Université Libre de Bruxelles
Faculté des Sciences Appliquées
CODE - Computers and Decision Engineering
IRIDIA - Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

Automatic Design of Controllers in Swarm Robotics

Gianpiero FRANCESCA

Promoteur:

Prof. Mauro BIRATTARI

Co-promoteur:

Prof. Marco DORIGO

Rapport d'avancement de recherche
Année Académique 2012/2013

Acknowledgments

The research leading to the results presented in this report has received funding from the Meta-X project from the Scientific Research Directorate of the French Community of Belgium.

Statement

This work describes an original research carried out by the author. It has not been previously submitted to any other university for the award of any degree. Nevertheless, some chapters of this thesis are partially based on papers, together with other co-authors, which have been published, submitted or prepared for publication in the scientific literature.

Chapter 3 is based on the paper:

Gianpiero Francesca, Manuele Brambilla, Vito Trianni, Marco Dorigo, and Mauro Birattari. Analysing an Evolved Robotic Behaviour Using a Biological Model of Collegial Decision Making. In *From Animals to Animats 12*, 2012.

Chapters 4 and 5 are based on the paper:

Gianpiero Francesca, Manuele Brambilla, Arne Brutschy, Vito Trianni, and Mauro Birattari. AutoMoDe, a Novel Approach to the Design of Controllers for Swarm Robotic Systems. in preparation, to be submitted for journal publication.

Contents

Acknowledgments **i**

Statement **ii**

Contents **iii**

List of Figures **v**

1 Introduction **1**

2 Our Research Project **4**
2.1 State of the art 5
2.1.1 Behavior-based Approach 5
2.1.2 Evolutionary Robotics 5
2.2 Our original idea 7
2.3 Vision 10

3 Preliminary studies in evolutionary robotics **12**
3.1 Introduction 12
3.2 Experimental Setup 14
3.3 Results 16
3.3.1 Model 16
3.3.2 Dynamics of robotics and model simulations 19
3.4 Final Remarks 21

4 AutoMoDe **23**
4.1 Automatic design methods 23
4.2 Characteristics of AutoMoDe 25
4.3 Proof of concept: AutoMoDe-`Vanilla` 26

5 Experiments and Results **30**
5.1 Experimental Setup 30
5.1.1 Aggregation 30
5.1.2 Foraging 31
5.1.3 Setup of evolutionary robotics 32
5.2 Results 32
5.2.1 Aggregation 33
5.2.2 Foraging 39
5.3 Summary of the results 41

6 Conclusions and Future Works **42**

List of Figures

2.1 The design process: 1. The behavior library is a collection of behavioral modules (Each behavioral module has a set of parameters in order to be applicable in different scenarios). 2. The designer derives an objective function from the requirements. 3. The automatic configuration evaluates automatically many candidate controllers using the objective function. 4. The automatic configuration obtains a controller that meets the requirements. 5. The obtained controller is installed on the robots. 8

2.2 An example of automatic configuration using a local search: 1. A candidate controller is generated by combining the behavioral modules of the behavior library (The search in the space of all the possible controllers is done using a local search.) 2. The controller is evaluated in simulation using the objective function: The controller is installed on all the virtual robots of the swarm and the simulation is run. The objective function evaluates the performance that the swarm obtains in simulation. 3. If a termination criterion is fulfilled the current controller is returned. Otherwise the algorithm goes back in 1 9

3.1 Left: the simulated experimental arena used for the experiments. Right: the e-puck robot, used for the simulated evolutionary experiments presented in this work. 14

3.2 The bifurcation diagram of our model for different values of $d = \frac{S}{N}$. The percentage of robots in area a and area c. 18

3.3 Comparison between the behaviour of the simulated experiments and the Monte Carlo experiments keeping the number of robots fixed to $N = 10$, and varying the size of the black areas from $r_i = 15\,\text{cm}$ to $r_i = 50\,\text{cm}$. 20

3.4 Comparison between the behaviour of the simulated experiments and the Monte Carlo experiments keeping a constant radius $r_i = 35\,\text{cm}$ ($S = 29$) and varying the number of robots from $N = 5$ to $N = 40$. 21

4.1 The e-puck robot 26
4.2 Example of PFSM obtained using AutoMoDe-`Vanilla`: the state STOP is the initial state and it has two conditions that go to the state APT. The state APT has one condition pointing at the state STOP. At the beginning the current state is set to STOP. At each time step after the execution of the current state, the conditions of the current state are evaluated: when a condition is true the state pointed by the condition becomes the current state. 28

5.1 Arena for the aggregation task 31
5.2 Arena for the foraging task. The circle at the bottom of the simulated arena is the light. 31
5.3 Screenshot obtained by the tracking system 33
5.4 **Aggregation – Performance of the obtained controllers using 10k simulation budget.** The plot shows, for each automatic design method, the performance of the 20 controllers (one for each independent run) both in simulation and on the real robots. The white boxplots are the results of evolutionary robotics and the gray ones are the results of AutoMoDe-`Vanilla` 34
5.5 **Aggregation – Performance of the obtained controllers using 50k simulation budget.**The plot shows, for each automatic design method, the performance of the 20 controllers (one for each independent run) both in simulation and on the real robots. The white boxplots are the results of evolutionary robotics and the gray ones are the results of AutoMoDe-`Vanilla` 34
5.6 **Aggregation – Performance of the obtained controllers using 200k simulation budget.** The plot shows, for each automatic design method, the performance of the 20 controllers (one for each independent run) both in simulation and on the real robots. The white boxplots are the results of evolutionary robotics and the gray ones are the results of AutoMoDe-`Vanilla` 35

5.7 **Aggregation – A controller designed by AutoMoDe-`Vanilla`.** At the beginning the robot moves toward the other robots (state RBT e.g. attraction). When it detects the black floor it stops (state STOP). In the STOP state it checks for its conditions. It changes state when it detects the gray floor. It also starts moving, with a 0.25 probability, independently from the floor color. 36
5.8 Aggregation – Resulting behavior of a controller designed using AutoMoDe-`Vanilla` 37
5.9 **Foraging – Performance of the obtained controllers using 10k simulation budget.** The plot shows, for each automatic design method, the performance of the 20 controllers (one for each independent run) both in simulation and on the real robots. The white boxplots are the results of evolutionary robotics, the gray ones are the results of AutoMoDe-`Vanilla` 38
5.10 **Foraging – Performance of the obtained controllers using 50k simulation budget.** The plot shows, for each automatic design method, the performance of the 20 controllers (one for each independent run) both in simulation and on the real robots. The white boxplots are the results of evolutionary robotics, the gray ones are the results of AutoMoDe-`Vanilla` 38
5.11 **Foraging – Performance of the obtained controllers using 200k simulation budget.** The plot shows, for each automatic design method, the performance of the 20 controllers (one for each independent run) both in simulation and on the real robots. The white boxplots are the results of evolutionary robotics, the gray ones are the results of AutoMoDe-`Vanilla` 39
5.12 Foraging – The two classes of controllers designed by AutoMoDe-`Vanilla`. 40

Chapter 1

Introduction

In this report, we introduce a novel method to automatically generate control software for swarm robotics systems.

Swarm robotics [15] is an approach to robotics in which a large number of robots cooperates to solve a task that would be impossible to solve for a single robot. A robot swarm, is an highly redundant system that acts in a self-organized way without the need of any form of centralized coordination. The collective behavior of a swarm is the result of the local interactions that each robot has with its neighbors and with the environment. We refer the reader to [9] for a complete review on swarm robotics.

The self-organized and distributed nature of a robot swarm helps in developing large robotics systems that are *scalable* with respect to the number of robots composing the swarm, *robust* against the failure of some of the robots, and *flexible*, that is, able to cope with a wide range of environmental conditions.

Unfortunately, the self-organized and distributed nature of robot swarms has a downside: it poses major challenges in the design of robot swarms. The requirements are typically expressed at the swarm level, by specifying the task that the swarm, as a whole, has to perform. However, the collective behavior of the swarm cannot be designed directly, since it is the result of the complex interactions between the robots composing the swarm, interactions whose result is difficult or impossible to foresee. For this reason, the designer's task is *indirect*: he needs to carefully design the individual behaviors of the robots to obtain the desired collective behavior of the swarm.

At the moment, there is no general approach to this design problem, even though some approaches have been recently proposed [10, 6, 30, 26]. Currently, most robot swarms are designed by hand using a trial-and-error process: an individual behavior is developed 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 and does not give any guarantee on the results.

A different approach to the design and development of a robot swarm are automatic design methods [9]. Automatic design methods are able to develop, using a computation intensive process, the control software of a robot swarm without the intervention of the human designer. The first automatic design method that has been adopted in swarm robotics is evolutionary robotics [37]. Results in evolutionary robotics showed that it is possible to automatically obtain the behavior of a robot swarms for a number of tasks. However, evolutionary robotics has number of limitations, such as the reality gap problem [34] and the difficulty of defining an effective fitness function [42]. Strengths and limitations of evolutionary robotics are discussed in Section 2.1.2.

In this report, we introduce AutoMoDe (automatic modular design). AutoMoDe is a novel automatic approach to the design and development of swarm robotics systems. Given a task, AutoMoDe is able to automatically generate the individual behaviors of the robots in the form of probabilistic finite state machines (PFSM). The individual behaviors are developed so that the resulting collective behavior accomplishes the desired task.

The novelty of our approach is that AutoMoDe generates these individual level PFSMs by searching for the best combination of given preexisting behavioral modules, which we call *atomic behaviors*. Examples of atomic behaviors are: random-walk, go-to-light, follow-robot. In other words, AutoMoDe develops a new controller using an optimization algorithm by selecting: the atomic behaviors, the topology of the PFSM, the transition rules and the internal parameters of the selected atomic behaviors.

In this report, we evaluate a proof-of-concept version of AutoMoDe using two tasks commonly studied in the swarm robotics literature: aggregation and foraging. The obtained results show that AutoMoDe automatically generates controllers able to tackle the two tasks with good performance. Moreover, the obtained controllers are naturally understandable for a human user and can be directly deployed on real robots without performance loss.

The rest of the report is organized as follows: in Chapter 2 we provide an overview of our research project, and by analyzing the state of the art, we highlight the motivations behind AutoMoDe. In Chapter 3 we present some preliminary studies on evolutionary robotics. In Chapter 4 we describe the AutoMoDe and its first proof-of-concept version AutoMoDe-`Vanilla`. In Chapter 5 we present the results achieved by AutoMoDe-`Vanilla` on the design of controllers for two tasks: aggregation and foraging. Finally, in Chapter 6 we draw some conclusions highlighting some future research di-

rections on AutoMoDe.

Chapter 2

Our Research Project

The goal of our research project is to develop a method to generate automatically modular controllers for swarm robotics systems. Swarm robotics [16, 8, 5] is an approach to collective robotics that takes inspiration from the self-organized behaviors of social animals [13]. Swarm robotics aims to develop large robotics systems that are scalable, robust and flexible.

In the design of swarm robotics systems, the requirements are typically expressed at the swarm level, by specifying the task that the swarm, as a whole, has to perform. The functioning of the swarm and its performance on the given task are the result of the behavior of each individual robot [16]. Currently, a fundamental open problem in the design of swarm robotics systems is bridging the gap between the desired collective properties of the swarm and the implementation of the individual behavior of each robot.

