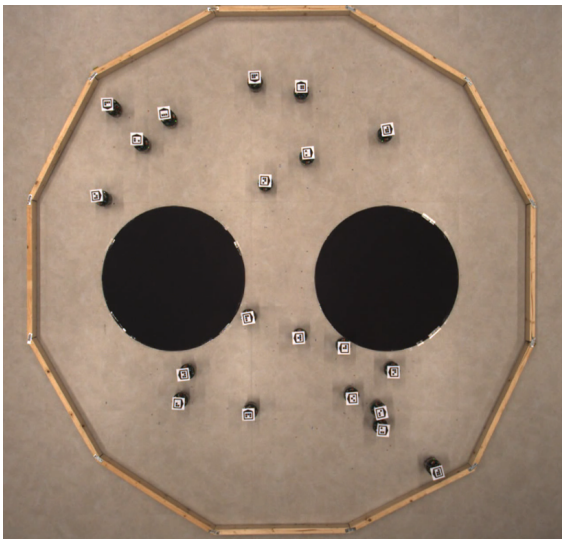
In the aggregation task, the swarm has to cluster in one of the two black areas of the arena's floor. The aggregation task is the same considered in the experiment presented in Section 3.4.

Figure 4.2 shows the arena for the aggregation task in both simulation and reality. The floor of the arena is gray and there are two black circular areas on the floor, namely a and b. The areas have the same radius of $0.35\,\mathrm{m}$ and are centered in $(0.6, 0)$ and $(-0.6, 0)$.

At the beginning of each run, the 20 e-puck robots are randomly distributed in the arena. The objective function is $F_{aggregation} = \max(N_a, N_b)/N$, where N_a and N_b are the number of robots that are in the black areas a or b at the end of the simulation, and $N = 20$ is the size of the swarm. This objective function equals 1 if all the robots



(a) Simulated arena and 20 e-pucks.



(b) Real arena and 20 e-pucks.

Figure 4.2: Arena for the aggregation task.

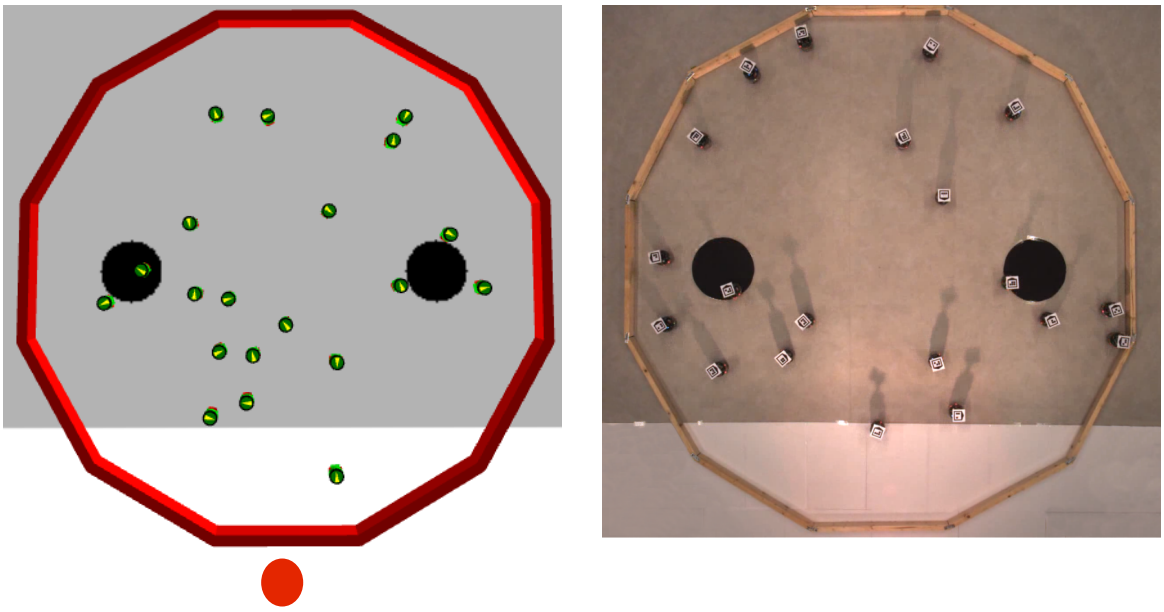
are aggregated in one of the two areas.

Foraging

In the foraging task, the swarm has to retrieve as many objects as possible from two sources and deposit them in the nest. Because the e-puck platform has no grasping capabilities, we abstract the actions of retrieving and depositing objects: We reckon that an e-puck has retrieved an object when it enters a source, which is represented by a black circle on the ground. Similarly, we reckon that an e-puck has deposited the object it is carrying when it enters the nest, which is represented by a white area. Our foraging task is inspired by the one presented in Liu et al. (2007). Figure 4.3 shows the arena for the foraging task in both simulation and reality. The two black areas have a radius of $0.15\,\mathrm{m}$ and are centered in $(0.75, 0)$ and $(-0.75, 0)$. Moreover, a light source is positioned behind the nest area, in $(0, 1.25)$ at $0.75\,\mathrm{m}$ from the ground. The objective function is $F_{foraging} = N_o$, where N_o is the total number of objects retrieved and deposited.

4.3 Results

We analyze the results of the experiments from two points of view: first, we estimate the performance of the control software produced by `Vanilla`, using the performance of the one produced by `EvoStick` as a yardstick; second, we compare the



(a) Simulated arena and 20 e-pucks. (b) Real arena and 20 e-pucks.

Figure 4.3: Arena for the foraging task. The red circle at the bottom of the simulated arena represents the light source.

performance of the control software produced by `Vanilla` in simulation and on the robots to evaluate the impact of the reality gap. Also in this case, we use `EvoStick` as a yardstick.

Moreover, for each task we provide a behavioral analysis of the swarms designed by `Vanilla` and by `EvoStick`. In this analysis, we also highlight the main differences between the behaviors observed in simulation and those observed in reality.

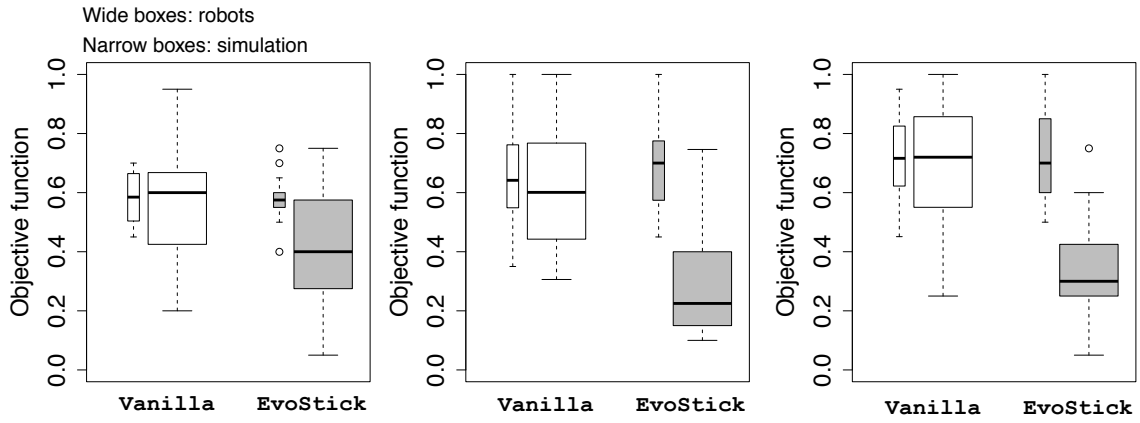
The complete set of experimental data and video recordings of all the robot experiments is available online (Francesca and Birattari, 2017).

4.3.1 Aggregation

Figure 4.4 shows the performance of `Vanilla` and of `EvoStick` in simulation and on the robots.

In all three sets of experiments, `Vanilla` designs robot swarms that perform better than those designed by `EvoStick`: for each design budget, the difference in performance between `Vanilla` and `EvoStick` is statistically significant according to the Wilcoxon test, with 95% confidence.

A visual inspection of the plots shows that `Vanilla` and `EvoStick` have similar performance in simulation. For what concerns the reality gap, in the case of `EvoStick` there is a large difference in performance between simulation and real-



(a) 10 000 design budget. (b) 50 000 design budget. (c) 200 000 design budget.

Figure 4.4: Aggregation—Performance of the control software obtained using different design budgets. The plot shows, for `Vanilla` and `EvoStick`, the performance of the 20 instances of the control software (one for each independent run) both in simulation (narrow boxes) and on the robots (wide boxes). A box comprises observations between the first and third quartile; the black horizontal line represents the median of the observations; the top whisker extends either to the largest observation or to 3/2 of the upper quartile (whatever is smaller); the bottom whisker is defined similarly; observations falling outside the extension of whiskers (if any) are outliers and are represented as circles.

ity. The control software obtained by `EvoStick`, even though it yields good results in simulation, is not able to reliably produce aggregation on the robots. On the contrary, in the case of `Vanilla` the difference between simulation and reality is small. A statistical analysis based on the Wilcoxon test is reported in Table 4.2. The data confirms that the mismatch between simulation and reality is lower in `Vanilla` than in `EvoStick`. The difference between the mismatch observed for `Vanilla` and for `EvoStick` is significant with a confidence level of at least 95%. The table indicates that the performance difference between simulation and reality observed in `EvoStick` increases with the design budget from 10 000 to 50 000, and then saturates. These observations could be explained as a result of overfitting: the larger the design budget, the longer the fine-tuning of the control software, and consequently the larger the risk of overfitting, up to saturation.

In all 60 runs, across the three budget levels, `Vanilla` has used the whole available design budget. As a result, `Vanilla` and `EvoStick` have always run the same number of simulated experiments.

Table 4.2: Aggregation—Estimated mismatch between simulation and reality, and corresponding confidence intervals according to the Wilcoxon test.

budget		estimated mismatch	95% confidence interval	
10 000	`Vanilla`	0.01	-0.06	0.08
	`EvoStick`	0.19	0.07	0.30
50 000	`Vanilla`	0.03	-0.12	0.17
	`EvoStick`	0.40	0.30	0.50
200 000	`Vanilla`	0.01	-0.11	0.12
	`EvoStick`	0.40	0.30	0.50

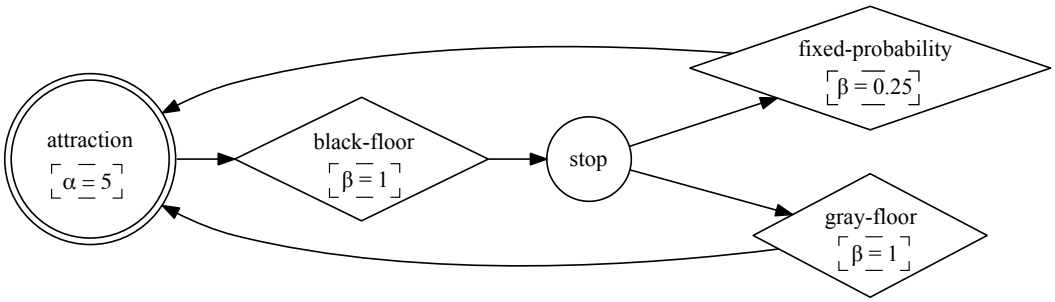


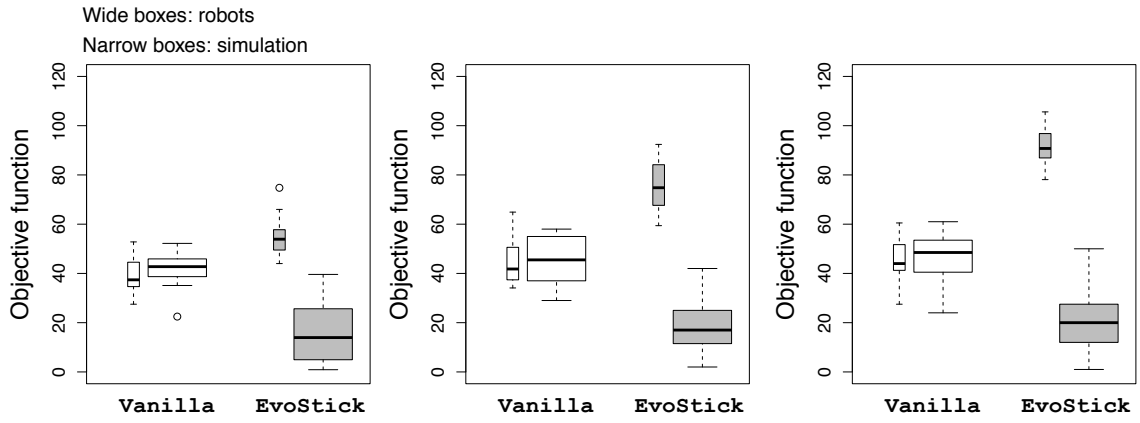
Figure 4.5: Aggregation—An instance of control software designed by `Vanilla`. The initial state is represented by the double-line circle. The robot performs attraction and moves toward the other robots. When it detects the black floor, it stops. In the stop state it checks for its transitions. It changes state when it detects the gray floor. It also starts moving, with a 0.25 probability, independently from the floor color.

Behavioral analysis

In this section, we describe the behavior of the control software designed by Vanilla and EvoStick for aggregation.

Vanilla. A feature of AutoMoDe is that the obtained control software is a probabilistic state machine, which is human readable. The 60 instances of the control software designed by Vanilla for the aggregation task have, with minor differences, the same structure. Figure 4.5 shows a representative instance. Each robot starts in the attraction state, that is, it moves toward other robots. With probability 1, a robot changes state to stop when it senses that the floor is black. In the stop state, the robot does not move. The robot then changes the state to attraction with a 0.25 probability or when it perceives the gray floor. This last event can happen because the robot is pushed outside the black area by other robots. The resulting collective behavior can be described as follows: Initially, robots tend to move in the direction of their neighbors and tend to cluster. Robots that enter a black area stop for some time and act as an attraction point for their neighbors. After a while, all robots are either in a black area or in its proximity. Eventually, most of the robots are attracted inside the black area where the majority of the robots are located. The behaviors observed in simulation and in reality are similar.

EvoStick. Because neural networks are not human readable, the only way to analyze the control software obtained by EvoStick is to instantiate it on the robots and observe the resulting behavior. The 60 instances of control software obtained by EvoStick show behaviors that are qualitatively similar to one another. When a robot is in the gray area, it moves following a circular trajectory. The radius of this trajectory decreases when the number of robots perceived increases. This movement allows the robots to create aggregates. When a robot enters a black area, the radius of its trajectory becomes very small, to the point that the robot almost rotates on the spot. In this condition, the robot leaves the black area only because pushed out by other robots. The robots that are in the black areas attract other robots. From our observations, it appears that the quality of the resulting collective behavior strongly depends on where the first aggregates are created: if these aggregates are far from the black areas, the robots are not able to find the black areas. In simulation, the behavior is qualitatively similar but the circular trajectories have a larger radius with respect to the ones observed in reality.



(a) 10 000 design budget. (b) 50 000 design budget. (c) 200 000 design budget.

Figure 4.6: Foraging—Performance of the control software obtained using different design budgets. The plot shows, for `Vanilla` and `EvoStick`, the performance of the 20 instances of the control software (one for each independent run) both in simulation (narrow boxes) and on the robots (wide boxes). See the caption of Figure 4.4 for an explanation of the graphical conventions adopted in the plot.

4.3.2 Foraging

Figure 4.6 shows the performance achieved on the foraging task by `Vanilla` and by `EvoStick`, both in simulation and on the robots. The obtained results show the same trend observed in the aggregation task.

In all three experiments, `Vanilla` designs robot swarms that perform better than those designed by `EvoStick`. Differences are all significant according to the Wilcoxon test, with 95% confidence.

Concerning the reality gap, it is interesting to note that `EvoStick` shows signs of overfitting: the mismatch between simulation and reality is large and increases with the design budget. On the contrary, `Vanilla` is able to design control software that is robust to the reality gap. The statistical analysis reported in Table 4.3 confirms these observations: In all the experiments, the difference between the mismatch observed for `Vanilla` and for `EvoStick` is significant with a confidence level of at least 95%. In the case of `Vanilla`, the simulation slightly underestimates the performance of the robots. This is shown by the fact that the expected difference is slightly negative. On the contrary, in the case of `EvoStick` the simulation greatly overestimates the performance of the robots.

In all 60 runs, across the three budget levels, `Vanilla` has used the whole available design budget. As a result, `Vanilla` and `EvoStick` have always run the same number of simulated experiments.

Table 4.3: Foraging—Estimated mismatch between simulation and reality, and corresponding confidence intervals according to the Wilcoxon test.

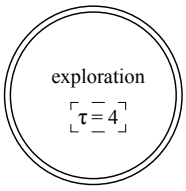
budget		estimated mismatch	95% confidence interval	
10 000	Vanilla	−3	−7	2
	EvoStick	39	31	47
50 000	Vanilla	−2	−6	3
	EvoStick	57	51	64
200 000	Vanilla	−1	−6	4
	EvoStick	70	64	79

Behavioral analysis

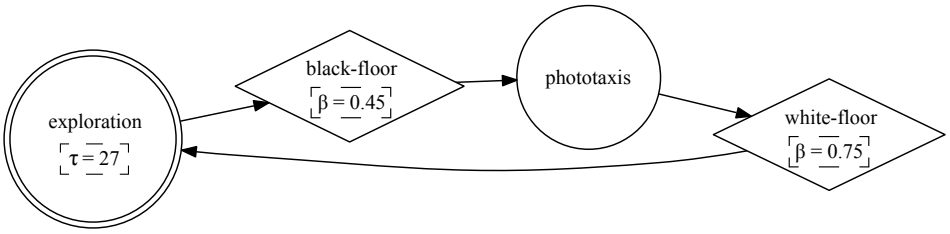
In this section, we describe the behavior of the control software designed by Vanilla and EvoStick for foraging.

Vanilla. The 60 instances of the control software obtained by Vanilla can be grouped into two classes. The instances in the first class feature only exploration. See Figure 4.7a for an example. By performing exploration, the robots periodically enter the black and the white areas incrementing the value of the objective function. Instances of the first class of control software are frequent when the lowest design budget is used (14 instances out of 20), while they are rare for the other design budgets (3 and 0 instances in the case of design budget 50 000 and 200 000, respectively). The instances of the second class feature an alternation between exploration and phototaxis. See Figure 4.7b for an example. A robot uses exploration to search for the black areas. When it finds a black area, it switches to phototaxis to return to the white area. When it reaches the white area, it resumes exploration. The behaviors observed in simulation and in reality are similar.

EvoStick. The 60 instances of the control software obtained by EvoStick show qualitatively similar behaviors. Robots explore the arena following curved trajectories that are perturbed by the presence of other robots, the color of the floor and the intensity of the light. As a result of the perturbations, robots follow the walls and sometimes cross the arena passing on the black areas (the sources) and on the white area (the nest). However, this behavior is strongly affected by interference among robots: frequently, robots create aggregates that dissolve after a while. Concerning the comparison between the simulation and reality, two are the main differences: i) robots interfere less with each other in simulation than in reality; and ii) circular trajectories have a larger radius in simulation than in reality.



(a) First class: an example.



(b) Second class: an example.

Figure 4.7: Foraging—The two classes of control software designed by `Vanilla`. The initial state is indicated by a double-line circle.

4.4 Discussion

In the experiments reported in Section 4.3, `Vanilla` overcomes the reality gap better than `EvoStick`. It should be noted that the two methods have been tested under the same conditions: same simulator, environment, objective function, design budget, robot platform, and reference model. The reality gap that `Vanilla` and `EvoStick` had to overcome is therefore the same. Nonetheless, the mismatch between simulation and reality that we observed is significantly higher for the control software generated by `EvoStick` than for the one generated by `Vanilla`. Following hypothesis hp1 that we presented in Chapter 3, we ascribe this difference to the different representational power of the control architecture adopted by `Vanilla` and `EvoStick`. It is true that, besides the control architecture, `Vanilla` and `EvoStick` also differ in the optimization algorithm adopted. Nonetheless, it is our contention that the impact of the optimization algorithm is negligible in this context and that the control architecture is the main responsible for the differences observed in our experiments. To backup this contention, we performed some exploratory experiments (Francesca and Birattari, 2017) in which we compare `EvoStick` with another method, `FnnStick`, in which the control architecture is the same neural network

adopted in EvoStick and the optimization process is the one adopted in Vanilla. The results of these exploratory experiments show that FnnStick performs slightly worse than EvoStick, which excludes that the differences between the performance of Vanilla and EvoStick can be explained by a superior performance of the optimization process adopted in Vanilla. We can therefore conclude that our experiments corroborate hypothesis hp1: the high representational power provided by the fine-grained control architecture adopted in EvoStick is not properly exploited and results in solutions that do not properly generalize to the real world. On the contrary, Vanilla, with its relatively low representational power, displays better generalization capabilities. In terms of the bias-variance tradeoff (Geman et al., 1992), Vanilla has a higher bias toward a relatively restricted class of behaviors—specifically, those that can be obtained by assembling a four-state probabilistic finite state machine starting from the six given constituent behaviors and the six given conditions. As a result, Vanilla expectedly features a lower variance compared to EvoStick. Eventually, this results in a superior ability to overcome the reality gap.

4.5 Summary

In this chapter, we presented the first assessment on AutoMoDe. This assessment is performed on Vanilla, which is a proof-of-concept instance of AutoMoDe specialized for the specific reference model of the e-puck robot, RM1. We performed an experimental analysis in which we used Vanilla to design control software for two different swarm robotics tasks: aggregation and foraging. The experiments were performed using a hands-off experimental protocol, that is, no human intervention has been allowed in the automatic design process. The results show that Vanilla is able to successfully design control software for both tasks. The control software obtained by Vanilla overcomes the reality gap: the performance in simulation and in reality is comparable.

We compared Vanilla with EvoStick, a design method that implements the current best practices in evolutionary robotics. The results show that Vanilla performs better than EvoStick on both aggregation and foraging.

The empirical assessment on Vanilla continues in Chapter 5 with two main goal: i) the empirical characterization of the class of tasks for which Vanilla can successfully design control software, and ii) the comparison of Vanilla with human designers.

The contribution of this study goes beyond the mere results achieved by Vanilla. In this study, we addressed some of the challenges described in Section 2.5: We

provided a comparison of our novel design method `Vanilla` with another design method, `EvoStick`. We adopted the reference model RM1 in order to insure that the `Vanilla` and `EvoStick` access the same resources and information provided by the adopted platform. The two design methods under analysis are precisely defined and identified univocally by the names `Vanilla` and `EvoStick`. The empirical assessment of the design methods is based on robot experiments.

Chapter 5

From Vanilla to Chocolate

In this chapter, we continue our assessment of AutoMoDe with two empirical studies: Study A and Study B. In Study A, we compare Vanilla and EvoStick with two human design methods: U−Human (i.e., unconstrained human) and C−Human (i.e., constrained human). U−Human is a design method in which the human designer implements the control software in the way (s)he deems appropriate, without any kind of restriction. C−Human is a design method where the human designer implements the control software by combining the modules of AutoMoDe-Vanilla. The comparison is performed on five swarm robotics tasks that are different from those on which Vanilla and EvoStick have been tested in Chapter 4. The results show that, under the experimental conditions considered, Vanilla performs better than EvoStick but it is not able to outperform human designers. The results indicate that Vanilla's weak element is the optimization algorithm employed to search the space of candidate designs. To improve over Vanilla and with the final goal of obtaining an automatic design method that performs better than human designers, we introduce Chocolate, which differs from Vanilla only in the fact that it adopts a more powerful optimization algorithm. In Study B, we perform an assessment of Chocolate. The results show that, under the experimental conditions considered, Chocolate outperforms both Vanilla and the human designers. Chocolate is the first automatic design method for robot swarms that, at least under specific experimental conditions, is shown to outperform a human designer.

The rest of the chapter is organized as follows: In Section 5.1, we introduce the two manual design methods U−Human and C−Human. In Section 5.2, we describe Study A. In Section 5.3, we introduce AutoMoDe-Chocolate, that is empirically assessed in Study B, described in Section 5.4. Finally, in Section 5.5, we conclude with a summary of the chapter.

5.1 Two manual design methods for a swarm of e-pucks

In this section, we describe U-Human and C-Human, the two manual methods that are compared with Vanilla and EvoStick in this chapter. U-Human and C-Human, similarly to Vanilla and EvoStick, design control software for a swarm of e-puck robots conforming to the reference model RM1 described in Section 3.4.1.

5.1.1 U-Human

U-Human is a manual design method in which a human designer implements the control software in the way (s)he deems appropriate, without any kind of restriction regarding the design to produce. The designer realizes a trial-and-error process: the control software is iteratively improved and tested until the desired behavior is obtained. Within this process, the designer assesses the quality of the control software by computing the value of the objective function and by observing the resulting behavior via the simulator's visual interface. As in the case of Vanilla and EvoStick, during the development of the control software, the designer is allowed to perform tests in simulation using ARGoS, but is not allowed to perform tests with the robots. In the implementation of the control software, the designer is free to access all the resources (s)he deems appropriate including the internet and her/his own previously developed code.

The control software is implemented as a C++ class that operates on the variables defined in RM1. These variables are manipulated by the control software via an API. The designer is provided with a complete programming and simulation environment based on ARGoS. Moreover, the designer is provided with the description of the task to be solved, a control software skeleton to be used as a starting point, the task-specific objective function to be optimized, and all the scripts that initialize ARGoS for the task at hand. The control software skeleton is an empty C++ class that complies with the specification of a valid control software for ARGoS. In other terms, the skeleton is a valid control software that compiles and runs correctly but that leaves the robot motionless in its initial position. The designer is required to fill in the skeleton with the appropriate logic for solving the given task. To reduce the burden on the designer, the skeleton contains commented instructions to access the variables of RM1 via the API. The task-specific objective function computes the performance of the swarm within the simulation. It is implemented via *loop functions*, which in ARGoS parlance are callback functions executed at each step of the simulation (Pinciroli et al., 2012). The objective function is computed automatically by the simulation environment in a way that is completely transparent to the de-

signer. To ease the assessment of the control software being implemented, a utility script is provided. The script compiles the control software, starts ARGoS, generates the simulated arena, runs and visualizes the simulation, and prints the value of the objective function. The designer is allowed to use debugging tools including gdb[1] and valgrind.[2]

5.1.2 C-Human

C-Human is a manual method in which the human designer is constrained to use Vanilla's control architecture and modules. In other words, the human designer takes the role of Vanilla's optimization algorithm and searches the same design space searched by Vanilla. As in Vanilla, the human is constrained to create finite state machines comprised of at most four states, each with at most four outgoing transitions—see Section 4.1 for the details on the restrictions on the finite state machines produced by Vanilla. As in U-Human, in C-Human the designer iteratively improves the control software in a trial-and-error process that comprises implementation phases interleaved with testing via simulation. The only difference between U-Human and C-Human is that in the case of C-Human, the designer implements the control software by combining the modules of Vanilla and setting their parameters, rather than directly writing C++ source code. To allow the designer to implement the control software in this fashion, a user interface is provided. The user interface allows the designer to specify the probabilistic finite state machine using a simple finite language. The user interface also graphically visualizes the probabilistic finite state machine specified by the designer. An example of a statement in this language is given in Figure 5.1, together with the graphical visualization produced by the user interface. The user interface also starts ARGoS, generates the simulated arena, runs and visualizes the simulation, and prints the value of the objective function.

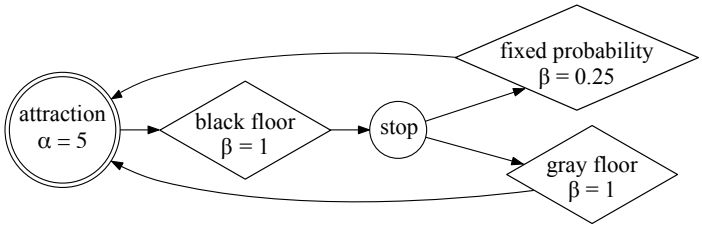
5.2 Study A: comparison of four design methods for RM1

The goal of this study is to compare Vanilla, EvoStick, U-Human, and C-Human.