To fill this gap, two common approaches have been proposed in the literature: the behavior-based approach [11] and the evolutionary approach [37]. In the behavior-based approach, the designer manually defines the behavior of the individual robots on the basis of the swarm level specifications. At the moment, no general formal method exists for determining how the individual robots should act so that the swarm performs a given task or displays desired properties. As a consequence, designing a system with the behavior-based approach is mostly a trial and error process. Thus, the quality of the result strongly depends on the experience and intuition of the designer. In the evolutionary approach, on the other hand, the individual controllers are obtained through an automatic optimization process inspired by natural evolution. The quality of the result depends on the ability of the designer to define the objective function that is to be maximized by the optimization process. The obtained controllers are black boxes that can hardly be analyzed, verified and maintained [34].

Our idea is to combine the advantages of the behavior-based approach and

the evolutionary robotics approach by developing a method for the automatic design of controllers that yields modifiable and modular controllers. In the proposed method, a controller is a probabilistic finite state machine in which the states are behavioral modules. This probabilistic finite state machine is configured automatically using well-tested optimization algorithms, thus relieving the designer of the difficult task of defining the interactions between different behaviors.

The main expected result of our research is the development of an automatic design method that leads to modifiable, engineered and maintainable controllers. We will implement our automatic design method as a software suite that we will release as an open-source project.

2.1 State of the art

In swarm robotics, the most commonly used design approaches are the behavior-based approach and the approach of evolutionary robotics.

2.1.1 Behavior-based Approach

In the behavior-base approach, the designer, using his experience, derives from the swarm specification the controller of the robots. The single behaviors (e.g., obstacle avoidance or phototaxis) are implemented as modules that can be combined in order to create complex controllers. The interaction among these modules are commonly managed by a complex probabilistic finite state machine (henceforth PFSM) [41]. Tuning these PFSMs is a complex task due to the multitude of interactions among the different modules. The main advantage of behavior-based approach is code reusability: a behavior can be reused or adapted to different problems. The main drawback of the behavior-based approach is the lack of a general method to define individual behaviors that meet the given swarm-level specifications [42, 3].

2.1.2 Evolutionary Robotics

Evolutionary Robotics is an automatic design method that applies artificial evolution techniques [23] to multi-robot systems [37]. In evolutionary robotics, the controllers are obtained through a process of artificial evolution in which a population of controllers is evaluated and, like in natural evolution, the best performing controllers are more likely to survive. The process is a series of generation. From one generation to another the controllers are

modified and combined using operations inspired by the natural evolution (i.e. mutation and crossover).

Typically the optimization algorithm is a genetic algorithm [27], that evolves a population of controllers based on *artificial neural networks*. For our purposes, an artificial neural network is simply a function with a certain number of input values and output values[1]. The mapping from the inputs to the outputs depends on the parameters of the neural network. Generally, in robotics, inputs are mapped to sensors and outputs to actuators. The controllers search space is defined by the free parameters of the artificial neural network. By instantiating these free parameters, the optimization algorithm can regulate the fine-grained aspects of the robot behavior. We refer the reader to [43] for more information about evolutionary robotics.

Evolutionary robotics has been successfully applied in many scenarios. In particular, evolutionary robotics is used as a tool to explore the possibility of obtaining automatically behaviors with defined properties. For example, in [4] an experimental analysis is carried out to investigate the evolution of communication strategies among robots. More recently, in [18] we applied an evolutionary robotics method to automatically design an aggregative behavior that shows dynamics that are similar to a biological model.

Artificial neural networks have been proved to be a successful way of controlling the behavior of the robots because of their versatility, that is, their ability to represent a wide range of different output functions. Virtually any function input-output can be obtained using a properly configured neural network. Thanks to this versatility, evolutionary robotics allows the designer to obtain controllers without having an *a priory* knowledge of the collective behavior needed to solve the task.

Since neural networks are able to emulate any input-output function they are defined *bias-free*, that is, they do not require any *a priory* knowledge on the output function to obtain. This extreme versatility of the neural networks has a downside: the quality of a neural network strongly depends on the process followed to configure it. This is known in the literature as the *bias and variance tradeoff* [22]. In the context of automatic design, the designer has to find the right balance between introducing a bias that can lead to neural networks that are not versatile enough to carry out the desired task and having a bias-free neural network that is highly versatile but also is highly dependent on the process followed to configure it. In particular, since the process of configuration of the neural network involves a simulator there is a very high risk of obtaining controllers that are affected by the reality

[1] for the sake of simplicity, we refer here only to feed-forward neural networks although our considerations can be extended also to recurrent neural networks

gap, that is, controllers which, while performing well in simulation, fail or have poor performance once installed on the real robots.

Beside the reality gap there are other critical aspects about the use of neural networks as controllers: neural network based controllers are not easily understandable and modifiable by the designer. In fact, since the parameters composing the neural network's configuration are meaningless and unstructured, understanding and/or modifying the resulting behavior of the controller is almost impossible.

2.2 Our original idea

Our idea is to develop a method for the automatic generation of modular and maintainable controllers. The controllers that our method will produce will be probabilistic finite state machines (PFSM) in which states are behavioral modules. Each of these modules implements a specific, self-contained behavior that the robot exhibits. These modules can be implemented independently by using any technique, even by a recursive application of our method. Each module receives input (e.g., sensor values) and computes output (e.g., actuator commands). The transitions between the behavioral modules are governed by the PFSM and depend on the input and output of the behavioral module.

Each behavioral module has a set of parameters that governs its functioning. These parameters, defined during the implementation, allow the behavioral module to be adapted to different applications. Also the PFSM has a set of parameters (e.g., transition probabilities), which regulate the interaction between the behavioral modules. In our method, the parameters of the PFSM and the parameters of each behavioral module are tuned in order to obtain a controller that, once installed on all the robots of the swarm, allows the swarm itself to perform the specific desired task. The search for good parameter values is an optimization problem that can be tackled using different techniques such as evolutionary algorithms, metaheuristics or parameter tuning techniques [23, 28, 7].

In order to perform the search for good parameter values by the automatic technique, the designer has to define an objective function that measures the swarm-level performance obtained via a given controller. In our method, the objective function does not necessarily evaluate the swarm-level performance as a whole. Instead, we plan to use credit assignment techniques [17, 45] to reward some specific swarm-level actions that are functional to meeting the given specifications, even if the overall performance of the swarm is poor.

Using our method, the outcome (i.e. the controller) of the design process

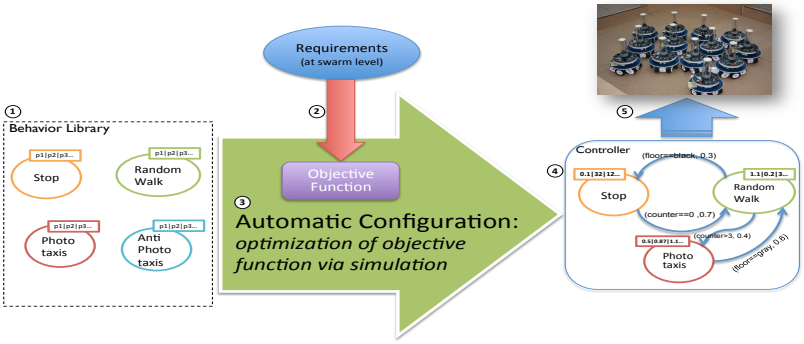


Figure 2.1: The design process:
1. The behavior library is a collection of behavioral modules (Each behavioral module has a set of parameters in order to be applicable in different scenarios).
2. The designer derives an objective function from the requirements.
3. The automatic configuration evaluates automatically many candidate controllers using the objective function.
4. The automatic configuration obtains a controller that meets the requirements.
5. The obtained controller is installed on the robots.

is the same of the behavior-based approach, that is, a PFSM in which the states are behavioral modules. On the other hand, the design process is different since an automatic configuration algorithm substitutes the trial-and-error approach of the human designer. Figure 2.1 shows the design process to obtain a controller in our method, while an example of automatic configuration algorithm is described in Figure 2.2. Our method combines the advantages of the behavior-based approach and the evolutionary robotics. As in the evolutionary robotics, in our method a controller consists of behavioral modules and, as in the behavior-based approach, the generation of the controller is automatic. The effort to build complex behaviors is reduced compared to the behavior-based approach, because the complex problem of defining the PFSM is solved automatically. In addition, unlike in the behavior-based approach, in our method the parameters of the controller are meaningful. This leads to controllers that can be modified and reused. Since the structure of the controller is defined by a PFSM, it is possible to reuse parts of structure itself. Finally, the parameter search space is smaller and structured, making

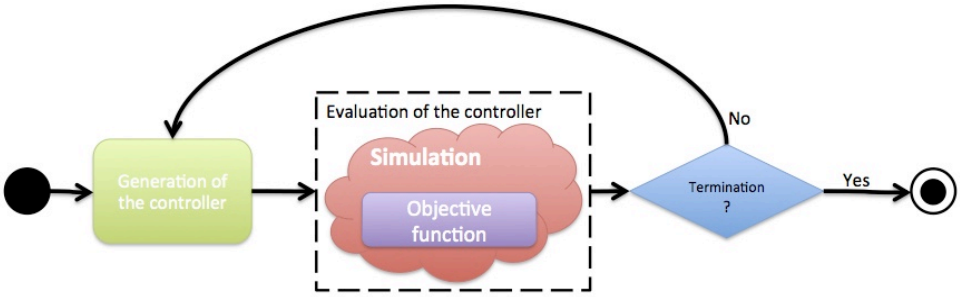


Figure 2.2: An example of automatic configuration using a local search:
1. A candidate controller is generated by combining the behavioral modules of the behavior library (The search in the space of all the possible controllers is done using a local search.)
2. The controller is evaluated in simulation using the objective function:
The controller is installed on all the virtual robots of the swarm and the simulation is run.
The objective function evaluates the performance that the swarm obtains in simulation.
3. If a termination criterion is fulfilled the current controller is returned. Otherwise the algorithm goes back in 1

this framework applicable to more complex problems.

In the context of this project, there are several possible directions to extend the work we proposed above. For instance, we wish to investigate the possibility of selecting automatically the structure of PFSM using model selection methods [12] from statistics and machine learning [38]. Another possible direction that we would like to explore is the application of self-adaptation methods that allows the robots to modify their parameters on the fly, as a result of the interaction with the environment.

We will conduct the experimental analysis using both simulations and real-robot experiments. Studying swarm robotic systems using simulation has the advantages of being faster and cheaper than experiments with real robots. In particular, in the context of our project, simulation has a crucial role because it allows the optimization algorithm to evaluate quickly the performance of the candidate controllers. Since simulation cannot reproduce all the details of reality, we will carry out experiments with real robots in order to validate the results obtained in simulation.

To demonstrate the validity of our method we will test it on complex swarm robotics problems. These problems will be representative of classes of real applications. At the moment, we have identified a candidate problem:

foraging. Foraging is a classic test-bed application in swarm robotics, in which robots have to retrieve prey objects from the environment and bring them back to the nest [46]. This problem is widely analyzed in literature [31, 21, 40], which allows me to compare our method with existing ones.

2.3 Vision

The vision behind our project is that in the future the robotics industry will be similar to the software industry. Nowadays, the software industry is composed by many companies each one developing software for a high specialized domain. Each one of those companies has deep knowledge on its domain of specialization. This knowledge is represented, among other things, by reliable software modules. When a new software is produced, those software modules are combined and instantiated to meet the requirements of the new customer. Similarly, we think that in future the robotics industry will be composed by many robotics companies specialized in solving a particular kind of task. Each one of those companies will provide swarm robotic systems for a particular real world application like oil spill cleanup, fruit collection, pest control, search and rescue, etc. In the same way as software companies, which have library of software modules, robotics companies will have library of behavioral modules. Those behavioral modules will be then combined and instantiated to obtain controllers for solving the particular task requested by the customer.

Differently from the software industry where the modules are combined by hand, in swarm robotics the combination between behavioral modules is a difficult task due to the complexity of the interactions between those behavioral modules. In this context, our method will be really useful: an optimization algorithm automatically designs the controller by combining and instantiating the behavioral modules of the library.