[1] https://www.gnu.org/software/gdb/
[2] http://valgrind.org/

```
--nstates 2
--s0 attraction --alpha0 5 --n0 1
  --n0x0 1 --c0x0 black-floor --beta0x0 1
--s1 stop  --n1 2
  --n1x0 0 --c1x0 fixed-probability --beta1x0 0.25
  --n1x1 0 --c1x1 gray-floor --beta1x1 1
```

(a) A finite state machine described by a statement in the language adopted by C-Human. The probabilistic finite state machine comprises 2 states. State 0 is "attraction", with parameter $\alpha = 5$, and has outdegree 1: edge 0 is connected to state 1, the condition for the transition is "black-floor", with parameter $\beta = 1$. State 1 is "stop" and has outdegree 2: edge 0 is connected to state 0, the transition is activated with fixed probability 0.25; edge 1 is connected to state 0, the condition for the transition is "gray-floor", with parameter $\beta = 1$.



(b) The resulting probabilistic finite state machine.

Figure 5.1: Example of a probabilistic finite state machine specified in the simple finite language adopted in C-Human (a) and its graphical visualization (b).

5.2.1 Experimental protocol

In both studies proposed in the chapter, a central role is played by five researchers, hereinafter referred to as experts.[3] The experts are PhD candidates with about two years of experience in the domain of swarm robotics. They have previously worked with the e-puck platform or with similar platforms. They are familiar with the ARGoS simulator and programming environment.[4] Within the protocol, each expert plays a threefold role: (i) define a task, (ii) solve a task via U-Human, and (iii) solve a task via C-Human. The tasks solved by an expert via U-Human and C-Human are different from each other and from the one proposed by the expert himself. Experts are not allowed to exchange information throughout the duration of the empirical study. The roles of each expert is summarized in Table 5.1.

[3]With the goal of establishing accountability and credit, the five experts are included among the authors of Francesca et al. (2015), the paper on which the results shown in this chapter are published.

[4]We think that PhD candidates are ideal subjects for this study. Indeed, it is our understanding that a large share of the robot swarms described in the domain literature have been programmed by PhD candidates. See Francesca and Birattari (2017) for data extracted from the publication record of our research laboratory.

Table 5.1: Role of the experts, anonymously indicated here by the labels E1 to E5. For each row, the column "task" gives the name of the task; the column "defined by" identifies the expert that has defined the task; the columns "U-Human" and "C-Human" identify the experts that have solved the task acting as U-Human and C-Human, respectively. The tasks defined by the experts are described in Section 5.2.1.

task	defined by	U-Human	C-Human
SCA – shelter with constrained access	E1	E5	E4
LCN – largest covering network	E2	E1	E5
CFA – coverage with forbidden areas	E3	E2	E1
SPC – surface and perimeter coverage	E4	E3	E2
AAC – aggregation with ambient cues	E5	E4	E3

Definition of the tasks

In the definition of the tasks, the experts are kept unaware of the design methods included in the empirical study, in order to avoid any influence in the experts' choices that could favor one method over the others. Experts are asked to define tasks that, according to their judgment, could be performed by a swarm of 20 robots conforming to RM1. The experts are given a set of constraints that the tasks must satisfy: The time available to the robots for performing a task is $T = 120\,\mathrm{s}$. The robots operate in a dodecagonal area of 4.91 m^2 surrounded by walls. The floor of the arena is gray. Up to three circular or rectangular patches may be present on the floor. The patches may be either white or black. The diameter of the circular patches and the sides of the rectangular patches cannot exceed $0.6\,\mathrm{m}$. The environmental setup may include a light source placed outside the south side of the arena. Up to 5 obstacles may be present in the arena. Obstacles are wooden cuboids of size $0.05\,\mathrm{m} \times 0.05\,\mathrm{m} \times L$, where L is in the range $[0.05, 0.80]\,\mathrm{m}$.

As part of the task definition, the experts are asked to define the task-specific performance measure that will be used to assess task execution. The performance measure should be computable on the basis of the position and orientation of the robots, evaluated every $100\,\mathrm{ms}$.

The procedure through which an expert defines a task can be interpreted as a sampling according to an unknown distribution defined over the space of tasks that can be performed by a swarm of 20 robots conforming to RM1, and that satisfy the given environmental constraints. The tasks that are relevant to our study can be defined in terms of the sampling procedure: the higher the probability that a task is sampled, the higher the relevance of the task.

Description of the tasks defined by the experts

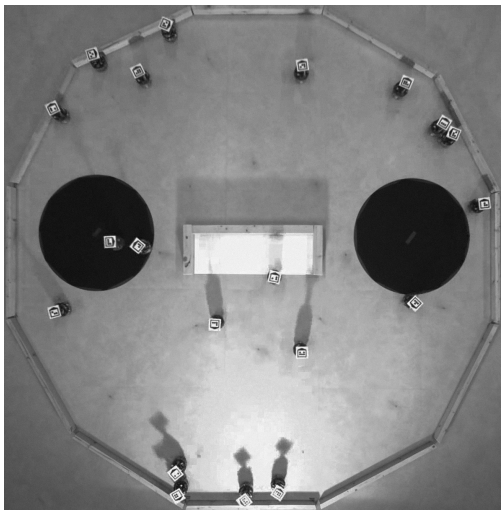
The following are the tasks defined by the experts according to the procedure given in Section 5.2.1. Overhead shots of the arenas are given in Figure 5.2. It should be noted that the performance of the proposed tasks is measured by an objective function to either maximize or minimize.

SCA – shelter with constrained access. The arena contains a rectangular white region of $0.15\,\mathrm{m} \times 0.6\,\mathrm{m}$. This region is closed on three sides by obstacles: only the south side is open for the robots to enter. In the arena, there are also two black circular patches, positioned aside the white region. The two circular patches have the same diameter of $0.6\,\mathrm{m}$. The setup also includes a light source placed on the south side of the arena. The task for the robots is to aggregate on the white region: the shelter. The robots can use the light source and the black circular patches to orientate themselves. The performance measure is defined in terms of an objective function to maximize: $F_{SCA} = \sum_{t=1}^{T} N(t)$, where $N(t)$ is the number of robots in the shelter at time t and T is the time available to the robots for performing the task.

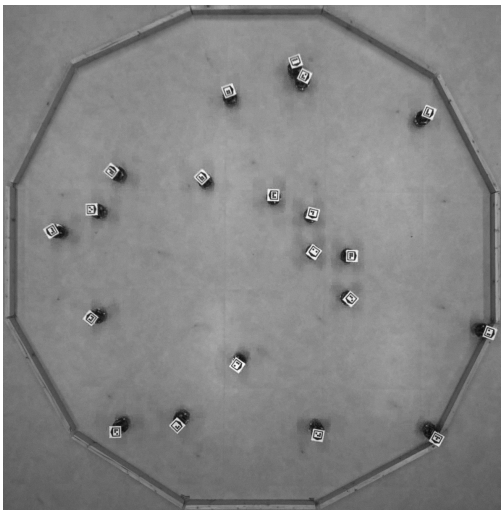
LCN – largest covering network. The arena does not contain any obstacle, floor patch or light source. The robots are required to create a connected network that covers the largest area possible. Each robot covers a circular area of $0.35\,\mathrm{m}$ radius. Two robots are considered to be connected if their distance is less than $0.25\,\mathrm{m}$. The performance measure is defined in terms of an objective function to maximize: $F_{LCN} = A_{C(T)}$, where $C(T)$ is the largest network of connected robots at the end T of the time available for performing the task and $A_{C(T)}$ is the area covered by $C(T)$.

CFA – coverage with forbidden areas. The arena contains three circular black regions, each with a diameter of $0.6\,\mathrm{m}$. The robots are required to cover the arena, avoiding the forbidden areas denoted by the black floor. The performance measure is defined in terms of an objective function to minimize: $F_{CFA} = E[d(T)]$, where $E[d(T)]$ is the expected distance, at the end T of the time available for performing the task, between a generic point of the arena and the closest robot that is not in the forbidden area. This objective function is measured in meters.

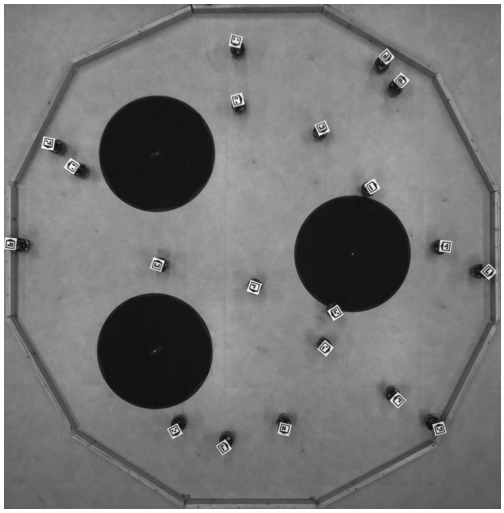
SPC – surface and perimeter coverage. The arena contains a circular black region with a diameter of $0.6\,\mathrm{m}$ and a square white region with sides of $0.6\,\mathrm{m}$. The robots are required to aggregate on the perimeter of the black circle and to cover the area of the white square. The performance measure is defined in terms of an objective function to minimize: $F_{SPC} = E[d_a(T)]/c_a + E[d_p(T)]/c_p$, where $E[d_a(T)]$ is the expected distance, at the end T of the time available for performing the task, between a generic point in the square region and the closest robot that is in the square region,



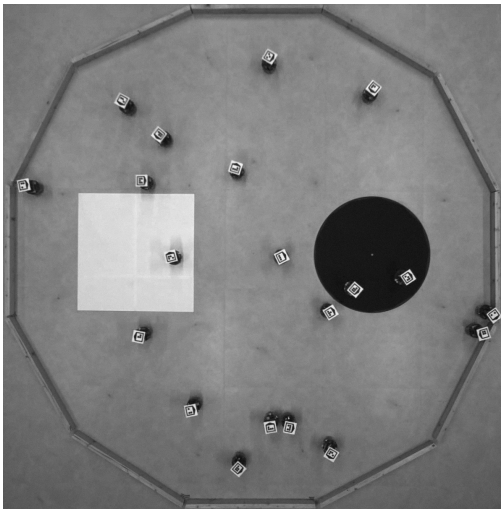
SCA – shelter with constrained access
robots must aggregate in the white region, the shelter.



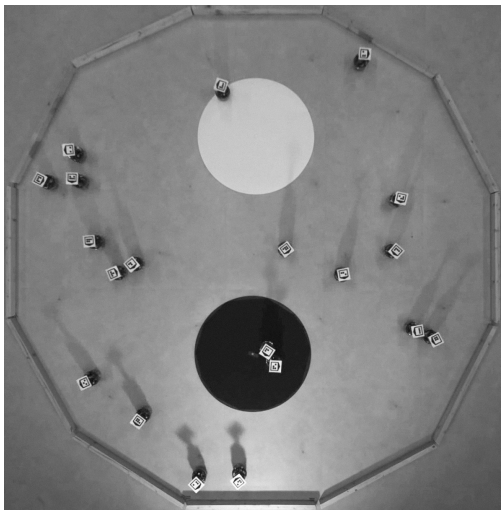
LCN – largest covering network
robots must create a connected network that covers the largest area possible.



CFA – coverage with forbidden areas
robots must cover all the arena except the forbidden black regions.



SPC – surface and perimeter coverage
robots must cover the area of the white square and the perimeter of the black circle.



AAC – aggregation with ambient cues
robots must aggregate on the black circle.

Figure 5.2: Overhead shots of the arenas used for the five tasks defined by the experts. The pictures show also the 20 e-puck robots.

$E[d_p(T)]$ is the expected distance between a generic point on the circumference of the circular region and the closest robot that intersects the circumference. $c_a = 0.08$ and $c_b = 0.06$ are scaling factors that correspond to the values of $E[d_a]$ and $E[d_p]$, respectively, under the ideal condition in which 9 robots are regularly and equally spaced on the surface of the white square and 9 on the perimeter of the black circle. See Francesca and Birattari (2017) for more details. If no robot is on the surface of the square region and/or on the perimeter of the circular region, $E[d_a(T)]$ and/or $E[d_p(T)]$ are undefined and we thus assign an arbitrarily large value to F_{SPC}. We consider this a major failure.

AAC–aggregation with ambient cues. The arena contains two circular regions, one black and one white, each with a diameter of $0.6\,\mathrm{m}$. The black region is placed closer to the light source, which is on the south side of the arena. The robots have to aggregate on the black region and can use the light and the white region to orientate themselves. The performance measure is defined in terms of an objective function to maximize: $F_{AAC} = \sum_{t=1}^{T} N(t)$, where $N(t)$ is the number of robots on the black region at time t.

Design methods under analysis and experimental setup

We compare `Vanilla`, `EvoStick`, `U-Human`, and `C-Human`. These four design methods are tested under the same conditions:

- Same platform. All methods target the same robotic platform: the specific version of the e-puck formally defined by $\mathrm{RM1}$.
- Same simulator. All methods employ ARGoS as a simulation software to evaluate design candidates.
- Same performance measures. All methods base the evaluation of a design candidate on the same task-specific performance measures.
- Same resources. To design the control software, the four methods are given a similar amount of time, with a slight advantage to human designers. `U-Human` and `C-Human` are given four hours per task. Time starts when the human designer receives the description of the task. `Vanilla` and `EvoStick` are given a budget of 200 000 executions of ARGoS per task. `Vanilla` and `EvoStick` are executed on a computer cluster that comprises 400 opteron6272 cores. Under this setting, `Vanilla` and `EvoStick` are able to complete a design session in approximately 2 hours and 20 minutes, wall-clock time.