One of the scenarios of possible application of swarm robotics is search and rescue. In this scenario, rescuers have to search for survivors in disaster zones. Due to the hazardous environments, rescuers might be killed or wounded in action. For this reason, the use of robotic rescuers instead of human rescuers has been studied and some remote controlled robotic rescuers were used in two recent disasters, the Fukushima nuclear disaster and the Costa Concordia disaster. However, in those occasions the traditional approach based on remote controlled robots showed restrictions that limit its employment. In particular, this approach assumes a real-time connection between the operator and the robot that is difficult to ensure in hazardous and not structured environment. On the contrary, an approach based on a swarm

of autonomous robots does not need such a continuous connection and can be used in a wider range of situations. In this context, our method can be used since it speeds up the design of controllers for swarm of robots. We imagine that robotic company specialized in search and rescue will have a library of behavioral modules for robots locomotion in hazardous environments and for detecting survivals. In case of a disaster, this company will launch the automatic configuration algorithm with all the information gathered from the disaster (i.e., estimated number of missing people, environmental conditions, weather conditions, etc.) in order to obtain a controller that fits better the particular situation of the disaster.

We think that in future swarm robotics will play an important role in the real world. In this context, we believe that our method will promote the use of swarm robotics in our every day life by simplifying the design and development process.

Chapter 3

Preliminary studies in evolutionary robotics:

Analysing an Evolved Robotic Behaviour Using a Biological Model of Collegial Decision Making

Evolutionary robotics can be a powerful tool in studies on the evolutionary origins of self-organising behaviours in biological systems. However, these studies are viable only when the behaviour of the evolved artificial system closely corresponds to the one observed in biology, as described by available models. In this study, we compare the behaviour evolved in a robotic system with the collegial decision making displayed by cockroaches in selecting a resting shelter. We show that artificial evolution can synthesise a simple self-organising behaviour for a swarm of robots, which presents dynamics that are comparable with the cockroaches behaviour.

3.1 Introduction

In recent studies, evolutionary robotics (ER, see [37]) has been used as an instrument to investigate the evolutionary conditions for the emergence of adaptive behaviour in groups of interacting agents. The main motivation behind these studies is that the evolution of certain adaptive traits and behavioural responses is tightly linked to ecological and social conditions. These conditions are extremely difficult or impossible to be controlled and replicated with empirical studies [1], while they can be completely managed in ER studies. The use of ER to analyse adaptive behaviours has been demonstrated in several occasions. For instance, the effects of genetic relatedness on

the evolution of cooperative communication strategies can be investigated by systematically varying the composition of interacting groups [44, 35]. Similarly, thanks to a simple ER experiment, it has been shown that the effect of stochastic variations in the evolutionary history could be at the basis of the emergence of diverse signalling strategies [47].

At the same time, ER represents a powerful design tool for the synthesis of collective, self-organising behaviours in swarms of robots [43]. It provides an automatic design methodology to synthesise the individual mechanisms leading to an optimal group response, according to a user-defined performance metric. Additionally, ER can shed light on the evolutionary pressures leading to the emergence of observed collective behaviours. However, it is necessary to understand whether or not the target behaviour can be evolved in the artificial system, and whether it displays dynamics comparable with the natural counterpart.

In this study, we perform this first step, that is, the validation of an ER system with respect to collegial decision making by cockroaches in selecting a resting shelter [2]. Cockroaches (*Blattella germanica*) are gregarious insects that manifest cooperative behaviour in selecting a resting site: whenever more than one site is present, the insects collectively choose to aggregate in one single place (provided that it is large enough to host them all). Experimental studies allowed to determine which are the social influences that lead to such a collegial decision-making process, and a dynamical model has been developed (see [2] and Section 3.3.1 for more details). The identified mechanisms have been successfully exploited for designing collective aggregation and decision-making behaviours in swarms of robots [20, 14, 10], allowing also mixed insect-robot experiments [25]. However, to the best of our knowledge, there has been no attempt to study the evolution of a similar decision-making behaviour in swarms of robots. In this study, we demonstrate that similar collegial decisions can be evolved in an artificial system. Our goal is to (i) verify the evolvability of the collegial decision making in the artificial system, and (ii) determine whether the dynamics of the system correspond qualitatively and quantitatively to the ones predicted by the biological model [2]. This will allow us to determine whether or not evolutionary robotics is suitable for formulating hypotheses about the evolutionary pressures that resulted in collective decision-making in cockroaches.

The chapter is organised as follows. In Section 3.2, we describe in detail the experimental setup for the ER experiments. In Section 3.3, we discuss the results obtained from the evolutionary experiments with respect to the evolvability of the decision-making behaviour in a robotic system. In Section 3.3.1, we present the dynamical model proposed in [2], and we discuss the methodology that leads us to fit the evolved behaviour to the model.

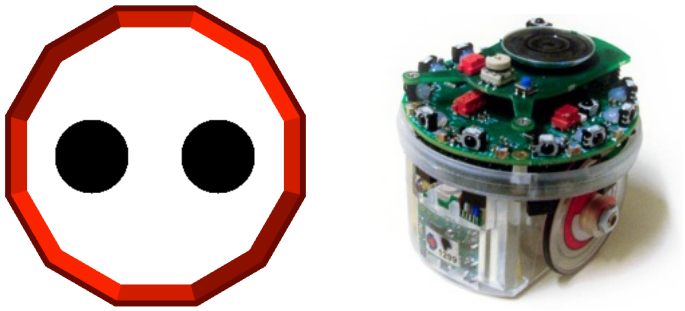


Figure 3.1: Left: the simulated experimental arena used for the experiments. Right: the e-puck robot, used for the simulated evolutionary experiments presented in this work.

In Section 3.3.2, we present a comparison of the dynamics of the evolved behaviour with the ones predicted by the model. Section 3.4 concludes the chapter with some final remarks.

3.2 Experimental Setup

We study the evolution of collegial decision making in a swarm of robots that have to aggregate in one of two areas within the experimental arena. Our experimental setup is based on the one used in Amé et al. [2]. The robots operate in a dodecagonal arena (Figure 3.1 left) of area 4.91 m^2 surrounded by walls. The floor of the arena is white with two black circular areas having the same radius ($r_a = r_b = 35$ cm) and centred at 67 cm from the walls. In the following, we refer to the two black areas as area a and b, and the remaining white area as c.

The experiments are carried out in simulation using ARGoS [39], a multi-engine simulator of swarm robotics systems. The robots and the environment are modelled using a 2D dynamic physics engine. We use a simulated model of the e-puck robot (Figure 3.1 right), a small wheeled robot designed for research and education [36]. In our experimental setup, each robot can perceive walls and other robots through eight infrared proximity sensors placed all around its chassis. It can sense the colour of the floor using three ground sensors placed under its front. Additionally, each robot features another sensor called *range and bearing* [24]. This sensor allows the robot to communicate locally with other robots by sending and receiving messages. In our experiments, the robot uses such a sensor only to perceive the number of other robots within a 70 cm range. To normalise the output of the sensor, we use the preprocessing function $z(n) = 1 - (\frac{2}{1+e^n})$, where n is the number

of robots perceived at any given moment. Since the real e-puck can perceive no more than 5 robots at a given time, $z(n)$ saturates to 1 for $n > 5$.

The controller that governs each robot is an artificial neural network. We assume that robots can achieve aggregation using a memoryless behaviour, that is, the behaviour of each robot depends only on the present values of sensors without any kind of internal state. For this reason, we use as controller a fully connected, feed-forward neural network. This neural network has 12 inputs, one for each sensor (8 infrared proximity, 3 ground sensors, 1 from the range and bearing), 2 outputs, one for each wheel, and no hidden units. The input values are linearly scaled in [0,1] when necessary. 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 two output neurons control the speed of the two wheels, by scaling their activation in the range $[-v_m, v_m]$, with $v_m = 16\,\mathrm{cm/s}$.

We use a simple evolutionary algorithm to set the parameters of the neural network. Each parameter is represented in the genotype by a real number in the range [-5,5]. The evolutionary algorithm works on a population of 100 genotypes, evolved for 200 generations. The population of the first generation is randomly generated. Subsequent generations are created using a selection and reproduction process that involves elitism and mutation. The 20 best genotypes—i.e., the elite—are included unchanged in the next generation. The remaining genotypes of the population are generated by mutation of the genotypes of the elite. The mutation is done by adding a random value to each element of the genotype. The random value is drawn from a normal distribution with mean 0 and variance 1.

The genotype is mapped into a controller that is instantiated in all the robots of the group ($N = 10$). To evaluate the performance, 10 trials of $T = 250$ seconds are run. The evaluation of the performance of the genotype is based on the function $f(t)$:

$$f(t) = \frac{|x_a(t) - x_b(t)|}{N} \in [0, 1] \tag{3.1}$$

where $x_i(t)$ is the number of robots in area $i \in \{a, b\}$ at time t and N is the total number of robots. The function $f(t)$ is equal to zero when a and b contain the same number of robots. On the contrary, $f(t)$ is equal to 1 when all the robots aggregate on the same area. Fluctuations of $f(t)$ are smoothed through an exponential moving average with time constant $\alpha = 0.9$:

$$G(t) = \alpha G(t-1) + (1-\alpha) f(t) \in [0, 1] \tag{3.2}$$

where $G(0) = 0$. Finally, the fitness F of the genotype is the average of $G(T)$ over all the 10 trials.

3.3 Results

We performed 20 evolutionary runs starting from different randomly generated populations. For each run, we selected the best controller within the final population: we evaluated the performance of every controller of the last generation for $K = 200$ trials, and we selected the one with the highest average fitness. All the evolutionary runs were able to produce controllers with high performance (data available as supplementary material in [19]).

A qualitative analysis of the obtained controllers reveals that the evolved behaviours are quite similar one to the other. In general, the robots act differently according to their position in the arena. When a robot is in the white area c, it explores the arena following a wide curved trajectory. If the robot reaches the external wall of the arena, it starts to follow it. The robot motion is influenced by the presence of other robots: curves become sharper when other robots are nearby. Such a perturbation makes the robot leave the border of the arena and eventually enter in one of the two black areas. When the robot is in one black area it follows a circular trajectory. The radius of the trajectory decreases as the number of robots in the area increases. In this way, if the area is empty the robot follows a wide trajectory and eventually leaves. On the contrary if the area is crowded the robot almost rotates on its axis. If the robot goes out of the black area it starts again to explore the arena. Example videos of the obtained controller are available as supplementary material [19].

There are qualitative similarities between the evolved behaviour just described and the self-organizing aggregation behaviour observed in groups of cockroaches. In particular, we observed that the probability that a robot leaves an area is inversely proportional to the number of the robots located in the area itself. To determine whether or not the evolved behaviour presents dynamics quantitatively similar to the biological system, we check the adherence of the evolved robotic behaviour[1] with the model introduced in [2]. In Section 3.3.1, we introduce the model and the methodology we used to estimate its parameters. In Section 3.3.2, we compare the dynamics of the evolved behaviour with the predictions of the mathematical model.

3.3.1 Model

In Amé et al.'s model [2], the behaviour of each individual insect is characterised by J_i, its probability to join area i, and L_i, its probability to leave area i. Both probabilities depend on x_i, the number of insects located in area

[1] To this aim, we select the best obtained controller among all evolutionary runs.

i, and on S, the *carrying capacity*, that is, the maximum number of insects that can be hosted in a single area.