It is important to notice that simulation plays a different role in automatic and manual design. `Vanilla` and `EvoStick` utilize simulation only to compute the value of the objective function. This value is then used by the optimization algorithm to steer

the search process. Beside the value of the objective function, no other piece of information is retained from the simulation. A graphical visualization of the simulation is not needed and therefore is not performed. When a graphical visualization is not performed, ARGoS is much faster than real time. As a consequence, the main benefit that `Vanilla` and `EvoStick` obtain from not requiring a graphical visualization is the fact that the objective function can be computed in a relatively short amount of time.

In contrast, human designers greatly benefit from observing the whole evolution of the simulation: the human designer observes the resulting behavior of the swarm and gets insights on how to improve the control software. Arguably, for a human designer visual observation is more informative than the value of the objective function. Both in `U-Human` and `C-Human`, the designer can choose to speed up the visualization with respect to real time or even to disable visualization altogether. By performing simulations with visualization (at some appropriate speed-up level), the human designer trades simulation speed for useful information.

Assessment on a swarm of e-pucks

The control software produced by `Vanilla`, `EvoStick`, `U-Human` and `C-Human` for each task is assessed via test runs with a swarm of 20 e-puck robots.

In this study we adopt a hands-off approach that reduces human intervention to a bare minimum. The control software is directly cross-compiled by the ARGoS simulator and it is uploaded onto each e-puck of the swarm without any modification. To reduce the risk that the negative effects of battery discharge and other environmental contingencies affect one method more than another, the order of the test runs is randomly generated so that runs with the control software produced by the four design methods are interleaved. The initial position of the e-pucks is generated by placing the e-pucks in known positions and letting them perform a random walk for 20 seconds. This effectively yields a randomized starting condition for each run.

To compute the task-dependent performance measure we use a tracking system (Stranieri et al., 2013) that gathers data via a ceiling-mounted camera. The tracking system logs position and orientation of each e-puck every 100 ms.

Objective of the study and statistical tools

The objective of the study is to compare the four design methods. We wish to answer two questions: (i) whether `Vanilla` performs better than `EvoStick` on the tasks proposed by the experts; and (ii) whether `Vanilla` performs better than a human

designer, represented here by the U-Human and C-Human methods.

As discussed in Section 5.2.1, the selected tasks can be seen as a sample extracted from a class of tasks. As such, these tasks allow one to draw conclusions that generalize, in a statistical sense, to the class of tasks from which they have been sampled. For this reason, we concentrate our attention on the aggregate performance of the methods over the tasks considered. For the sake of completeness, we report also a boxplot of the per-task performance and the results obtained in simulation by the control software produced by the methods under analysis. Nonetheless, the focus of our study remains the aggregate analysis.

For each task, we perform 40 independent runs: 10 for the control software generated by each of the four methods under analysis. We analyze the aggregated results using the Friedman test (Conover, 1999), with the task as a blocking factor: the total pool of results of the 200 runs $= 10$ runs $\times 4$ methods $\times 5$ tasks, are aggregated by task and ranked independently from the design method.

As the Friedman test is a rank-based non-parametric test, it does not require scaling the performance measure computed for each of the tasks nor formulating any restrictive hypothesis on the underlying distribution of the different performance measures. This test requires only to convert the objective functions of all tasks into the objective functions of the equivalent minimization problems. Given the rank-based nature of the Friedman test, this operation is trivial: it can be performed via any function that inverts the rank order. Specifically, to obtain a minimization problem from a maximization one, we use as objective function the inverse of the original one. We represent the result of the Friedman test in a graphical way: a plot that shows the expected rank obtained by each design method, together with a 95% confidence interval. If the confidence intervals of two methods do not overlap, the difference between the expected rank of the two is statistically significant.

Concerning the per-task results of the four design methods, we show five notched box-and-whisker boxplots: one for each task. A notched box-and-whisker boxplot gives a visual representation of a sample. The horizontal thick line denotes the median. The lower and upper sides of the box are called upper and lower hinges and represent the 25-th and 75-th percentile of the observations, respectively. The upper whisker extends either up to the largest observation or up to 1.5 times the difference between upper hinge and median—whichever is smaller. The lower whisker is defined analogously. Small circles represent outliers (if any), that is, observations that fall beyond the whiskers. Notches extend to $\pm 1.58\,\mathrm{IQR}/\sqrt{n}$, where IQR is the interquartile range and $n = 10$ is the number of observations. Notches indicate the 95% confidence interval on the position of the median. If the notches of two boxes

do not overlap, the observed difference between the respective medians is significant (Chambers et al., 1983).

In the boxplots, we include also the results obtained in simulation in order to appraise the impact of the reality gap on the four design methods. Results obtained with robots are represented by wide boxes and those obtained in simulation by narrow boxes. As with the assessment performed with the e-puck robots, also in simulation we perform 10 independent runs for the control software instance generated by each of the four design methods under analysis.

5.2.2 Per-task results

We report in the following the results obtained by the methods under analysis on each of the five tasks. The notched box-and-whisker boxplots are given in Figure 5.3. In the boxplots, the arrows indicate whether the objective function is to maximize or minimize. Videos of the test runs and the complete data are available as online supplementary material (Francesca and Birattari, 2017).

SCA – shelter with constrained access. `C-Human` and `Vanilla` perform better than `U-Human` and `EvoStick`. In particular, `Vanilla` performs significantly better than `EvoStick`. `EvoStick` is unable to overcome the reality gap. This holds true also for `U-Human`, but to a far minor extent. `C-Human` and `Vanilla` overcome the reality gap satisfactorily.

LCN – largest covering network. `C-Human` and `EvoStick` perform better than other methods and `Vanilla` performs significantly better than `U-Human`. `Vanilla` and `U-Human` do not successfully overcome the reality gap.

CFA – coverage with forbidden areas. The performance of the four methods is comparable: differences are within a range of few centimeters, that is, less than half of the e-puck's diameter. Regarding the reality gap, all methods display differences between simulation and reality, but these differences are small in absolute terms—they are all within few centimeters.

SPC – surface and perimeter coverage. `EvoStick` is visibly unable to overcome the reality gap and performs significantly worse than the other methods. In the boxplot, the four `X` indicate four runs that resulted in a major failure. `Vanilla`, `U-Human`, and `C-Human` perform comparably well.

AAC – aggregation with ambient cues. `Vanilla`'s results are slightly better than those of `U-Human` and `C-Human`, and significantly better than those of `EvoStick`. The four methods differ in their ability to overcome the reality gap: `EvoStick` has

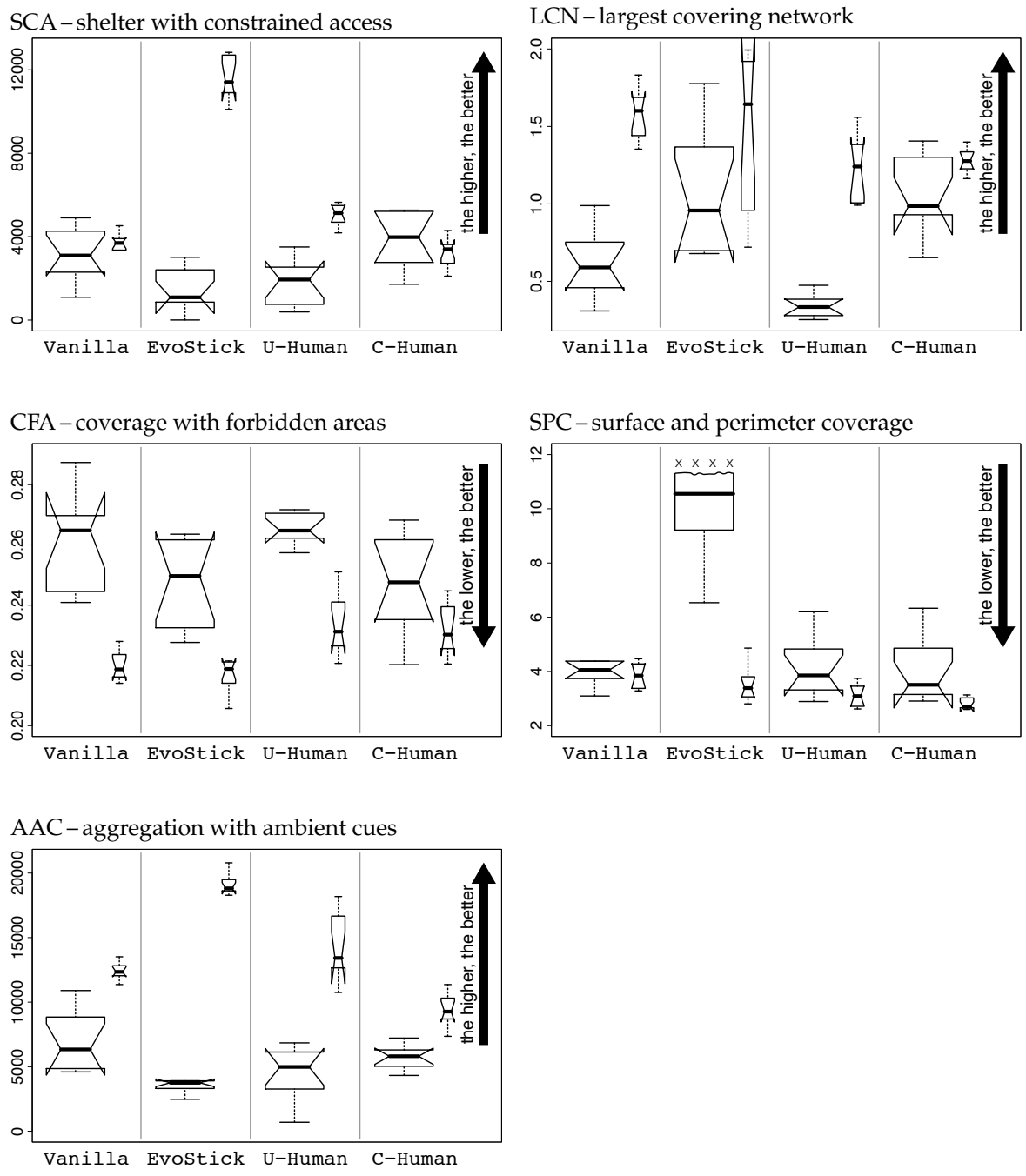


Figure 5.3: Study A—notched box-and-whisker boxplots of the results obtained by `Vanilla`, `EvoStick`, `U-Human`, and `C-Human` on the five tasks. In each boxplot, the vertical axis reports the values of the task-specific performance measure. Wide boxes represent the results obtained with the robots and narrow boxes of those obtained in simulation. See Section 5.2.1 for a guide on how to read notched box-and-whisker boxplots.

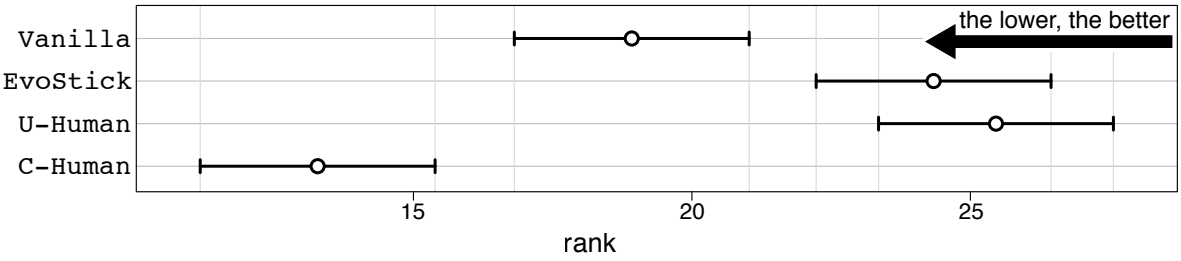


Figure 5.4: Study A—Friedman test on aggregate data from the five tasks. `Vanilla` performs significantly better than `EvoStick` and `U-Human`, but significantly worse than `C-Human`. See Section 5.2.1 for an explanation of how to read the plot.

the most severe difficulties, followed by `U-Human`; also `Vanilla` and `C-Human` display difficulties, but to a minor extent.

5.2.3 Aggregate analysis and discussion

The aggregate analysis is given in Figure 5.4: `C-Human` performs significantly better than `Vanilla`, and both `C-Human` and `Vanilla` perform significantly better than `EvoStick` and `U-Human`. The comparison between `Vanilla` and `EvoStick` confirms the results obtained in Chapter 4.

Concerning manual design, some interesting facts can be observed. The results show that, at least under the specification–development–deployment life cycle, also manual design suffers from the reality gap. It is interesting to notice that `C-Human` appears to be more effective than `U-Human` at overcoming the reality gap. This confirms the reasoning based on the notion of bias-variance tradeoff (Geman et al., 1992) that we presented in Chapter 3: the reality gap problem is an instance of a wider problem related to generalization. Methods that are more constrained and therefore have a reduced representational power also have a reduced variance. As a result, they appear to have better generalization properties.

A second interesting fact is that, under the protocol adopted, human designers produce more effective control software when constrained to use the few and relatively simple predefined modules of `Vanilla`. This result was unexpected—particularly by the experts themselves—and appears counter-intuitive. We were convinced that human designers would have obtained the best results when left free to structure the control software following their intuition and understanding. Also, we expected that the constraint to limit the design activity to assembling 12 predefined modules would have hindered their creativity and would have limited their effectiveness. The results proved us wrong: `C-Human` clearly outperforms `U-Human`. Apparently, the aforementioned reduction of the variance that is obtained by intro-

ducing constraints more than compensates for the disadvantages that derive from the introduction of a bias.

Concerning the comparison between manual and automatic design, the results show that Vanilla performs significantly better than U-Human but worse than C-Human. This is a promising result but indicates that we have not attained our goal yet: Vanilla cannot be claimed to be more effective than a human designer.

The results obtained by C-Human and Vanilla corroborate the hypothesis that Vanilla's set of modules are generally appropriate for tackling relevant tasks with robots conforming to RM1. The results also highlight a major issue with Vanilla: the optimization algorithm F-Race appears to be unable to fully exploit the potential of the modules. This is clearly indicated by the fact that C-Human, which adopts the same modules adopted by Vanilla, performs significantly better.