The joining probability J_i decreases slightly with the number of insects in area i because of crowding effects. This accounts for the observation that it is less probable to join an area that is already densely populated. Amé et al. define J_i as:

$$J_i = \mu \left(1 - \frac{x_i}{S}\right), \quad i = [a, b]; \tag{3.3}$$

where μ represents the area quality, that is, the probability that an individual joins the area without social influences, x_i is the number of insects already in area i, and S is the carrying capacity.

Similarly, the leaving probability L_i is inversely proportional to the number of individuals in area i. This accounts for social influences among individuals, which tend to stay close together. L_i is low when the area is densely populated and high when it is sparsely populated. Amé et al. define L_i as:

$$L_i = \frac{\theta}{1 + \rho \left(\frac{x_i}{S}\right)^2}, \quad i = [a, b]; \tag{3.4}$$

where θ depends on the quality of the area, and ρ is a reference surface ratio related to the area carrying capacity. Using J_i and L_i it is possible to describe the time evolution of the number of individuals in the different areas through a system of differential equations:

$$\frac{dx_i}{dt} = J_i x_c - L_i x_i = \mu x_c \left(1 - \frac{x_i}{S}\right) - \frac{\theta x_i}{1 + \rho \left(\frac{x_i}{S}\right)^2}, \quad i = [a, b] \tag{3.5}$$

$$N = x_c + x_a + x_b \tag{3.6}$$

where N is the total number of individuals and x_c is the number of individuals in c, that is, the individuals outside the black areas. This model therefore describes the dynamics of the aggregation behaviour in terms of the number of individuals present in the different areas of the arena (see [2] for details).

To evaluate the correspondence of the evolved behaviour with the model, we estimated the model parameters from the results of simulated experiments.

The carrying capacity S was estimated using a linear function $S = m\frac{r_a}{r_r} + q$, where r_a is the radius of area a, and r_r is the radius of the robot. We performed 200 trials observing how many robots could be hosted in an area using $N = 100$, $r_a = [0.15, 0.35, 0.45, 0.5]$ and $r_r = 0.035$. The obtained parameters are $m = 4.24$ and $q = -13.38$ ($R^2 = 0.999$, $p\text{-}val < 0.001$).

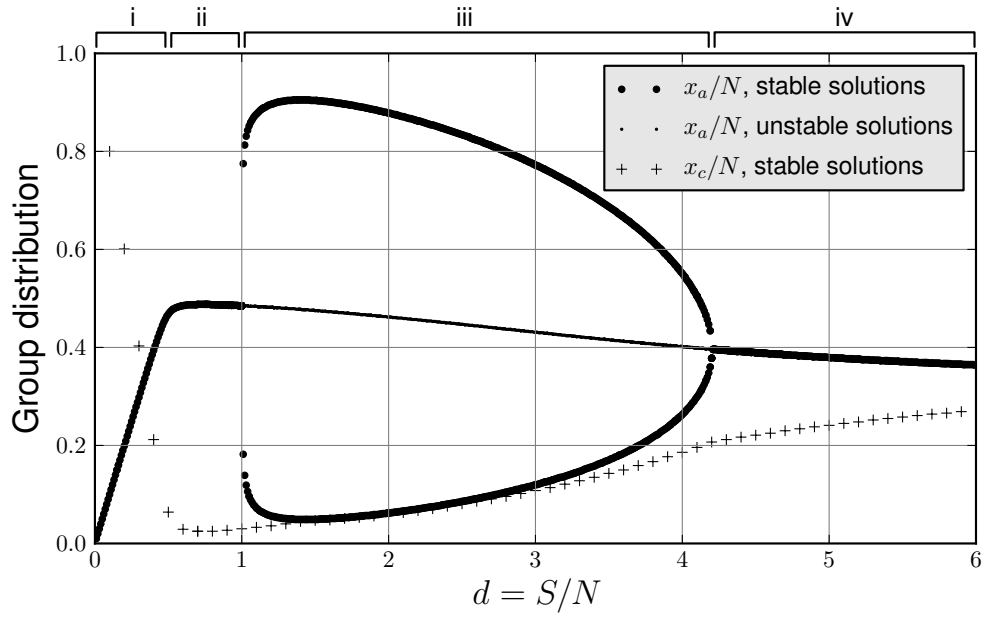


Figure 3.2: The bifurcation diagram of our model for different values of $d = \frac{S}{N}$. The percentage of robots in area a and area c.

To estimate the parameters of J_i and L_i, we gathered the empirical probabilities by performing 200 simulated experiments in the same conditions as presented in Section 3.2 ($N = 10$, $r = 0.35\,\text{cm}$, $S = 29$). We separately fitted the parameters for J_i and L_i to our data using the non-linear least squares method. The obtained parameters are: $\mu = 0.008$, $\theta = 0.008$ and $\rho = 138.574$. We measured the quality of the fitting by computing the coefficient of determination R^2. While the fitting on L_i is excellent ($R^2 = 0.979$, $p\text{-}val < 0.001$), the fitting on J_i is not as good ($R^2 = 0.560$, $p\text{-}val = 0.148$). This is due to the fact that in our robotic system, J_i appears to be non-linear, differently from Amé et al.'s model. Even though the fitting is not good, we decided to be consistent with Amé et al.'s model and not change J_i. A discussion of the possible effects of this decision is presented in Section 3.4.

Following the analysis presented in [2], we studied the system behaviour described by eq. (3.5) and (3.6) for different values of $d = \frac{S}{N}$. In Fig. 3.2, it is possible to see the bifurcation diagram of the model. Four different situations can be observed: (i) For d lower than 0.5, the areas are too small to host all the robots; the robots fill completely the areas and some remain in c. (ii) For $0.5 \leq d < 1$, a single area is too small to host all the robots, so the areas are filled equally. However, in this second case, since there is enough space on the areas for all the robots, only few robots are on c. (iii) For $1 < d \leq 4.2$

the areas are big enough for aggregation to happen. Two stable solutions are found, corresponding to area a or area b hosting the majority of the robots. Additionally an unstable solution is found, corresponding to both areas filled equally. (iv) For d greater than 4.2 the areas are too big and the robots are less likely to perceive the presence of other robots in the same area. Thus, the stable solution corresponds to both areas filled equally.

The number of robots present in c described by eq. (3.6) also varies with d. Two different situations can be observed: For d lower than 0.5, x_c/N decreases sharply: as areas a and b get bigger, more space is available and the areas can host more and more robots; For d greater than 0.5, the population fraction on c increases steadily. This is due to the fact that, as S becomes bigger, the probabilities J_i and L_i increase, resulting in a system less likely to converge on a state in which all robots are in areas a or b.

We consider that a collective decision has occurred when $x_a/N > 0.8$. In the bifurcation diagram in Fig. 3.2 this happens only for d between 1 and 2.8. For d between 2.8 and 4.2 the model predicts a more variable condition with a still unbalanced distribution of robots among the two areas, and an increasing number of robots that move from one area to the other. In the following, we verify these model predictions with respect to the experimental data, presenting a comparison between the results obtained in simulation and those obtained with the model.

3.3.2 Dynamics of robotics and model simulations

We compared the results obtained from simulated robotics experiments and Monte Carlo experiments for different values of $d = S/N$. We carried out two different analyses. In the first one, the different values of d are obtained by keeping the number of robots fixed to $N = 10$ and varying the carrying capacity S, by changing r_i. In the second, we keep $r_i = 35\,\text{cm}$ (which corresponds to $S = 29$) and we vary the number of robots. For each value of d, we run 1000 trials of $T = 500$ seconds, both for the robotic and the Monte Carlo simulations. For each trial, we collected the final group distribution x_i over the different areas.

Figures 3.3 and 3.4 show the obtained results. For each value of d, one bar for each area of the arena is reported. The colours in the stacked bars show the frequency of individual distributions. We divided the distributions in five classes (0-20%, 20-40%, 40-60%, 60-80%, 80-100%), giving each class a different colour. The size of each class in the figures is proportional to its frequency. If the robots are able to perform a collegial decision and aggregate in one single area most frequently, the bars of the areas a and b are mostly dark blue, corresponding to a bimodal distribution with peaks in 0-20% and

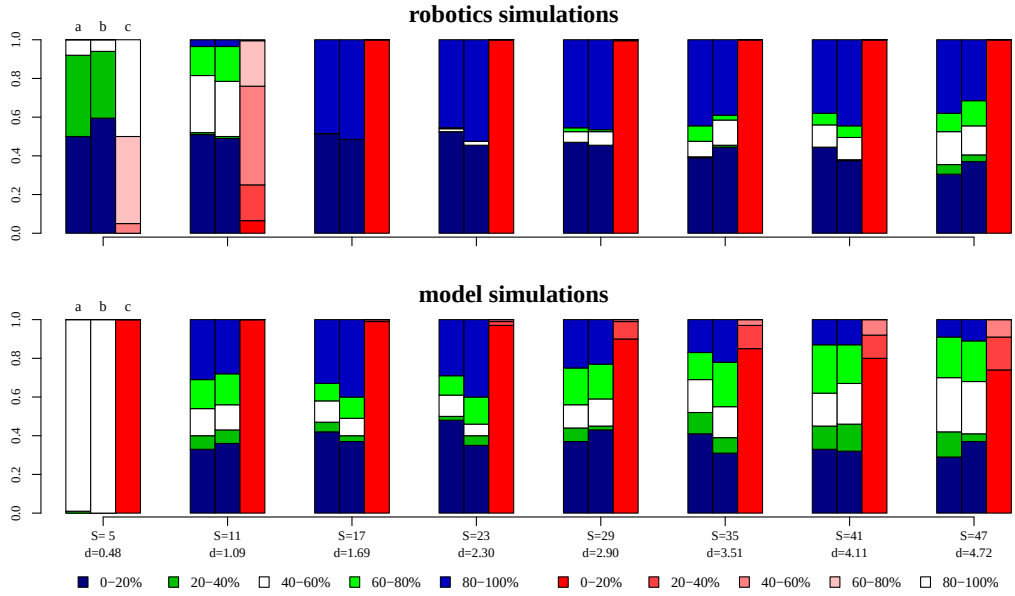


Figure 3.3: Comparison between the behaviour of the simulated experiments and the Monte Carlo experiments keeping the number of robots fixed to $N = 10$, and varying the size of the black areas from $r_i = 15\,\text{cm}$ to $r_i = 50\,\text{cm}$.

80-100%. On the contrary, if the group splits by aggregating in both areas, the bars of the areas a and b are mostly white, corresponding to a unimodal distribution centred in 40-60%. Area c is depicted in dark red when empty (0-20%) and white when full (80-100%).

Figure 3.3 shows the comparison between robotics and model simulations when the number of robots is fixed to $N = 10$. Apart from low values of S, there is a good correspondence between the model and the evolved behaviour. Moreover, the evolved behaviour looks more stable for $d > 2.9$, indicating that the robots have a better tendency to perform collegial decision than predicted by the model. For $S = \{5, 11\}$—corresponding to $r_i = \{15, 20\}\,\text{cm}$—the robots find the areas with difficulty due to the small radius and aggregates are less stable.

Figure 3.4 shows the results when the carrying capacity S is fixed to 29. The evolved behaviour presents a smoother transition from equally occupying the areas at low d to collegial decisions at high d. For $8 < N < 12$ there is a good correspondence, as the robotic system is close to the evolutionary conditions. Differently, for $N \geq 13$ robots split more frequently than aggregating, while the model predicts splitting only when one area is saturated. Overall, we observe a good qualitative correspondence between

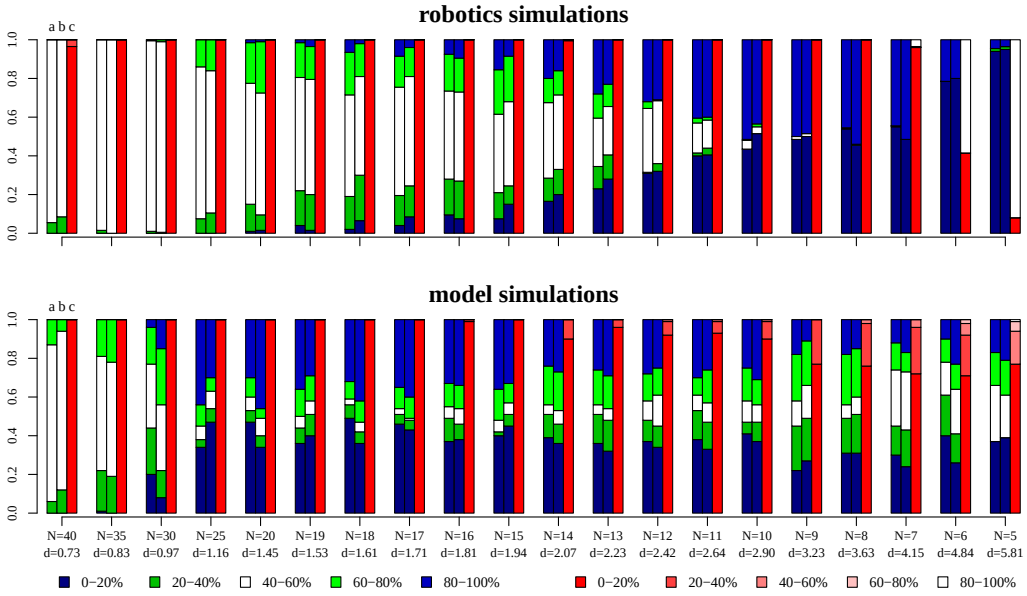


Figure 3.4: Comparison between the behaviour of the simulated experiments and the Monte Carlo experiments keeping a constant radius $r_i = 35\,\text{cm}$ ($S = 29$) and varying the number of robots from $N = 5$ to $N = 40$.

robotics simulations and the model, but mostly within the range of parameters used to evolve the robotics behaviour. A more detailed discussion about these discrepancies follows in the next section.

3.4 Final Remarks

In this study, we demonstrated that evolutionary robotics techniques can be used to synthesise a collegial decision making behaviour similar to the one observed in cockroaches. This is an important result, especially considering that the robotic controllers are simple feed-forward neural networks without internal states. That is, also in a robotic system collegial decisions can emerge solely from simple individual behaviours modulated by social interactions.

We compared the dynamics of the evolved robotic behaviour with the predictions of the model proposed in [2], finding some qualitative correspondence. However, quantitative comparisons revealed similar dynamics mostly for a small parameter range around the evolutionary conditions ($N = 10$, $S = 29$). We identify two reasons for these discrepancies: (i) the evolved system exploits geometric regularities, such as the arena dimension and the positioning of the areas; (ii) the sensing radius for the robots (75 cm) is quite

large with respect to the arena dimensions. Both these issues have a bearing on the probability of joining an area, which also explains the non perfect fit of the model parameters observed in Section 3.3.1. In practice, we observe that the evolved behaviour depends on both S and N, and not only on the their ratio d, as predicted by the model. In future work, by removing geometric regularities from the evolutionary setup, we hope to obtain a better quantitative matching with the model predictions. If successful, we plan to exploit this artificial experimental setup to investigate the optimality of the evolved behaviour with respect to different selective pressures, genetic relatedness among individuals in the group, and variable ecological conditions. We believe this can be useful to better understand the evolutionary path leading to collegial decision making.

Chapter 4

AutoMoDe

4.1 Automatic design methods

In this section, we present the definition of automatic design method for swarm robotics controllers.

Given a task to carry out, the goal of an *automatic design method* (hereafter ADM) is to automatically design and develop a collective behavior for a robot swarm able to effectively tackle the given task. As said before, designing a collective behavior means, in practice, designing the individual controllers of the robots composing the swarm. This means that, ultimately, the goal of an ADM is to design individual controllers that, once instantiated in a robot swarm, result in the proper collective behavior to solve the given task.

In its most generic form, an ADM is an iterative process based on an *optimization algorithm* that explores a set of *candidate controllers* searching for the best solutions for the given task. Such candidate controllers are sampled from the *controller search space*, that is, the set of all possible controllers. Each candidate controller is evaluated using an *objective function* which gives a metric on the quality of the controller, that is, its efficacy in solving the given task when instantiated in a robot swarm. Usually, this evaluation is performed using a *computer simulation* of the chosen *robotic platform* and the *experimental conditions* in which the final system will operate. The objective function is used to select the best candidates which are used for the next iteration of the procedure. In other words, the objective function guides the optimization algorithm in the search for the best solutions. At the end of this process, the obtained solution is instantiated and tested using real robots.

From this general description of an ADM, we can see that a number of

elements (presented in italics above) are part of the *setup*, that is, must be manually chosen by the human designer. In particular, the setup of an ADM is composed of:

The controller search space – The controller search space represents the space of all the possible controllers that the ADM can generate. Usually a controller is composed of two parts: a template, chosen by the human designer, and a set of parameters. The ADM generates individual controller by instantiating each parameter of the template. The human designer needs to carefully choose the template and the range of the parameters in order to have a search space large enough to include all potentially good controllers, but not so large that it hinders the search.

The objective function – The objective function is a mathematical function used to evaluate the performance of the swarm in solving the desired task. It is used to compute a metric that must be maximized or minimized by the optimization algorithm. The objective function needs to be chosen carefully because it plays an critical role in the ADM: it guides the search process of optimization algorithm.

The optimization algorithm – The optimization algorithm explores the controller search space for controllers able to solves the given task. The optimization algorithm evaluates the performance of the candidate controllers through the objective function. The algorithm stops when a termination criteria is fulfilled. Usually, the termination criteria are either in the form of a threshold on the metric given by the objective function, or in the form of the number of evaluations performed by the optimization algorithm.

The robotic platform – The robotic platform, that is, the specific kind of robot composing the swarm, must be chosen considering the capabilities needed to solve the task. The capabilities of the chosen robot influence the ADM: different capabilities result in different behaviors, as the ADM optimizes the use of the sensors/actuators available.

The simulation platform – The simulation platform is a software tool used to simulate the robot swarm and its behavior in a virtual environment. The simulation platform is needed because evaluating the candidate controllers on the real robots is time consuming and potentially dangerous. The main limit in the use of simulators is that it is impossible of perfectly and completely reproduce the robots and the real experimental conditions:

the differences between the simulated experiments and the real-world experiments fall in the so called *reality gap* [34, 29]. The important consequence of the reality gap is that controllers obtained in simulation may have a very different performance when tested in the real world. Since the reality gap cannot be completely removed, the designer has to identify the limits of the selected simulation platform and devise methods to reduce the reality gap, at least in the aspects more relevant to the task to perform. The problem of the reality gap is central to all ADMs.

The experimental conditions– The experimental conditions represents the environment in which the swarm operates. They include the characteristics of the environment as, for example, the size and the geometry of the environment. These experimental conditions have to be carefully replicated in the simulation tool.

4.2 Characteristics of AutoMoDe

In this section we describe the main characteristics and features of the proposed method following the definitions given in Section 4.1.

AutoMoDe (automatic modular design) is a novel automatic method to the design and development of swarm robotics systems. Given a task, AutoMoDe is able to automatically generate the individual behaviors of the robots in the form of a probabilistic finite state machines (PFSM). The individual behaviors are developed so that the resulting collective behavior solves the desired task. The novelty of my proposed method is that AutoMoDe generates these individual level PFSMs by searching for the best combination of given preexisting behavioral modules, which we call *atomic behaviors*. Examples of atomic behaviors are: random-walk, go-to-light, follow-robot. In detail, AutoMoDe develops a new controller using an optimization algorithm by selecting: the atomic behaviors, the topology of the PFSM, the transition rules and the internal parameters of the selected atomic behaviors.

Following the definitions of Section 4.1, in AutoMoDe the controller search space is defined by all the possible PFSMs that can be obtained by combining the atomic behaviors and their internal parameters. This controller search space can be explored using a wide range of optimization algorithms.

The motivation behind AutoMoDe is to define an automatic design method that leads to understandable and modifiable modular controllers that suffer less from the reality gap, that is, they can be installed on the real robots without significant performance loss. The controllers obtained using AutoMoDe

Figure 4.1: The e-puck robot

are human-readable since are based on PFSMs. This leads to controllers that can be also modified by hand when needed. Moreover, the fact that the controllers are based on modules (i.e. atomic behavior) facilitates the reuse of components. With AutoMoDe the reality can be overcome because, the single atomic behaviors can be tested on the real robots.

4.3 Proof of concept: AutoMoDe-`Vanilla`

In this section we describe a proof-of-concept version of AutoMoDe called AutoMoDe-`Vanilla` that is used to carry out the experiments described in Section 5.1. Our goal is not to define the ultimate automatic design method but to show that the core ideas of AutoMoDe are valid. For this reason AutoMoDe-`Vanilla`, is unsophisticated in many aspects like the way in which the probabilistic finite state machines are represented and optimized. We will explore more sophisticated versions of AutoMoDe in future research.

AutoMoDe-`Vanilla` is a version of AutoMoDe, implemented to design controllers for the e-puck robot[1], (Figure 4.1) a small wheeled robot designed for research and education [36].

In the following, we present the various element composing the setup of AutoMoDe-`Vanilla`:

The robotic platform – we chose a swarm of e-puck robots. Each e-puck can move using two differential wheels. The e-puck is equipped with

[1]http://www.e-puck.org/

light, proximity and ground sensors. Moreover, the e-pucks can communicate with each other using the *range and bearing* [24]. This sensor allows the robot to communicate locally with other robots by sending and receiving messages within a 70 cm range. When a robot receives a message it also receives information about the distance (range) and the angle (bearing) of the sending robot.

The simulation platform – The robot swarm is simulated using the ARGoS simulator [39], a multi-engine simulator of swarm robotics systems. The robots and the environment are modeled using a 2D dynamic physics engine. A very interesting feature of ARGoS is that the controllers obtained in simulation can be ported directly on the real robots, making the passage from simulation to real robot seamless.

The controller search space – The controller is defined as a probabilistic finite state machine (PFSM) composed of atomic behaviors linked by conditional state transitions. Both the atomic behaviors and the conditional state transitions use the sensors and actuators of the e-puck. Figure 4.2 shows an example of PFSM obtainable using AutoMoDe-`Vanilla`.

As stated in Section 4.2, one of the unique features of AMD is that the controllers are generated composing pre-available atomic behaviors. In AMD-vanilla these are the atomic behaviors available:

1. *Random walk* (RW): the robot goes straight and when it hits an obstacle it turns around for a random number of steps chosen in the interval $[0, rwm]$, where rwm is a parameter of the atomic behavior.
2. *Stop* (Stop): the robot stays still
3. *Phototaxis* (PT): The robot moves toward towards the closest light source if available, otherwise it moves straight. The light source is perceived using the light sensor.
4. *Anti-Phototaxis* (APT): same as PT, but with opposite direction.
5. *Attraction* (RBT): The robot uses the range and bearing sensor to compute the direction towards the center of the robots in communication range. The direction vector w is computed by aggregating the received messages following the equation:

$$w = \sum_{m \in M} \left(\frac{att}{range(m)}, \angle bearing(m)\right) \tag{4.1}$$

where are att is a parameter.