5.3 Chocolate

Chocolate is an improved version of Vanilla. As Vanilla, Chocolate designs control software for RM1. Chocolate differs from Vanilla in a single aspect: the optimization algorithm used to explore the design space. Chocolate adopts Iterated F-Race (Balaprakash et al., 2007; Birattari et al., 2010; López-Ibáñez et al., 2011), an algorithm for automatic configuration originally devised to fine-tune the parameters of metaheuristics. Iterated F-Race is based on F-Race (Birattari et al., 2002; Birattari, 2009), the optimization algorithm adopted in Vanilla.

In Iterated F-Race, the optimization process goes through a series of iterations each of which is an execution of the F-Race algorithm. In the first iteration, an initial set of candidate solutions is generated by sampling the space of feasible solutions in a uniformly random way. The initial candidates undergo a first execution of the F-Race algorithm. When the F-Race algorithm terminates, the surviving solutions—that is, the candidate solutions that have not been discarded—are used as a seed to generate a new set of candidate solutions on which the following iteration will operate. The new set of candidates is obtained by sampling the space of feasible solutions according to a distribution that gives a higher probability of being selected to solutions that are close to the surviving solutions. See López-Ibáñez et al. (2011) for the details. The new set of candidates undergoes a further execution of the F-Race algorithm. The process is iterated and stops when a predefined budget of evaluations have been performed.

The implementation of Iterated F-Race that is adopted in Chocolate is the one provided by the *irace* package (López-Ibáñez et al., 2011) for R (R Development Core

Team, 2008). Chocolate uses the default parameters of *irace* and samples the design space using the built-in sampling procedure of *irace* (López-Ibáñez et al., 2011).

Our working hypotheses are that, by adopting a more effective optimization algorithm, (i) Chocolate improves over Vanilla and, most importantly, (ii) the improvement is such that Chocolate outperforms C-Human.

5.4 Study B: assessment of Chocolate

The goal of this study is to empirically assess Chocolate and corroborate the working hypotheses formulated in Section 5.3.

5.4.1 Experimental protocol

The experimental protocol that we adopt to evaluate Chocolate shares its main features with the one defined in Section 5.2.1. The only differences concern (i) the way in which the tasks are defined/selected and (ii) the design methods under analysis.

Tasks and design methods under analysis

The study is performed on the five swarm robotics tasks considered in Section 5.2. We do not perform again the procedure described in Section 5.2.1 but we directly adopt the task defined by the experts for Study A and already described in Section 5.2.1

We compare Chocolate, with Vanilla, and C-Human. We exclude EvoStick and U-Human from this study because they were clearly outperformed by C-Human and Vanilla in Study A. Concerning Vanilla and C-Human, we adopt the same control software generated in Study A.

Chocolate, Vanilla, and C-Human share three key characteristics: (i) they all produce robot control software in the form of a probabilistic finite state machine; (ii) they operate on the same set of modules; and (iii) they all adopt the same simulator to compare and select candidate designs.

The three design methods under analysis are tested under the same conditions adopted in Study A: same platform, same simulator, same performance measures, same resources. See Section 5.2.1 for the details.

Assessment on a swarm of e-pucks

As in Study A, the control software produced by the design methods are assessed via test runs with a swarm of 20 e-puck robots.

Although in Section 5.2 we have already performed an assessment of the control software generated by Vanilla and C-Human on the five tasks considered, we repeat the assessment here. This allows us to compare the three methods under the exact same conditions. It would be indeed practically impossible to reproduce the same conditions under which Study A was performed: robots, batteries, and light sources have been subject to wear and their properties have possibly changed. Moreover, we have updated the firmware of the e-pucks and the software we use to manage the e-pucks and to track their position over time. Finally, the arena has been disassembled and later reassembled in a different position.

We adopt here the same hands-off approach we adopted in Study A: the control software is cross-compiled by ARGoS and it is uploaded onto each e-puck of the swarm without any modification. Moreover, the order of the test runs is randomly generated so that runs with the control software produced by the methods under analysis are interleaved. The initial positions of the e-pucks are randomly generated as in Study A. Also the performance measures are computed via the tracking system as in Study A.

Objective of the study and statistical tools

The objective of the study is to compare the three design methods. In particular, we wish to confirm our working hypotheses: (i) Chocolate improves over Vanilla; and, most of all, (ii) the improvement is such that Chocolate outperforms C-Human. As in Study A, we are interested in the aggregate performance of the methods over the five tasks. Therefore, the main statistical tool we use is the Friedman test. For completeness, we report also the per-task notched box-and-whisker boxplots.

5.4.2 Per-task results

The results obtained by the three methods under analysis on each of the five tasks are represented by the notched box-and-whisker boxplots reported in Figure 5.5. Videos of the test runs and the complete data are available as online supplementary material (Francesca and Birattari, 2017).

SCA–shelter with constrained access. The control software instance designed by Chocolate performs better than the ones designed by Vanilla and C-Human. The differences in performance between Chocolate and C-Human and between Chocolate and Vanilla are significant. Moreover, C-Human performs significantly better than Vanilla. Regarding the difference between simulation and real-

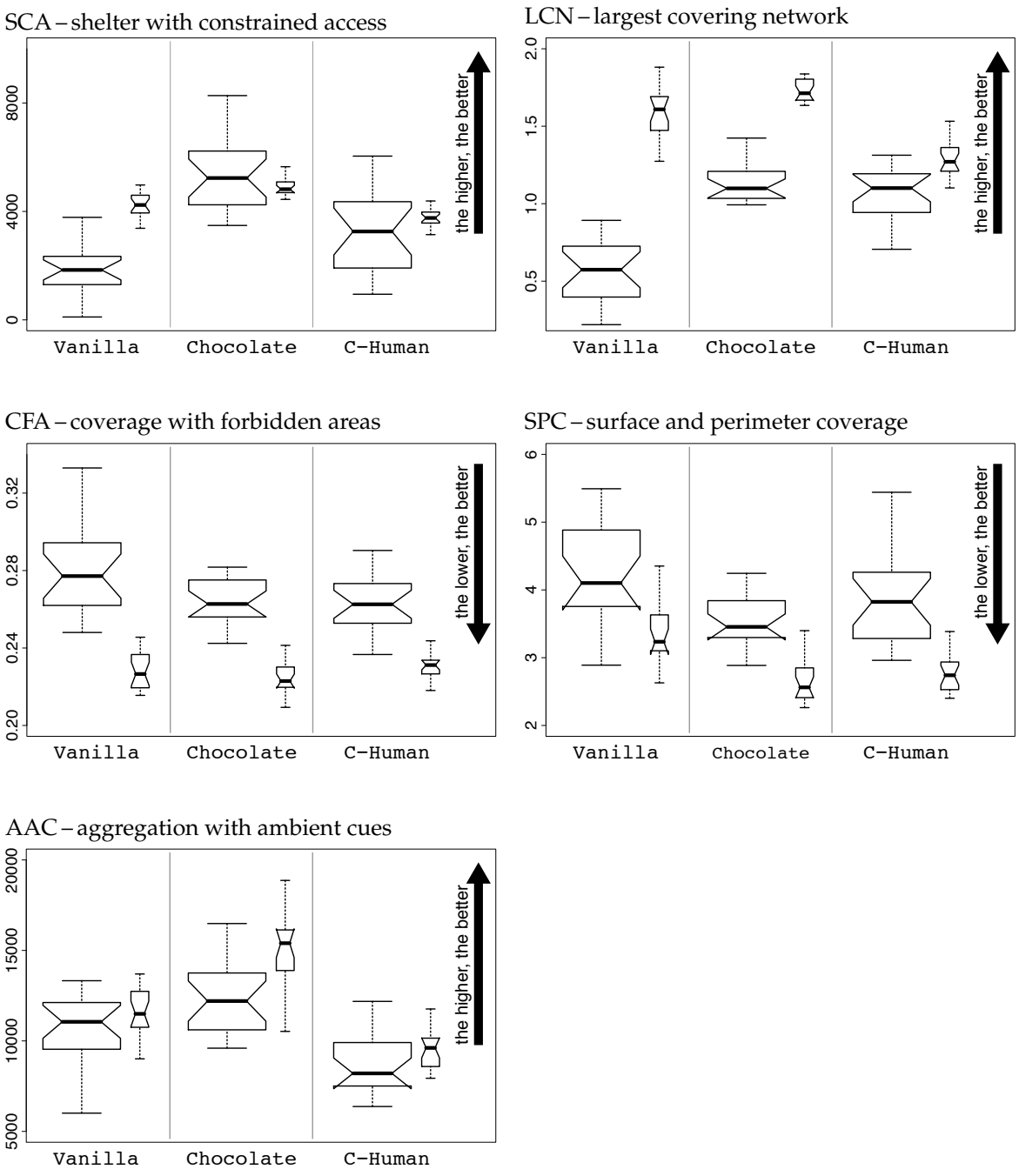


Figure 5.5: Study B—notched box-and-whisker boxplots of the results obtained by `Vanilla`, `Chocolate`, and `C-Human` on the five tasks. In each boxplot, the vertical axis reports the values of the task-specific performance measure. Wide boxes represent the results obtained with the robots and narrow boxes of those obtained in simulation. See Section 5.2.1 for a guide on how to read notched box-and-whisker boxplots.

ity, Chocolate and C-Human appear to overcome the reality gap successfully: they have similar performance in simulation and in reality. On the contrary, Vanilla shows a significant mismatch.

LCN – largest covering network. Chocolate and C-Human have qualitatively the same performance. On the other hand, Vanilla performs significantly worse than both Chocolate and C-Human. For what concerns the effects of the reality gap, all three design methods present a rather noticeable difference between simulation and reality. C-Human displays the smallest mismatch. The mismatch between the performance in simulation and reality is possibly due to the fact that, to solve this task, the e-pucks rely on their ability to measure the distance of the neighboring robots. The measurement of the distance is obtained via the range-and-bearing board, which is imprecise and highly dependent on uncontrolled factors such as battery levels and light conditions.

CFA – coverage with forbidden areas. Chocolate and C-Human have similar performance. Vanilla is slightly worse. Differences are small: medians are all within a range of less than two centimeters. Regarding the reality gap, the three methods present a similar difference between simulation and reality. The flattening of the results with small differences that have no practical implications is possibly due to the imprecision of the distances measured via the range-and-bearing board.

SPC – surface and perimeter coverage. The median performance recorded for Chocolate is better than the one recorded for C-Human and Vanilla. The difference between Chocolate and Vanilla is significant. Concerning the difference between the performance observed in simulation and on the robots, all three the methods show some mismatch. Like in the case of SCA – shelter with constrained access, this difference is possibly due to the fact that part of the task relies on the imprecise estimation of the distance provided by the range-and-bearing board.

AAC – aggregation with ambient cues. Both Chocolate and Vanilla perform significantly better than C-Human. The median recorded for Chocolate is slightly better than the one recorded for Vanilla. Concerning the difference between the performance observed in simulation and on the robots, Vanilla shows a smaller difference with respect to Chocolate and C-Human.

5.4.3 Aggregate analysis and discussion

The results of the aggregate analysis are reported in Figure 5.6. These results confirm those presented in Section 5.2: C-Human outperforms Vanilla. The better performance of C-Human over Vanilla corroborates the hypothesis formulated in Sec-

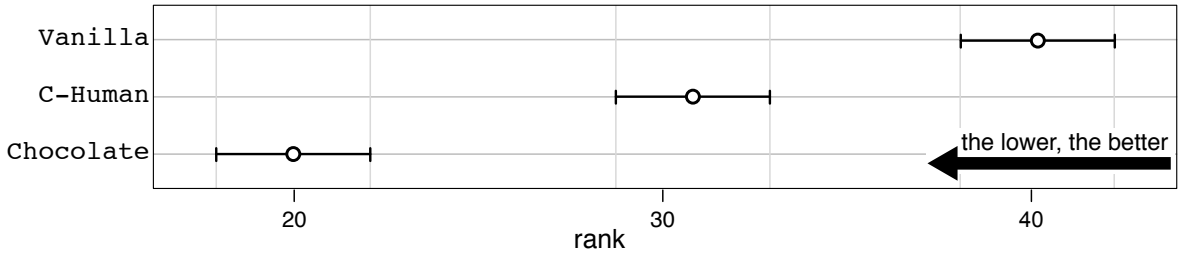


Figure 5.6: Study B—Friedman test on aggregate data from the five tasks. `Chocolate` performs significantly better than both `Vanilla` and `C-Human`. See Section 5.2.1 for an explanation of how to read the plot.

tion 5.2.3: as `C-Human` and `Vanilla` design control software combining the same modules, the failure of `Vanilla` to match `C-Human`'s performance is to be ascribed to `Vanilla`'s optimization algorithm. The hypothesis is confirmed by the fact that `Chocolate`, which adopts a more advanced optimization algorithm, outperforms `Vanilla`. This improvement of `Chocolate` is such that, under the experimental conditions considered in the study, `Chocolate` outperforms also `C-Human`. In other words, under the experimental conditions defined by the protocol presented in Section 5.4.1, `Chocolate` produces better control software than the one produced by a human designer that operates on the same set of modules as `Chocolate`.

The results presented in Section 5.2 show that `C-Human` outperforms `U-Human`, that is, the human designer that produces control software without any restriction on the structure of the control software. Together, the results presented in Section 5.2 and those presented here lead to a stronger statement: under the experimental conditions defined in Section 5.2.1 and Section 5.4.1, `Chocolate` designs control software that outperforms the one produced by a human designer, whether the human is constrained to use `Chocolate`'s (and `Vanilla`'s) modules or not.

Generality and limitations

An interesting question that naturally arises concerns the generality of the automatic design methods discussed in the chapter. The question should be addressed at two different levels: (i) the generality of `Vanilla` and `Chocolate`, as specializations of AutoMoDe to a given reference model; and (ii) the generality of AutoMoDe as an approach. In Chapter 3, we have already conceptually framed the notion of specialization of AutoMoDe and the generality of a specialization. In the following, we focus on original remarks that can be made on the basis of the results presented in this chapter. Regarding the generality of `Vanilla` and `Chocolate`, the results presented in this chapter corroborate the hypothesis that underlies the definition

of Vanilla: the modules originally proposed for Vanilla and then adopted by Chocolate and C-Human are sufficiently general to produce control software for tasks that can be accomplished by a swarm of robots conforming to RM1. The hypothesis is corroborated by the results because of the way in which the tasks have been defined: as already noted, they can be considered as sampled according to an unknown distribution defined over the space of tasks that can be accomplished by a swarm of robots conforming to RM1. Regarding AutoMoDe, statements on the general applicability of the approach can be made only by specializing AutoMoDe to a large number of different reference models (of the same or different robots) and then assessing these specializations on multiple tasks via studies similar to those presented in this chapter. This is clearly a research program that requires a large amount of empirical work and that goes far beyond the possibilities of this thesis.