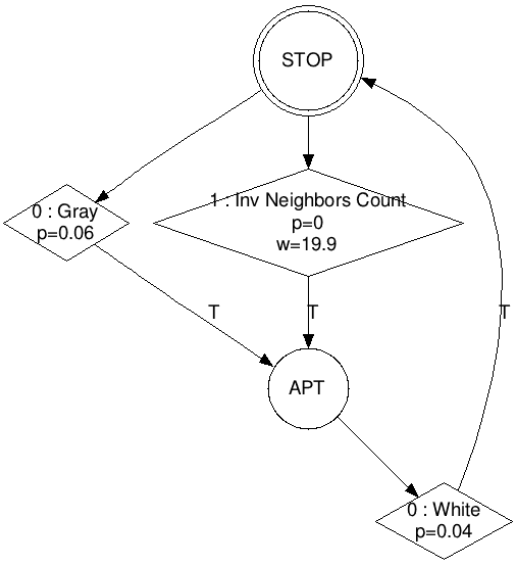


Figure 4.2: Example of PFSM obtained using AutoMoDe-`Vanilla`: the state STOP is the initial state and it has two conditions that go to the state APT. The state APT has one condition pointing at the state STOP. At the beginning the current state is set to STOP. At each time step after the execution of the current state, the conditions of the current state are evaluated: when a condition is true the state pointed by the condition becomes the current state.

6. *Repulsion* (ARBT): Same as RBT, but with opposite direction. The parameter *rep* has the same meaning of *apt* in RBT.

The atomic behaviors are linked with conditional state transitions. The conditions on these state transitions are evaluated at each step after the execution of the atomic behavior. If a condition is true, then the current state updated to the state pointed by the condition. As for the atomic behaviors, the conditions can have parameters. In AutoMoDe-`Vanilla` there are six conditions available:

1. *Black Floor* (Black): The condition returns true if the robot's ground sensor read that the floor is black. False otherwise. The parameter p is the tunable probability of transition.

2. *Gray Floor* (Gray): The condition returns true if the robot's ground sensor read that the floor is gray. False otherwise. The parameter p is the tunable probability of transition.

3. *White Floor* (White): The condition returns true if the robot's ground sensor read that the floor is white. False otherwise. The parameter p is the tunable probability of transition.

4. *Probabilistic Neighbors Count* (Neighbors Count): The condition returns true according to the probabilistic distribution $z(n) = \frac{1}{1+e^{w(p-n)}}$, with n that is the number of robots in the neighborhood, p and w are tunable parameters.

5. *Inverted-Probabilistic Neighbors Count* (Inv-Neighbors Count): The condition returns true according to the probabilistic distribution $z(n) = 1 - \frac{1}{1+e^{w(p-n)}}$, with n that is the number of robots in the neighborhood, p and w are tunable parameters.

6. *Fixed Probability* (Fixed Probability): The condition returns true with a probability p, where p is a parameter.

In order to limit the complexity of the obtained PFSM, we limit the number of state usable by AutoMoDe-`Vanilla` to 4, that is, AMD-vanilla can generate probabilistic finite state machines with up to 4 states where each state can have up to 4 conditions.

The optimization algorithm – The optimization algorithm automatically explores the space of all the possible controllers that can be obtained by combining the behavioral modules and the conditions, and instantiating the respective internal parameters. In AutoMoDe-`Vanilla`, we use F-Race[7], a racing algorithm for tuning metaheuristics.

Chapter 5

Experiments and Results

5.1 Experimental Setup

To asses the capabilities of AutoMoDe-`Vanilla` we carry out a series of experiments in which the proposed method is used to obtain controllers for a swarm of 20 e-pucks in order to solve two different tasks: *aggregation* and *foraging*.

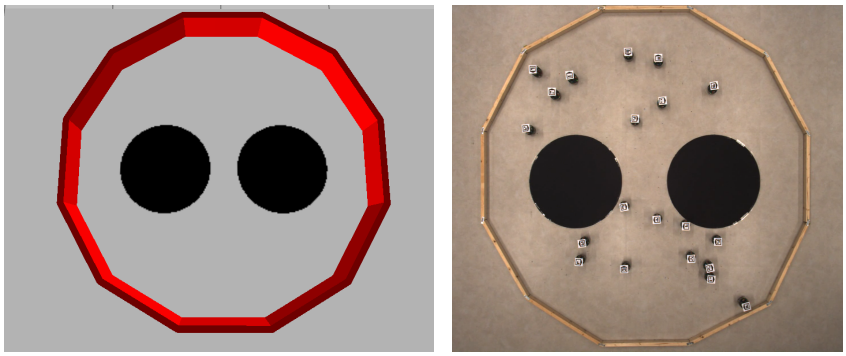
5.1.1 Aggregation

In the aggregation task the swarm of robots has to aggregate on one of the two black areas of the arena's floor. The aggregation task is the same analyzed in [18] and presented in Chapter 3. Figure 5.1 shows the arena for the aggregation task in both simulation and reality. The arena is a dodecagonal area of $4.91\,\mathrm{m}^2$ surrounded by walls. The floor of the arena is gray and there are two black circular areas on the floor, a and b, that have the same radius ($r_a = r_b = 35\,\mathrm{cm}$) and are centered at 67 cm from the walls.

To evaluate the controllers simulations of $T = 250$ s are run. A swarm of 20-epuck equipped with the controller to test, is randomly distributed in the arena and then started. The objective function for the aggregation task is calculated as:

$$F = \frac{max(x_a(T), x_b(T))}{N} \in [0, 1] \tag{5.1}$$

where $x_a(T)$ and $x_b(T)$ are the number of robots present on the black areas a or b at the end of the simulation and N is the total number of the robots composing the swarm. This objective function is equals to 1 when at the end of the simulation all the robots are aggregated on one of the two areas.

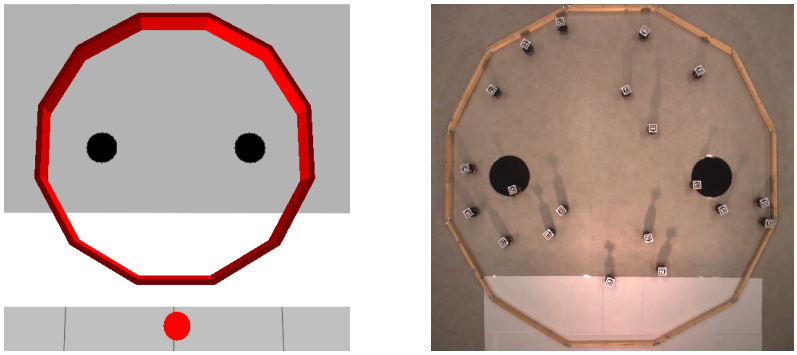


(a) simulated arena (b) real arena (and 20 e-pucks)

Figure 5.1: Arena for the aggregation task

5.1.2 Foraging

The foraging task consists of retrieving objects from the food sources represented by black circles on the floor and storing them in the nest area represented by the white floor. Figure 5.2 shows the arena for the foraging task in both simulation and reality. The arena has the same shape and size of the aggregation task, but the two black areas have a radius of 20 cm. Moreover, there is a light behind the nest area that the robots can perceive through the light sensors.



(a) simulated arena (b) real arena (and 20 e-pucks)

Figure 5.2: Arena for the foraging task. The circle at the bottom of the simulated arena is the light.

The objective function for the foraging task is calculated as the number of objects retrieved and stored in the nest: each time a robot goes on a black area and then goes back to the white area the objective function is incremented by one.

5.1.3 Setup of evolutionary robotics

In our experimental setup, we compare the results of AutoMoDe-`Vanilla` with an evolutionary robotics method. For the evolutionary robotics method we use the same setup described in Chapter 3, here we report the setup of the controller while for the description of the evolutionary algorithm we refer the reader to Section 3.2. The e-puck can perceive obstacles and light intensity using respectively the proximity and the light sensors. The color of the floor is read using three ground sensors. The range and bearing is used to calculate the number of robots in a 70 cm range and the vector w that is a vectorial sum of the positions of the perceived robots. The controller is a fully connected, feed-forward neural network. This neural network has 24 inputs, 2 outputs and no hidden units. The inputs are: 8 proximity sensors, 8 light sensors, 3 ground sensors and 5 aggregated input from the range and bearing: one input is obtained using $z(n) = 1 - \frac{1}{1+e^n}$ where n is the number of the perceived robots. The other four input are computed as scalar projection on the angles $45°, 135°, 225°, 315°$ of the vector perceived robots w (computed following the Equation 4.1 with $att = 1$). 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 the neural networks regulate the speed of the two wheels, by scaling their activation in the range $[v_m, -v_m]$, with $v_m = 16\,\mathrm{cm}/s$.

The neural network has a set of 50 parameters. Each parameter is a real value in the range $[-5, 5]$. To set these parameters we use the evolutionary algorithm described in Section 3.2.

5.2 Results

For each task we run three experimental sessions. The sessions differ on the *simulation budget*, that is, the total number of simulations used by each design method in the search for the best controller. The three simulation budgets are: 200000, 50000, and 10000. For each simulation budget we perform 20 independent runs for both AutoMoDe-`Vanilla` and evolutionary robotics. Since at the end of each run the design method returns one controller. The 20 controllers obtained by each method are then evaluated once in simulation and once on the real robots. The evaluation on the real robots is done using a tracking system that allows us to compute the objective functions by tracking the position of the robots using a ceiling camera and a marker on the top of each robot. Figure 5.3 shows a screenshot obtained by the tracking system.

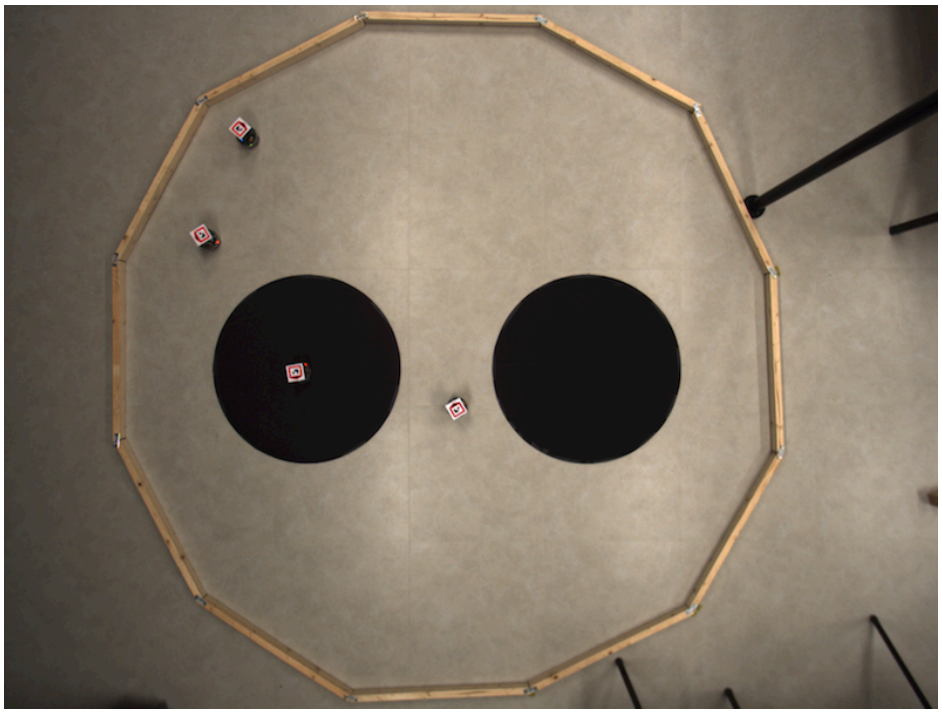


Figure 5.3: Screenshot obtained by the tracking system

5.2.1 Aggregation

For assessing the performance of AutoMoDe-`Vanilla` and for comparing it with evolutionary robotics we observe two aspects of the results. First of all, we compare the performance of the two design methods on the real robots in order to understand which one is able to deploy better controllers. Secondly, within the same design method we compare the performance in simulation and on the real robots in order to evaluate the reality gap.