A similar reasoning applies to the adoption of Iterated F-Race within AutoMoDe. As the results obtained by Iterated F-Race in Chocolate are fully satisfactory, Iterated F-Race will likely be the first optimization algorithm that we will consider in the specialization of AutoMoDe for new reference models. Nonetheless, it is perfectly possible that a new reference model (and therefore a new set of modules) requires adopting another optimization algorithm. It should be noted that the optimization algorithm to be used is not part of the definition of AutoMoDe, but rather of its specializations. Whether Iterated F-Race scales satisfactorily with the number of modules and whether some characteristics of the modules create particular problems to Iterated F-Race is an empirical question that can be addressed only by specializing AutoMoDe to a large number of reference models that require an increasingly larger set of modules. Also in this case, this research program is clearly beyond the possibility of this thesis.

A limitation of the studies presented in the chapter is related to the size and the composition of the group of experts. Five researchers, all affiliated with the same research group, cannot be considered as a sufficiently large and representative sample of the entire population of swarm robotics experts. As a consequence, the results presented in this chapter do not generalize to the entire population of swarm robotics experts. These results show only that Chocolate was able, under the specific experimental conditions considered in this chapter, to outperform a group of qualified domain experts, although these experts are not representative of the whole population. Nonetheless, this chapter presents the first empirical study in which an automatic method for the design of control software for robot swarms has been shown to outperform human designers under controlled experimental conditions.

5.5 Summary

In this chapter, we presented two empirical studies on the design of control software for robot swarms in which automatic and manual methods are compared. In Study A, we compared two automatic methods—`Vanilla` and `EvoStick`—with two manual methods—`U-Human` and `C-Human`. `Vanilla` produces control software by assembling preexisting modules into a finite state machine. `EvoStick` is a rather standard evolutionary robotics method. The two automatic methods have been already used in Chapter 4 and have been applied in this chapter without any modification whatsoever. We performed the comparison on five new tasks, different from those on which `Vanilla` and `EvoStick` had been previously tested. The tasks have been defined by human experts that, at the time of the definition of the tasks, were unaware of the functioning of `Vanilla` and `EvoStick`. The results show that `Vanilla` produces better control software than `EvoStick`, which confirms the results previously obtained on other tasks. Moreover, the results show that `Vanilla` outperforms `U-Human` but is outperformed by `C-Human`. As `C-Human` is a method in which a human manually synthesizes finite state machines by assembling the same modules on which `Vanilla` operates, we conclude that the difference in performance between `Vanilla` and `C-Human` has to be ascribed to shortcomings in `Vanilla`'s optimization algorithm.

To confirm our hypothesis, we defined a new automatic design method, `Chocolate`, that differs from `Vanilla` only in the optimization algorithm adopted. We assessed `Chocolate` in Study B: our results show that, under the specific experimental conditions considered in the study, `Chocolate` outperforms both `Vanilla` and `C-Human`. `Chocolate` is the first automatic design method for robot swarms that is shown to outperform a human designer, albeit under specific experimental conditions.

The contribution of this chapter goes beyond the mere results achieved by `Chocolate`. We make another important contribution to the literature on the automatic design methods for robot swarms by addressing the challenges described in Section 2.5: A notable trait of the empirical studies presented in the chapter is the fact that all the design methods under analysis adopt the same reference model RM1. As stated in Section 2.5, the definition of a reference model of the robotic platform at hand is a fundamental and necessary step to enable the fair and meaningful comparison of multiple design methods, whether manual or automatic. The design methods `Chocolate`, `U-Human`, and `C-Human` are precisely defined and univocally identified by their names. We proposed and demonstrated an experimental protocol to

compare automatic and manual design methods. The protocol has been developed for use with up to four methods and five experts. Nonetheless, the protocol can be straightforwardly extended to a larger number of methods and experts. This protocol includes a procedure to create benchmark tasks that ensures that the tasks are not generated to favor a particular design method. In particular, according to the protocol, the roles of defining the benchmark tasks and the automatic design methods are rigidly separated. The benchmark tasks are defined by researchers that are unaware of the design methods under analysis. Eventually, the empirical assessment of the design methods is based on robot experiments.

Chapter 6

Conclusions

This thesis gives a contribution to the automatic design of robot swarms.

A robot swarm is a highly redundant system that acts in a self-organized way without the need of any form of centralized coordination. The main challenge in swarm robotics is designing the individual behavior of the robots so that a desired collective behavior is obtained. Typically, the designer adopts a trial-and-error approach. Automatic methods are a promising alternative to the trial-and-error approach. In automatic methods, the design problem is cast into an optimization problem that can be tackled using an optimization algorithm. Evolutionary robotics, the most common automatic approach, has been successfully applied to design swarms that perform many tasks. However, despite many successes, evolutionary robotics presents some known limitations. In particular, we reckon the difficulty in overcoming the reality gap. Our intuition is that this difficulty is due to an excessive representational power of the control architecture typically adopted in evolutionary robotics, that is, neural networks. This intuition is the starting point of the thesis.

In this thesis, with the aim of conceiving an effective automatic design approach, we presented a conceptual framework in which the reality gap problem is described and analyzed from a machine learning perspective. It is our contention that the reality gap bears a strong resemblance to the generalization problem experienced in supervised machine learning. Following this parallel with machine learning, we analyze the reality gap using the bias-variance tradeoff. In machine learning, a consequence of the bias-variance tradeoff is that a correlation exists between the generalization capabilities of an approximator and its complexity (or the amount of computational effort devoted to its training). By transposing the bias-variance tradeoff in the context of the design of robot swarms, we defined two hypotheses that we used to understand the reality gap and its implications. Hypothesis hp1: past an optimal level, increasing the complexity of the control architecture (i.e., its representational

power) is counterproductive. Hypothesis hp2: past an optimal level, increasing the design effort is counterproductive. In both cases, past an optimal level of the representational power or of the design effort, which is unknown *a priori*, the performance in simulation increases while the performance in reality decreases. Both hypotheses hp1 and hp2 have been corroborated in this thesis.

We corroborated hypothesis hp1 by introducing and evaluating AutoMoDe, a novel automatic design approach that adopts a control architecture that has low representational power. We presented two specializations of AutoMoDe for the e-puck robot called `Vanilla` and `Chocolate` and we compared them with an automatic design method based on evolutionary robotics called `EvoStick`. The results show that `Vanilla` and `Chocolate` overcome the reality gap better than `EvoStick`.

We corroborated hypothesis hp2 with an experiment in which we compare the performance of swarms obtained by `EvoStick` using increasing design budgets. The results show that, past an optimal level, the performance in simulation and reality diverge.

The results presented in this thesis show the validity of the intuition that the reality gap bears a strong resemblance with the generalization problem faced in machine learning. Moreover, we showed that the reality gap problem can be tackled using the same tools that are used to tackle the generalization problem in machine learning: both the reduction of representational power and early stopping can be beneficial.

We compared the two specializations of AutoMoDe, `Vanilla` and `Chocolate` with two manual design methods and `U-Human` (i.e., unconstrained human) and `C-Human` (i.e., constrained human). The results show that while `Vanilla` outperforms only `U-Human`, `Chocolate` outperforms `U-Human` and `C-Human`: for the first time in the literature, an automatic design method, AutoMoDe-`Chocolate`, has been shown to design robot swarms that perform better than the ones designed by human designers.

Notwithstanding the results achieved in this thesis, it is our opinion that our findings are far from being conclusive and many scientific questions regarding hypotheses hp1 and hp2 remain open for further research. Regarding hypothesis hp1, the fact that AutoMoDe, which adopts a control architecture with low representational power, overcomes the reality gap better than `EvoStick` corroborates hp1. However, this does not imply that reducing the representational power is always beneficial. Testing the boundaries of hypothesis hp1 by extensively analyzing a larger pool of automatic design methods featuring different degrees of representational power constitutes an interesting future research direction. Regarding hypothesis hp2, we showed in a proof-of-concept experiment that an early stopping mechanism could

be beneficial. However, this does not imply that early stopping is always beneficial. As for hypothesis hp1, an extensive analysis involving a wide pool of automatic design methods and a wide pool of tasks is an interesting future research direction.

Regarding the empirical assessment of AutoMoDe, many questions still remain open. For instance, in the thesis we showed that `Vanilla` and `Chocolate` are able to design swarms of e-pucks that perform different tasks. Such tasks belong, *de facto*, to the class of the tasks than can be performed by a swarm of e-pucks that conform to the reference model RM1. In this regards, whether `Vanilla` and `Chocolate` are able to design swarm of e-pucks for *any* of the tasks that belong to this class is an open question that requires further investigation. Similarly, further investigation is required to assess the validity of AutoMoDe as a general automatic design approach. An empirical assessment performed on robot swarms conforming to the reference model RM1 is not sufficient to evaluate AutoMoDe as a general design approach. For this reason, further investigation is needed. This investigation could consider, for instance, more complex reference models. In our opinion, reference models that include communication abilities are an interesting and challenging testbed.

The contribution of this thesis goes beyond the results obtained by AutoMoDe. In this thesis, we provide a review of the literature with a particular focus on how the automatic design methods are assessed. This review shows that the literature on the automatic design of control software for robot swarms appears to be scattered and composed by isolated contributions: with few exceptions, no comparison between design methods are provided and new ideas and methods are not properly assessed against a well-established state of the art. It is our contention that the lack of an empirical practice hinders the progress of the domain.

We highlighted four main challenges that we think need to be addressed to establish a proper empirical practice in the domain of the automatic design of robot swarms: (i) Every study that proposes or applies an automatic design method should clearly define a reference model for the robotic platform considered. (ii) Every automatic design method should be precisely defined in all its parts and parameters, and univocally identified by a name. (iii) Libraries of standard benchmarks should be defined and adopted by the community for assessing newly proposed methods and ideas. (iv) Robot experiments should be the ultimate way to assess methods for the automatic design of control software for robot swarms and should be an essential element of any research study in the domain.

In this thesis, we attempted to address these four challenges: (i) We defined the reference model RM1 for the e-puck robot. (ii) We precisely defined the design methods adopted in the experiments and we gave them univocal names: `Vanil-`

`la`, `Chocolate`, `EvoStick`, `U-Human`, and `C-Human`. Moreover, we released an implementation of `Vanilla`, `Chocolate`, and `EvoStick`. (iii) We implemented a procedure for creating benchmark task. (iv) We performed empirical analyses in which experiments with the robots play a prominent role.

This is a first attempt to define an empirical practice suitable for the comparison of methods in the domain of the automatic design of robot swarms. We believe that the definition of an empirical practice is critical for the domain. In particular, we are convinced that a solid, well-established, and consistently applied empirical practice would allow the community to promote the best ideas proposed so far, to focus on promising directions, and to attract further researches and investments to the domain of the automatic design of control software for robot swarms.

Bibliography

Ackerman, E. (2012). Warehouse robots get smarter with ant intelligence. Available on IEEE Spectrum at `https://goo.gl/P4F4QV` (shortened URL). Accessed January, 2016.

Babaoglu, O., Canright, G., Deutsch, A., Di Caro, G. A., Ducatelle, F., Gambardella, L. M., Ganguly, N., Jelasity, M., Montemanni, R., Montresor, A., and Urnes, T. (2006). Design patterns from biology for distributed computing. *ACM Transactions on Autonomous and Adaptive Systems*, 1(1):26–66.

Bachrach, J., Beal, J., and McLurkin, J. (2010). Composable continuous-space programs for robotic swarms. *Neural Computing and Applications*, 19(6):825–847.

Balaprakash, P., Birattari, M., and Stützle, T. (2007). Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement. In Bartz-Beielstein, T., Blesa Aguilera, M. J., Blum, C., Naujoks, B., Roli, A., Rudolph, G., and Sampels, M., editors, *Hybrid Metaheuristics, HM 2007*, volume 4771 of *LNCS*, pages 108–122. Springer, Berlin, Germany.

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

Basile, F., Chiacchio, P., Coppola, J., and Gerbasio, D. (2015). Automated warehouse systems: A cyber-physical system perspective. In *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–4.

Bauer, F., Pereverzev, S., and Rosasco, L. (2007). On regularization algorithms in learning theory. *Journal of Complexity*, 23:52–72.

Bayindir, L. and Şahin, E. (2007). A review of studies in swarm robotics. *Turkish Journal of Electrical Engineering and Computer Sciences*, 15(2):115–147.

Bengler, K., Dietmayer, K., Farber, B., Maurer, M., Stiller, C., and Winner, H. (2014). Three decades of driver assistance systems: Review and future perspectives. *IEEE Intelligent Transportation Systems Magazine*, 6(4):6–22.

Beni, G. (2005). From swarm intelligence to swarm robotics. In Şahin, E. and Spears, W. M., editors, *Swarm Robotics*, volume 3342 of *LNCS*, pages 1–9. Springer, Berlin, Germany.

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

Bianco, R. and Nolfi, S. (2004). Toward open-ended evolutionary robotics: evolving elementary robotic units able to self-assemble and self-reproduce. *Connection Science*, 16(4):227–248.

Birattari, M. (2004). On the estimation of the expected performance of a metaheuristic on a class of instances. How many instances, how many runs? Technical Report TR/IRIDIA/2004-001, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium.

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

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 Dorigo, M., Birattari, M., Li, X., López-Ibáñez, M., Ohkura, K., Pinciroli, C., and Stützle, T., editors, *Swarm Intelligence, ANTS 2016*, volume 9882 of *LNCS*, pages 149–160. Springer, Berlin, Germany.