Figures 5.4, 5.5 and 5.6 show the performance of AutoMoDe-`Vanilla` and evolutionary robotics in simulation and on the real robots. In all three the sessions, AutoMoDe-`Vanilla` performs better than evolutionary robotics on the real robots. For each budget, the difference in performance between AutoMoDe-`Vanilla` and evolutionary robotics is statistically significant at the 95% confidence level, according to the Friedman test.

For what concerns the comparison between simulated and real robots, the controllers designed using evolutionary robotics, while performing well in simulation, show poor performance on the real robots. This difference in performance is statistically significant according to the Friedman test. On the contrary, the controllers designed using AutoMoDe-`Vanilla` show similar performance between simulation and real robots. According to the Friedman

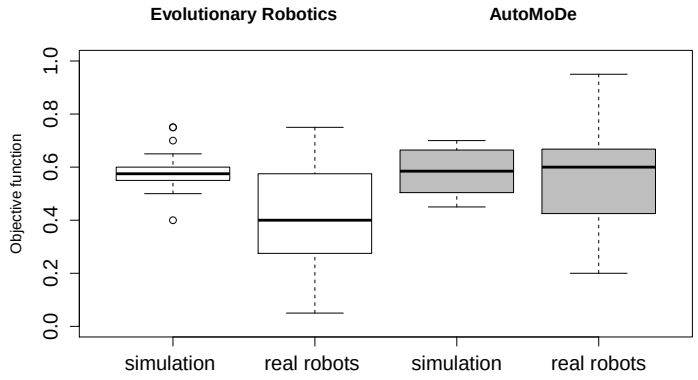


Figure 5.4: **Aggregation – Performance of the obtained controllers using 10k simulation budget.** The plot shows, for each automatic design method, the performance of the 20 controllers (one for each independent run) both in simulation and on the real robots. The white boxplots are the results of evolutionary robotics and the gray ones are the results of AutoMoDe-`Vanilla`

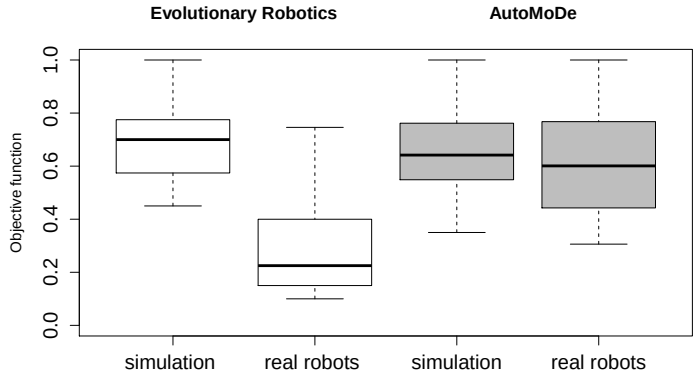


Figure 5.5: **Aggregation – Performance of the obtained controllers using 50k simulation budget.**The plot shows, for each automatic design method, the performance of the 20 controllers (one for each independent run) both in simulation and on the real robots. The white boxplots are the results of evolutionary robotics and the gray ones are the results of AutoMoDe-`Vanilla`

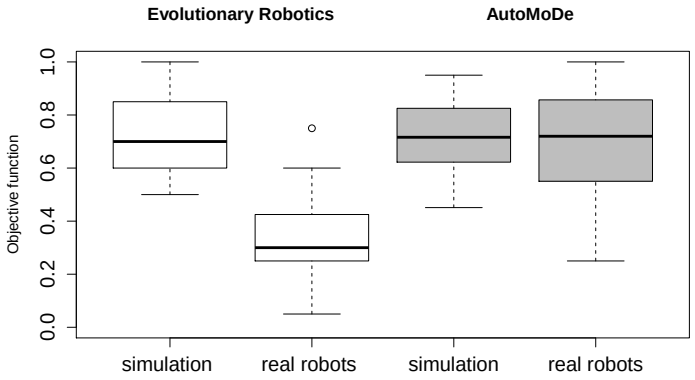


Figure 5.6: **Aggregation – Performance of the obtained controllers using 200k simulation budget.** The plot shows, for each automatic design method, the performance of the 20 controllers (one for each independent run) both in simulation and on the real robots. The white boxplots are the results of evolutionary robotics and the gray ones are the results of AutoMoDe-`Vanilla`

test, the difference in performance between simulation and real robots is not statistically significant.

Behavioral analysis

Here we describe the behaviors of the controllers designed by evolutionary robotics and AutoMoDe-`Vanilla`. As discussed in Section 2.1.2, one of the limits of evolutionary robotics is the impossibility to directly analyze the obtained controller. In fact, the numerical values obtained by evolutionary robotics have no meaning from the human point of view. The only way to analyze a behavior obtained with evolutionary robotics is to instantiate it on robots and observe their behavior. We here provide an analysis of the behavior obtained from evolutionary robotics for the aggregation task. The controllers designed by evolutionary robotics show behaviors that are qualitatively similar one to the other. 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 by the range and bearing increases. Moreover, the trajectory is perturbed by the collisions with other robots or with the walls. When a robot enters in a black area the radius of its trajectory becomes so small that the robot almost rotates on its spot. In this condition, the robot leaves the black area only because of collisions with other robots.

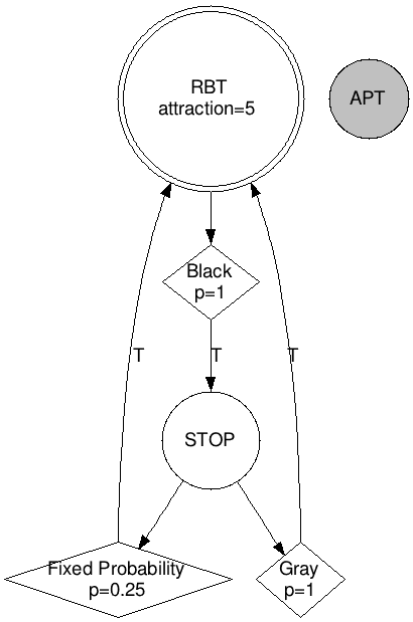
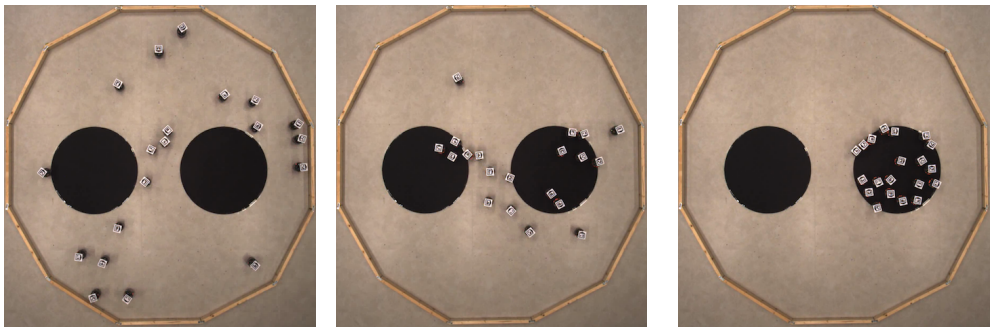


Figure 5.7: **Aggregation – A controller designed by AutoMoDe-`Vanilla`.** At the beginning the robot moves toward the other robots (state RBT e.g. attraction). When it detects the black floor it stops (state STOP). In the STOP state it checks for its conditions. It changes state when it detects the gray floor. It also starts moving, with a 0.25 probability, independently from the floor color.



(a) beginning (b) after 50 s (c) end of the run

Figure 5.8: Aggregation – Resulting behavior of a controller designed using AutoMoDe-`Vanilla`

In general, the resulting collective behavior is an aggregative behavior but it seems to be strongly dependent on some aspects of the experiment that are difficult to foreseen, in particular the collisions. In some situations, the robots are not able to get free after a collision and they get stuck outside the black areas.

A feature of AutoMoDe is that we the obtained controllers are in the form of a probabilistic state machine, which is easily readable by the human developer. We here analyze the behaviors obtained from AutoMoDe for the aggregation task. All controllers designed by AutoMoDe-`Vanilla` for the aggregation task have, with small differences, the same structure. Figure 5.7 shows a representative controller that we explain in the following. At the beginning the robot is in the state attraction (called RBT), that is, it moves toward the other robots. The robot changes state when the floor is black with a probability 1. In the state STOP, the robot does not move. The robot changes the state to RBT with a 0.25 probability or when it perceives the gray floor. The resulting collective behavior is shown in Figure 5.8. At the beginning, the robots go closer to each other. The robots that enter in the black areas stop for a while and then start again to move. Thanks to this behavior, the robots that are on the black areas result in an attraction point for the other robots. After a while (Figure 5.8b), all the robots are in the proximity of the black areas with some of them that are in the black areas. Eventually the robots outside a black area move to the black area where there are already more robots (Figure 5.8c).

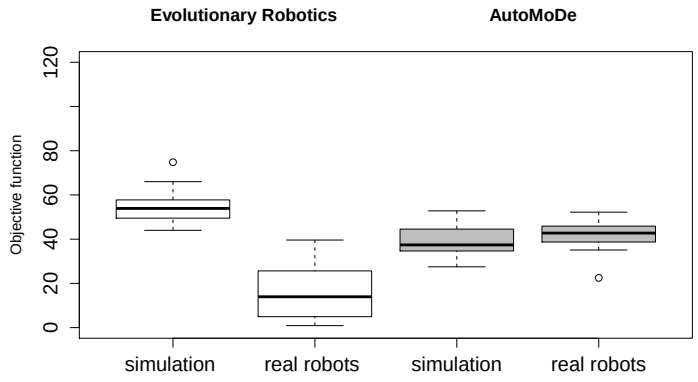


Figure 5.9: **Foraging – Performance of the obtained controllers using 10k simulation budget.** The plot shows, for each automatic design method, the performance of the 20 controllers (one for each independent run) both in simulation and on the real robots. The white boxplots are the results of evolutionary robotics, the gray ones are the results of AutoMoDe-`Vanilla`

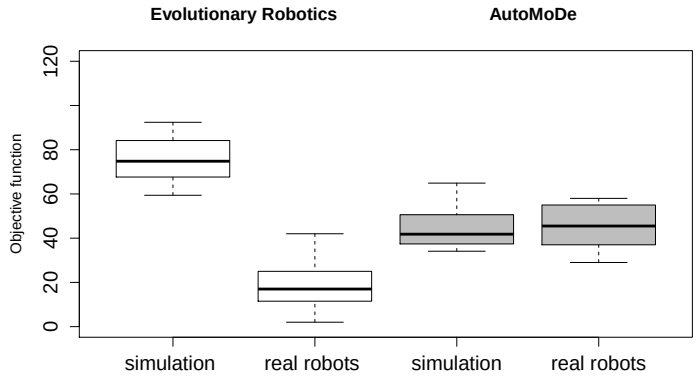


Figure 5.10: **Foraging – Performance of the obtained controllers using 50k simulation budget.** The plot shows, for each automatic design method, the performance of the 20 controllers (one for each independent run) both in simulation and on the real robots. The white boxplots are the results of evolutionary robotics, the gray ones are the results of AutoMoDe-`Vanilla`