Birattari, M. and Dorigo, M. (2007). How to assess and report the performance of a stochastic algorithm on a benchmark problem: Mean or best result on a number of runs? *Optimization Letters*, 1(3):309–311.

Birattari, M., Stützle, T., Paquete, L., and Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. In Langdon, W. B. et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 11–18. Morgan Kaufmann, San Francisco, CA.

Birattari, M., Yuan, Z., Balaprakash, P., and Stützle, T. (2010). F-Race and Iterated F-Race: An overview. In Bartz-Beielstein, T., Chiarandini, M., Paquete, L., and

Preuss, M., editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 311–336. Springer, Berlin, Germany.

Bongard, J., Zykov, V., and Lipson, H. (2006). Resilient machines through continuous self-modeling. *Science*, 314(5802):1118–1121.

Bongard, J. C. (2013). Evolutionary robotics. *Communications of the ACM*, 56(8):74–83.

Brambilla, M. (2014). *Formal methods for the design and analysis of robot swarms*. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium.

Brambilla, M., Brutschy, A., Dorigo, M., and Birattari, M. (2015). Property-driven design for swarm robotics: A design method based on prescriptive modeling and model checking. *ACM Transactions on Autonomous and Adaptive Systems*, 9(4):17.1–28.

Brambilla, M., Ferrante, E., Birattari, M., and Dorigo, M. (2013). Swarm robotics: A review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41.

Brambilla, M., Pinciroli, C., Birattari, M., and Dorigo, M. (2012). Property-driven design for swarm robotics. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 139–146, Richland, SC. IFAAMAS.

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

Brettel, M., Friederichsen, N., Keller, M., and Rosenberg, M. (2014). How virtualization, decentralization and network building change the manufacturing landscape: an industry 4.0 perspective. *International Journal of Mechanical, Aerospace, Industrial, Mechatronic and Manufacturing Engineering*, 8(1):37 – 44.

Brooks, R. (1990). Elephants don't play chess. *Robotics and autonomous systems*, 6(1-2):3–15.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23.

Brooks, R. A. (1991). Intelligence without representation. *Artificial Intelligence*, 47(1–3):139–159.

Brooks, R. A. (1992). Artificial life and real robots. In Varela, F. J. and Bourgine, P., editors, *Toward a Practice of Autonomous Systems. Proceedings of the First European Conference on Artificial Life (ALIFE)*, pages 3–10. MIT Press, Cambridge, MA.

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. *Swarm Intelligence*, 9(1):1–22.

Capi, G. (2007). Multiobjective evolution of neural controllers and task complexity. *IEEE Transactions on Robotics*, 23(6):1225–1234.

Caruana, R., Lawrence, S., and Giles, L. (2001). Overfitting in neural nets: backpropagation, conjugate gradient, and early stopping. In Leen, T., Dietterich, T., and Tresp, V., editors, *Advances in Neural Information Processing Systems 13, NIPS 2000*, pages 402–408. MIT Press.

Casan, G. A., Cervera, E., Moughlbay, A. A., Alemany, J., and Martinet, P. (2015). ROS-based online robot programming for remote education and training. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 6101–6106, Piscataway NJ, USA. IEEE.

Casini, M., Garulli, A., Giannitrapani, A., and Vicino, A. (2014). A remote lab for experiments with a team of mobile robots. *Sensors*, 14(9):16486–16507.

Chambers, J. M., Cleveland, W. S., Kleiner, B., and Tukey, P. A. (1983). *Graphical Methods for Data Analysis*. Wadsworth & Brooks/Cole, Pacific Grove, CA.

Chobotix (2008). Chemical robots. Available at `http://www.chobotix.cz`.

Chouard, T. and Venema, L. (2015). Machine intelligence. *Nature*, 521(7553):435–435.

Christensen, A. L. and Dorigo, M. (2006). Evolving an integrated phototaxis and hole-avoidance behavior for a swarm-bot. In *Artificial Life (ALIFE)*, pages 248–254, Cambridge, MA. MIT Press.

Christensen, A. L., Duarte, M., Costa, V., Rodrigues, T., Gomes, J., Silva, F., and Oliveira, S. (2016). A sea of robots. AAAI Video Competition, `https://youtu.be/JBrkszUnms8`. Best Robot Video.

Christensen, A. L., Oliveira, S. M., Postolache, O., de Oliveira, M. J., Sargento, S., Santana, P., Nunes, L., Velez, F., Sebastiao, P., Costa, V., Duarte, M., Gomes, J., Rodrigues, T., and Silva, F. (2015). Design of communication and control for swarms of aquatic surface drones. In *Proceedings of the International Conference on Agents and Artificial Intelligence (ICAART)*, pages 548–555. SCITEPRESS, Setúbal, Portugal.

Clark, R. J., Arkin, R. C., and Ram, A. (1992). Learning momentum: online performance enhancement for reactive systems. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 111–116, Piscataway, NJ. IEEE.

Conover, W. J. (1999). *Practical Nonparametric Statistics*. Wiley, New York.

Şahin, E. (2005). Swarm robotics: From sources of inspiration to domains of application. In *Proceedings of the 2004 International Conference on Swarm Robotics*, volume 3342 of *LNCS*, pages 10–20, Berlin, Germany. Springer.

Cybenko, G. (1989). Approximations by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314.

Di Mario, E. and Martinoli, A. (2014). Distributed particle swarm optimization for limited-time adaptation with real robots. *Robotica*, 32(02):193–208.

Di Mario, E. L., Navarro Oiza, I., and Martinoli, A. (2015). A distributed noise-resistant particle swarm optimization algorithm for high-dimensional multi-robot learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5970 – 5976, Piscataway, NJ. IEEE.

Doncieux, S. and Mouret, J.-B. (2014). Beyond black-box optimization: a review of selective pressures for evolutionary robotics. *Evolutionary Intelligence*, 7(2):71–93.

Dorigo, M., Birattari, M., and Brambilla, M. (2014). Swarm robotics. *Scholarpedia*, 9(1):1463.

Duarte, M., Costa, V., Gomes, J. C., Rodrigues, T., Silva, F., Oliveira, S. M., and Christensen, A. L. (2016). Evolution of collective behaviors for a real swarm of aquatic surface robots. *PLOS ONE*, 11(3):1–25.

Duarte, M., Oliveira, S. M., and Christensen, A. L. (2014a). Evolution of hierarchical controllers for multirobot systems. In *Artificial Life (ALIFE)*, pages 657–664, Cambridge, MA. MIT Press.

Duarte, M., Oliveira, S. M., and Christensen, A. L. (2014b). Hybrid control for large swarms of aquatic drones. In *Artificial Life (ALIFE)*, pages 785–792, Cambridge MA, USA. MIT Press.

Dudek, G., Jenkin, M. R. M., Milios, E., and Wilkes, D. (1996). A taxonomy for multi-agent robotics. *Autonomous Robots*, 3(4):375–397.

Elfwing, S., Uchibe, E., Doya, K., and Christensen, H. I. (2011). Darwinian embodied evolution of the learning ability for survival. *Adaptive Behavior*, 19(2):101–120.

Ferrante, E., Guzmán, E. D., Turgut, A. E., and Wenseleers, T. (2013). GESwarm: Grammatical evolution for the automatic synthesis of collective behaviors in swarm robotics. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 17–24. ACM, New York, NJ.

Ferrante, E., Turgut, A., Duéñez-Guzmán, E., Dorigo, M., and Wenseleers, T. (2015). Evolution of self-organized task specialization in robot swarms. *PLOS Computational Biology*, 11(8):e1004273.

Floreano, D., Husbands, P., and Nolfi, S. (2008). Evolutionary robotics. In Siciliano, B. and Khatib, O., editors, *Springer Handbook of Robotics*, pages 1423–1451. Springer, Berlin, Germany.

Floreano, D. and Keller, L. (2010). Evolution of adaptive behaviour in robots by means of Darwinian selection. *PLOS Biology*, 8(1):e1000292.

Ford, M. (2015). *Rise of the Robots: Technology and the Threat of a Jobless Future*. Basic Books.

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

Francesca, G. and Birattari, M. (2017). A modular approach to the automatic design of control software for robot swarms from a novel perspective on the reality gap to AutoMoDe. Supplementary material. Available at `http://iridia.ulb.ac.be/supp/IridiaSupp2017-002/`.

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. *Swarm Intelligence*, 9(2–3):125–152.

Francesca, G., Brambilla, M., Brutschy, A., Garattoni, L., Miletitch, R., Podevijn, G., Reina, A., Soleymani, T., Salvaro, M., Pinciroli, C., Trianni, V., and Birattari, M. (2014a). An experiment in automatic design of robot swarms. In Dorigo, M., Birattari, M., Garnier, S., Hamann, H., Montes de Oca, M., Solnon, C., and Stützle, T., editors, *Swarm Intelligence, ANTS 2014*, volume 8667 of *LNCS*, pages 25–37. Springer, Berlin, Germany.

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. *Swarm Intelligence*, 8(2):89–112.

Francesca, G., Brambilla, M., Trianni, V., Dorigo, M., and Birattari, M. (2012). Analysing an evolved robotic behaviour using a biological model of collegial decision making. In *From Animals to Animats 12*, volume 7426 of *LNCS*, pages 381–390, Berlin, Germany. Springer.

Garattoni, L. and Birattari, M. (2016). Swarm robotics. *Wiley Encyclopedia of Electrical and Electronics Engineering*, pages 1–19.

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

Gardelli, L., Viroli, M., and Omicini, A. (2007). Design patterns for self-organising systems. In Burkhard, H.-D., Lindemann, G., Verbrugge, R., and Varga, L. Z., editors, *Multi-Agent Systems and Applications V*, volume 4696 of *LNCS*, pages 123–132. Springer, Berlin, Germany.

Gauci, M., Chen, J., Li, W., Dodd, T. J., and Groß, R. (2014a). Clustering objects with robots that do not compute. In Lomuscio, A. et al., editors, *Autonomous Agents and Multiagent Systems (AAMAS)*, pages 421–428, Richland, SC. IFAAMAS.

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

Gazi, V. and Fidan, B. (2007). Coordination and control of multi-agent dynamic systems: models and approaches. In Şahin, E., Spears, W. M., and Winfield, A. F. T., editors, *Swarm Robotics*, volume 4433 of *LNCS*, pages 71–102. Springer, Berlin, Germany.

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

Gerkey, B. P. and Matarić, M. J. (2004). A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954.

Gomes, J., Urbano, P., and Christensen, A. L. (2013). Evolution of swarm robotics systems with novelty search. *Swarm Intelligence*, 7(2-3):115–144.

Gusikhin, O., Filev, D., and Rychtyckyj, N. (2008). *Intelligent Vehicle Systems:Applications and New Trends*, pages 3–14. Springer, Berlin, Germany.

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)*, pages 3111–3116, Piscataway, NJ. IEEE Press.

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

Hamann, H. and Wörn, H. (2008). A framework of space–time continuous models for algorithm design in swarm robotics. *Swarm Intelligence*, 2(2):209–239.

Harvest (2008). Harvest automation, inc. Available at `https://www.harvestai.com`. Accessed January, 2016.

Harvey, I., Husbands, P., Cliff, D., Thompson, A., and Jakobi, N. (1997). Evolutionary robotics: The Sussex approach. *Robotics and Autonomous Systems*, 20(2):205–224.

Hauert, S., Berman, S., Nagpal, R., and Bhatia, S. N. (2013). A computational framework for identifying design guidelines to increase the penetration of targeted nanoparticles into tumors. *Nano Today*, 8(6):566–576.

Hauert, S. and Bhatia, S. N. (2014). Mechanisms of cooperation in cancer nanomedicine: towards systems nanotechnology. *Trends Biotechnol.*, 32(9):448–455. Special Issue: Next Generation Therapeutics.

Hauert, S., Zufferey, J.-C., and Floreano, D. (2008). Evolved swarming without positioning information: An application in aerial communication relay. *Autonomous Robots*, 26(1):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 Dorigo, M., Birattari, M., Blum, C., Christensen, A. L., Engelbrecht, A. P., Groß, R., and Stützle, T., editors, *Swarm Intelligence, ANTS 2012*, volume 7461 of *LNCS*, pages 252–259, Berlin, Germany. Springer.

Hsieh, M., Loizou, S., and Kumar, V. (2007). Stabilization of multiple robots on stable orbits via local sensing. In *International Conference on Robotics and Automation (ICRA)*, pages 2312–2317, Piscataway, NJ. IEEE Press.

Iocchi, L., Nardi, D., and Salerno, M. (2001). Reactivity and deliberation: a survey on multi-robot systems. In *Balancing Reactivity and Social Deliberation in Multi-agent Systems*, volume 2103 of *LNCS*, pages 9–32. Springer, Berlin, Germany.

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

Jakobi, N., Husbands, P., and Harvey, I. (1995). Noise and the reality gap: The use of simulation in evolutionary robotics. In *Advances in Artificial Life (ECAL)*, volume 929 of *LNCS*, pages 704–720, Berlin, Germany. Springer.

Kazadi, S., Lee, J. R., and Lee, J. (2007). Artificial physics, swarm engineering, and the hamiltonian method. In *World Congress on Engineering and Computer Science*, pages 623–632, Hong Kong. Newswood.

Kazadi, S., Lee, J. R., and Lee, J. (2009). Model independence in swarm robotics. *International Journal of Intelligent Computing and Cybernetics*, 2(4):672–694.

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

Koos, S., Cully, A., and Mouret, J.-B. (2013a). Fast damage recovery in robotics with the t-resilience algorithm. *The International Journal of Robotics Research*, 32(14):1700–1723.

Koos, S., Mouret, J.-B., and Doncieux, S. (2013b). The transferability approach: Crossing the reality gap in evolutionary robotics. *IEEE Transactions on Evolutionary Computation*, 17(1):122–145.

Kuhn, T. (1962). *The Structure of Scientific Revolutions*. University of Chicago Press, Chicago IL, USA.

Kulich, M., Chudoba, J., Kosnar, K., Krajnik, T., Faigl, J., and Preucil, L. (2013). Syrotek–distance teaching of mobile robotics. *IEEE Transactions on Education*, 56(1):18–23.

Lee, J. and Arkin, R. C. (2003). Adaptive multi-robot behavior via learning momentum. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2029–2036, Piscataway, NJ. IEEE Press.

Lee, J., Kao, H.-A., and Yang, S. (2014). Service innovation and smart analytics for industry 4.0 and big data environment. *Procedia CIRP*, 16:3 – 8.

Lehman, J., Risi, S., D'Ambrosio, D., and O Stanley, K. (2013). Encouraging reactivity to create robust machines. *Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems*, 21(6):484–500.

Lehman, J. and Stanley, K. O. (2011). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223.

Lerman, K. and Galstyan, A. (2002). Mathematical model of foraging in a group of robots: effect of interference. *Autonomous Robots*, 13(2):127–141.

Lerman, K., Galstyan, A., Martinoli, A., and Ijspeert, A. J. (2001). A macroscopic analytical model of collaboration in distributed robotic systems. *Artificial Life*, 7(4):375–393.

Letzing, J. (2012). Amazon adds that robotic touch. *The Wall Street Journal*.

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

Liu, W., Winfield, A., Sa, J., Chen, J., and Dou, L. (2007). Strategies for energy optimisation in a swarm of foraging robots. In Şahin, E., Spears, W. M., and Winfield, A. F. T., editors, *Swarm Robotics*, volume 4433 of *LNCS*, pages 14–26. Springer, Berlin, Germany.

Liu, W. and Winfield, A. F. (2010). Modeling and optimization of adaptive foraging in swarm robotic systems. *The International Journal of Robotics Research*, 29(14):1743–1760.

Lopes, Y. K., Leal, A., Dodd, T. J., and Groß, R. (2014). Application of supervisory control theory to swarms of e-puck and kilobot robots. In Dorigo, M., Birattari, M., Garnier, S., Hamann, H., Montes de Oca, M., Solnon, C., and Stützle, T., editors, *Swarm Intelligence, ANTS 2014*, volume 8667 of *LNCS*, pages 62–73. Springer, Berlin, Germany.

Lopes, Y. K., Trenkwalder, S. M., Leal, A. B., Dodd, T. J., and Groß, R. (2016). Supervisory control theory applied to swarm robotics. *Swarm Intelligence*, 10(1):65–97.

López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., and Birattari, M. (2011). The irace package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium.

Lubin, G. (2011). Presenting the robot farmers of the future. Available at `http://read.bi/s8Q0SA`. Accessed January, 2016.

MacKay, W. P. (1981). A comparison of the nest phenologies of three species of pogonomyrmex harvester ants (hymenoptera: Formicidae). *Psyche*, 88(1-2):25–74.

Martinoli, A., Easton, K., and Agassounon, W. (2004). Modeling swarm robotic systems: a case study in collaborative distributed manipulation. *The International Journal of Robotics Research*, 23(4–5):415–436.

Martinoli, A., Ijspeert, A. J., and Mondada, F. (1999). Understanding collective aggregation mechanisms: From probabilistic modelling to experiments with real robots. *Robotics and Autonomous Systems*, 29(1):51–63.

Matarić, M. J. (2001). Learning in Behavior-Based Multi-Robot Systems: Policies, Models, and Other Agents. *Cognitive Systems Research*, 2(1):81–93.

Matarić, M. J. and Cliff, D. (1996). Challenges in evolving controllers for physical robots. *Robotics and Autonomous Systems*, 19(1):67–83.

McCulloch, W. and Pitts, W. (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(4):115–133.

Miglino, O., Lund, H. H., and Nolfi, S. (1995). Evolving mobile robots in simulated and real environments. *Artificial Life*, 2(4):417–434.

Minsky, M. and Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA.

Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J.-C., Floreano, D., and Martinoli, A. (2009a). E-puck website. Last checked on November 2013.

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

Mondada, F., Franzi, E., and Guignard, A. (1999). The Development of Khepera. In *Experiments with the Mini-Robot Khepera, Proceedings of the First International Khepera Workshop*, HNI-Verlagsschriftenreihe, Heinz Nixdorf Institut, pages 7–14.

Mondada, F., Franzi, E., and Ienne, P. (1993). Mobile robot miniaturization: A tool for investigation in control algorithms. In Yoshikawa, T. and Miyazaki, F., editors, *Experimental Robotics III*, pages 501–513, Berlin, Germany. Springer.

Morgan, N. and Bourlard, H. (1990). Generalization and parameter estimation in feedforward nets: Some experiments. In Touretzky, D., editor, *Advances in Neural Information Processing Systems 2, NIPS 1990*, pages 630–637. Morgan Kaufman, San Mateo, CA.

Nilsson, N. J. (1984). Shakey the robot. Technical Report 323, AI Center, SRI International, Menlo Park, CA, USA.

Nolfi, S. and Floreano, D. (2000). *Evolutionary Robotics*. MIT Press, Cambridge, MA.

Nouyan, S., Groß, R., Bonani, M., Mondada, F., and Dorigo, M. (2009). Teamwork in self-organized robot colonies. *IEEE Transactions on Evolutionary Computation*, 13(4):695–711.

Parker, L. E. (1996a). On the design of behavior-based multi-robot teams. *Journal of Advanced Robotics*, 10:547–578.

Parker, L. E. (1996b). Task-oriented multi-robot learning in behavior-based systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1478–1487, Piscataway, NJ. IEEE.

Parker, L. E. (1997). L-ALLIANCE: task-oriented multi-robot learning in behavior-based systems. *Advanced Robotics*, 11(4):305–322.

Parker, L. E. (2000). Current state of the art in distributed autonomous mobile robotics. In Parker, L. E., Bekey, G., and Barhen, J., editors, *Distributed Autonomous Robotic Systems 4*, pages 3–12. Springer, Tokyo, Japan.

Pinciroli, C., Lee-Brown, A., and Beltrame, G. (2015). Buzz: an extensible programming language for self-organizing heterogeneous robot swarms. Available online at http://arxiv.org/abs/1507.05946.

Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G., Ducatelle, F., Birattari, M., Gambardella, L. M., and Dorigo, M. (2012). ARGoS: A modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, 6(4):271–295.

Prechelt, L. (1998). Early stopping – but when? In Montavon, G., B. Orr, G., and Müller, K.-R., editors, *Neural Networks: Tricks of the Trade*, volume 1524 of *LNCS*, pages 55–69. Springer, Berlin, Germany.

Pugh, J. and Martinoli, A. (2009). Distributed scalable multi-robot learning using particle swarm optimization. *Swarm Intelligence*, 3(3):203–222.

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

R Development Core Team (2008). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria.

Rabin, M. O. (1963). Probabilistic automata. *Inform. Control*, 6(3):230–245.

Radziwon, A., Bilberg, A., Bogers, M., and Madsen, E. S. (2014). The smart factory: exploring adaptive and flexible manufacturing solutions. *Procedia Engineering*, 69:1184–1190.

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

Reina, A. (2016). *Engineering swarm systems: A design pattern for the best-of-n decision problem*. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium.

Reina, A., Dorigo, M., and Trianni, V. (2014). Towards a cognitive design pattern for collective decision-making. In Dorigo, M., Birattari, M., Garnier, S., Hamann, H., Montes de Oca, M., Solnon, C., and Stützle, T., editors, *Swarm Intelligence, ANTS 2014*, volume 8667 of *LNCS*, pages 194–205. Springer, Berlin, Germany.

Reina, A., Miletitch, R., Dorigo, M., and Trianni, V. (2015a). A quantitative micro–macro link for collective decisions: the shortest path discovery/selection example. *Swarm Intelligence*, 9(2-3):75–102.

Reina, A., Salvaro, M., Francesca, G., Garattoni, L., Pinciroli, C., Dorigo, M., and Birattari, M. (2015b). Augmented reality for robots: Virtual sensing technology applied to a swarm of e-pucks. In *NASA/ESA Conference on Adaptive Hardware and Systems*, pages 1–6, Los Alamitos, CA. IEEE Computer Society Press.

Reina, A., Valentini, G., Fernández-Oto, C., Dorigo, M., and Trianni, V. (2015c). A design pattern for decentralised decision making. *PLOS ONE*, 10(10):e0140950.

RHEA (2010). Rhea project. Available at `http://www.rhea-project.eu`. Accessed January, 2016.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408.

Rubenstein, M., Ahler, C., Hoff, N., Cabrera, A., and Nagpal, R. (2014a). Kilobot: A low cost robot with scalable operations designed for collective behaviors. *Robotics and Autonomous Systems*, 62(7):966–975.

Rubenstein, M., Cornejo, A., and Nagpal, R. (2014b). Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799.

Sartoretti, G., Hongler, M.-O., de Oliveira, M. E., and Mondada, F. (2014). Decentralized self-selection of swarm trajectories: from dynamical systems theory to robotic implementation. *Swarm Intelligence*, 8(4):329–351.

Sarvašová, N., Ulbrich, P., Tokárová, V., Zadražil, A., and Štěpánek, F. (2015). Artificial swarming: Towards radiofrequency control of reversible micro-particle aggregation and deposition. *Powder Technology*, 278:17 – 25.

Schmidhuber, J. (2015). Deep learning in neural networks: an overview. *Neural Networks*, 61:85–117.

Silva, F., Duarte, M., Correia, L., Oliveira, S. M., and Christensen, A. L. (2015a). Open issues in evolutionary robotics. *Evolutionary Computation*, 24(2):205–236.

Silva, F., Urbano, P., Correia, L., and Christensen, A. L. (2015b). odNEAT: An algorithm for decentralised online evolution of robotic controllers. *Evolutionary Computation*, 23(3):421–449.

Son, D. W., Chang, Y. S., Kim, N. U., and Kim, W. R. (2016). Design of warehouse contol system for automated warehouse environment. In *5th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI)*, pages 980–984.

Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127.

Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2):111–147.

Stranieri, A., Turgut, A., Salvaro, M., Garattoni, L., Francesca, G., Reina, A., Dorigo, M., and Birattari, M. (2013). IRIDIA's arena tracking system. Technical Report TR/IRIDIA/2013-013, IRIDIA, Université Libre de Bruxelles, Belgium.

Sun, Z., Bebis, G., and Miller, R. (2006). On-road vehicle detection: a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(5):694–711.

Teo, J. and Abbass, H. (2004). Automatic generation of controllers for embodied legged organisms: A pareto evolutionary multi-objective approach. *Evolutionary Computation*, 12(3):355–394.

Teo, J. and Abbass, H. (2005). Multiobjectivity and complexity in embodied cognition. *IEEE Transactions on Evolutionary Computation*, 9(4):337–360.

Trianni, V. (2008). *Evolutionary swarm robotics*. Springer, Berlin, Germany.

Trianni, V. (2014). Evolutionary robotics: Model or design? *Frontiers in Robotics and AI*, 1(13):1–6.

Trianni, V. and López-Ibáñez, M. (2015). Advantages of task-specific multi-objective optimisation in evolutionary robotics. *PLOS ONE*, 10(8):e0136406.

Trianni, V. and Nolfi, S. (2009). Self-organising sync in a robotic swarm. A dynamical system view. *IEEE Transactions on Evolutionary Computation*, 13(4):722–741.

Trianni, V. and Nolfi, S. (2011). Engineering the evolution of self-organizing behaviors in swarm robotics: A case study. *Artificial Life*, 17(3):183–202.

University of Copenhagen (2011). Aseta project. Available at `http://plen.ku.dk/english/research/crop_sciences/plant_protection/aseta/`. Accessed January, 2016.

Usui, Y. and Arita, T. (2003). Situated and embodied evolution in collective evolutionary robotics. In *International Symposium on Artificial Life and Robotics*, pages 212–215, Berlin, Germany. Springer.

Valentini, G. (2016). *The best-of-n problem in robot swarms*. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium.

Valentini, G., Brambilla, D., Hamann, H., and Dorigo, M. (2016a). Collective perception of environmental features in a robot swarm. In *Swarm Intelligence, ANTS 2016*, LNCS, pages 65–76, Berlin, Germany. Springer.

Valentini, G., Ferrante, E., Hamann, H., and Dorigo, M. (2016b). Collective decision with 100 Kilobots: Speed versus accuracy in binary discrimination problems. *Autonomous Agents and Multi-Agent Systems (AAMAS)*, 30(3):553–580.

Valentini, G., Hamann, H., and Dorigo, M. (2014). Self-organized collective decision making: The weighted voter model. In Lomuscio, A., Scerri, P., Bazzan, A., and Huhns, M., editors, *Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 45–52. IFAAMAS.

Waibel, M., Keller, L., and Floreano, D. (2009). Genetic team composition and level of selection in the evolution of cooperation. *IEEE Transactions on Evolutionary Computation*, 13(3):648–660.

Watson, R., Ficici, S., and Pollack, J. (1999). Embodied evolution: Embodying an evolutionary algorithm in a population of robots. In *IEEE Congress on Evolutionary Computation (CEC)*, volume 1, pages 335 – 342, Piscataway, NJ. IEEE.

Watson, R., Ficici, S., and Pollack, J. (2002). Embodied evolution: Distributing an evolutionary algorithm in a population of robots. *Robotics and Autonomous Systems*, 39(1):1–18.

Werbos, P. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, Cambridge, MA.

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

Wischmann, S., Stamm, K., and Wörgötter, F. (2007). Embodied evolution and learning: The neglected timing of maturation. In Almeida e Costa, F., Rocha, L. M., Costa, E., Harvey, I., and Coutinho, A., editors, *Advances in artificial life (ECAL)*, volume 4648 of *LNCS*, pages 284–293, Berlin, Germany. Springer.

Wolpert, D. (1997). On bias plus variance. *Neural Computation*, 9(6):1211–1243.

Yang, G.-Z. and McNutt, M. (2016). Robotics takes off. *Science*, 352(6291):1255–1255.

Zeiger, F., Schmidt, M., and Schilling, K. (2009). Remote experiments with mobile-robot hardware via internet at limited link capacity. *IEEE Transactions on Industrial Electronics*, 56(12):4798–4805.