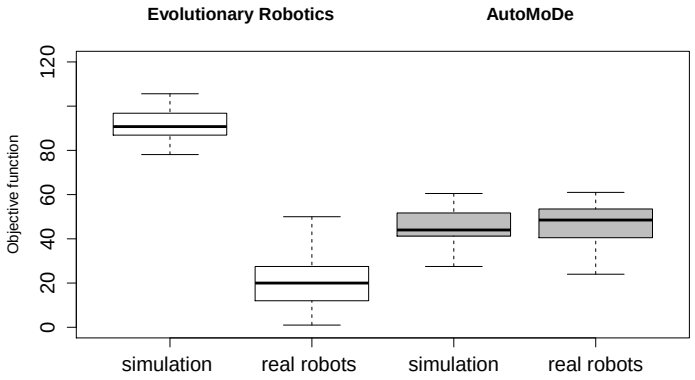


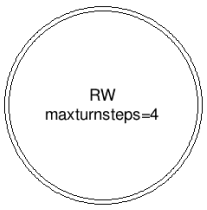
Figure 5.11: **Foraging – Performance of the obtained controllers using 200k simulation budget.** The plot shows, for each automatic design method, the performance of the 20 controllers (one for each independent run) both in simulation and on the real robots. The white boxplots are the results of evolutionary robotics, the gray ones are the results of AutoMoDe-`Vanilla`

5.2.2 Foraging

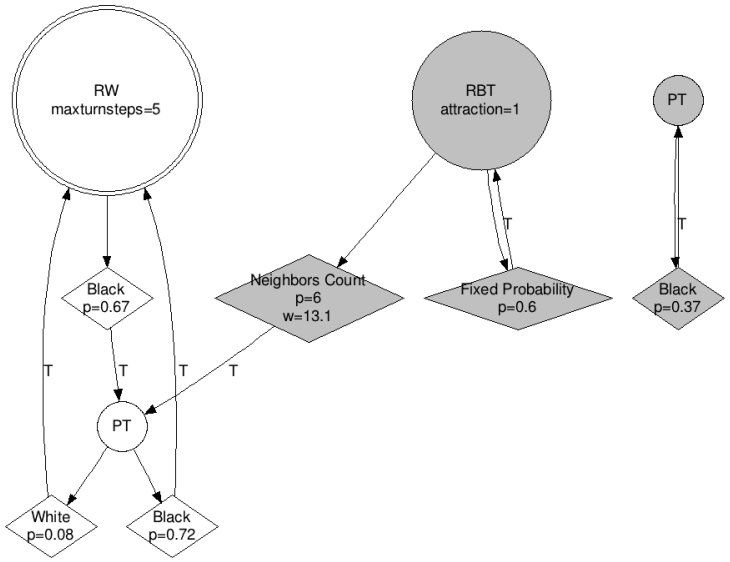
Figures 5.9, 5.10 and 5.11 show the performance achieved by AutoMoDe-`Vanilla` and evolutionary robotics in simulation and on the real robots. In all the three sessions, AutoMoDe-`Vanilla` performs significantly better than evolutionary robotics.The performance of AutoMoDe-`Vanilla` is constant independently from the used budget. Concerning the reality gap, the controllers obtained using AutoMoDe-`Vanilla` achieve the same performance in simulation and on the real robots, since the difference in performance is not statistically significant. For evolutionary robotics, there is a wide difference due to the reality gap. The magnitude of the difference in performance between simulation and real robots increases with the simulation budget. While the performance of the controllers in simulation increases with the simulation budget, the performance of the controllers on the real robots is constant. In all the three sessions, the difference in performance between simulation is statistically significant.

Behavioral analysis

The controllers obtained by evolutionary robotics show qualitatively similar behaviors. The robots explore the arena following curved trajectories. These trajectories are perturbed by the presence of other robots, the color of the floor and the intensity of the light. As result of these perturbation the robots



(a) random walk



(b) random walk / phototaxis

Figure 5.12: Foraging – The two classes of controllers designed by AutoMoDe-`Vanilla`.

The gray nodes are unreachable

follow the walls and sometimes they cross the arena. This behavior results in robots passing on the black areas (the preys) an on the white area (the nest). However, the performance of this behavior is strongly affected by the collisions among robots. Very ofter the robots get stuck and create an aggregate near the walls.

Concerning the controllers obtained by AutoMoDe-`Vanilla`, we identify two classes of controllers: random walk controllers, and random walk / phototaxis controllers. Figure 5.12 shows a representative of the random walk controllers: the robot moves randomly in the arena. Thanks to this random movement the robot enters the black and the white areas increasing the value of the objective function. This class of controllers is frequent when the lower simulation budget is used while is rare when the higher simulation budget is used. The controllers of the random walk / phototaxis class (Figure 5.12b) are based on the alternation between random walk and phototaxis. A robot uses the random walk to search for the black areas. When it finds a black area it changes the state to phototaxis to go back to the white area. When it reaches the white area it starts again to do random walk.

5.3 Summary of the results

The results just presented show that AutoMoDe-`Vanilla` is able to design performing controllers for both aggregation and foraging starting from the same atomic behaviors and conditions. The obtained controllers appear to be immune from the reality gap since they show similar performance in simulation and on the real robots. Comparing AutoMoDe-`Vanilla` with evolutionary robotics the results on the real robots show that AutoMoDe-`Vanilla` outperforms evolutionary robotics in all the experiments. Moreover, the behavioral analysis shows that the controllers designed by AutoMoDe-`Vanilla`, since are based on probabilistic finite state machines, are easy to understand. This aspect is important since it allows the designer to debug and modify manually the obtained controllers, if necessary.

Chapter 6

Conclusions and Future Works

In this report, we presented AutoMoDe a novel approach to the design and development of swarm robotics controllers. Given a task, AutoMoDe develops automatically the controllers of the robots so that the resulting collective behavior of the swarm accomplishes the desired task. In the AutoMoDe, the controllers are defined in the form of probabilistic finite state machines. These probabilistic finite state machines are obtained by a combination of preexisting modules: the atomic behaviors. In AutoMoDe, a new controller is generated through an optimization process that searches for the best combination of such atomic behaviors.

The controllers designed by AutoMoDe are naturally understandable by a human user, because they are based on probabilistic state machines. For this reason, the controllers can be analyzed and modified by hand, if necessary.

In this report we presented a proof-of-concept version of AutoMoDe called AutoMoDe-`Vanilla`. Our aim was not to define the ultimate design algorithm but to show the feasibility of AutoMoDe. For this reason, AutoMoDe-`Vanilla` can be improved in many aspects, in particular, for what concerns the representation of the probabilistic finite state machine and the optimization algorithm.

We evaluated AutoMoDe-`Vanilla` on the design of controllers for two different swarm robotics tasks: aggregation and foraging. We compared the performance of AutoMoDe-`Vanilla` with controllers obtained using evolutionary robotics on the real robots. The results showed that in all the experiments, AutoMoDe-`Vanilla` was able to design controller that outperform the ones obtained by evolutionary robotics. Moreover, the controllers obtained by AutoMoDe-`Vanilla` showed to be almost immune from the reality gap: they were deployed on the real robots without any significant performance loss.

As future work, we will improve over AutoMoDe-`Vanilla` by implement-

ing better performing optimization algorithms and more sophisticated ways to represent the probabilistic finite state machines. For what concerns the optimization algorithms, we will keep on focussing on tuning algorithms that showed to be very promising. Concerning the representation of the probabilistic finite state machines, we will analyze grammar-based representations that have been successfully applied in the field of automatic design of algorithm [32, 33].

Bibliography

[1] Christoph Adami. Digital genetics: unravelling the genetic basis of evolution. *Nature Reviews Genetics*, 7(2):109–118, 2006.

[2] Jean-Marc Amé, José Halloy, Colette Rivault, Claire Detrain, and Jean Louis Deneubourg. Collegial decision making based on social amplification leads to optimal group formation. *Proceedings of the National Academy of Sciences*, 103(15):5835–5840, 2006.

[3] Christos Ampatzis. *On the Evolution of Autonomous Time-based Decision-making and Communication in Collective Robotics*. PhD thesis, IRIDIA, Université Libre de Bruxelles, Belgium, 2008.

[4] Christos Ampatzis, Elio Tuci, Vito Trianni, and Marco Dorigo. Evolution of Signaling in a Multi-Robot System: Categorization and Communication. *Adaptive Behavior*, 16(1):5–26, 2008.

[5] Gerardo Beni. From swarm intelligence to swarm robotics. In Erol Şahin and William M. Spears, editors, *Swarm Robotics*, volume 3342 of *Lecture Notes in Computer Science*, pages 1–9. Springer Berlin Heidelberg, 2005.

[6] Spring Berman, Vijay Kumar, and Radhika Nagpal. Design of control policies for spatially inhomogeneous robot swarms with application to commercial pollination. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 378–385. IEEE, 2011.

[7] Mauro Birattari. *Tuning Metaheuristics: A Machine Learning Perspective*. Springer Verlag, 2009.

[8] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, USA, 1999.

[9] Manuele Brambilla, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41, 2013.

[10] Manuele Brambilla, Carlo Pinciroli, Mauro Birattari, and Marco Dorigo. Property-driven design for swarm robotics. In *Proceedings of 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, pages 139–146, Richland, SC, 2012. IFAAMAS.

[11] Rodney Brooks. Intelligence without representation. *Artificial intelligence*, 47:139–159, 1991.

[12] K.P. Burnham and D.R. Anderson. *Model selection and multimodel inference: a practical information-theoretic approach*. Springer, 2002.

[13] Scott Camazine, Nigel R. Franks, James Sneyd, Eric Bonabeau, Jean-Louis Deneubourg, and Guy Theraula. *Self-Organization in Biological Systems*. Princeton University Press, 2001.

[14] Alexandre Campo, Simon Garnier, Olivier Dédriche, Mouhcine Zekkri, and Marco Dorigo. Self-Organized Discrimination of Resources. *PLoS ONE*, 6(5):e19888+, May 2011.

[15] Erol Şahin. *Swarm Robotics: From Sources of Inspiration to Domains of Application*, volume 3342 of *Lecture notes in computer science*, pages 10–20. Springer Berlin / Heidelberg, 2005.

[16] Erol Şahin. Swarm robotics: From sources of inspiration to domains of application. *Swarm Robotics*, 2005.

[17] Alvaro Fialho. *Adaptive Operator Selection for Optimization*. PhD thesis, Université Paris-Sud XI, 2010.

[18] Gianpiero Francesca, Manuele Brambilla, Vito Trianni, Marco Dorigo, and Mauro Birattari. Analysing an Evolved Robotic Behaviour Using a Biological Model of Collegial Decision Making. In *From Animals to Animats 12*, 2012.

[19] Gianpiero Francesca, Manuele Brambilla, Vito Trianni, Marco Dorigo, and Mauro Birattari. Analysing an evolved robotic behaviour using a biological model of collegial decision making: Complete data, 2012. Supplementary information page at `http://iridia.ulb.ac.be/supp/IridiaSupp2012-009/`.

[20] Simon Garnier, Jacques Gautrais, Masoud Asadpour, Christian Jost, and Guy Theraulaz. Self-organized aggregation triggers collective decision making in a group of cockroach-like robots. *Adaptive Behavior*, 17(2):109–133, 2009.

[21] Veysel Gazi, Kevin M. Passino, and Senior Member. Stability Analysis of Social Foraging Swarms. *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, 34:539–557, 2004.

[22] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.

[23] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., 1989.

[24] Álvaro Gutiérrez, Alexandre Campo, Marco Dorigo, Jesus Donate, Félix Monasterio-Huelin, and Luis Magdalena. Open E-puck range & bearing miniaturized board for local communication in swarm robotics. In *ICRA'09: Proceedings of the 2009 IEEE international conference on Robotics and Automation*, pages 3111–3116, Piscataway, NJ, 2009. IEEE Press.

[25] José Halloy, Grégory Sempo, Gilles Caprari, Colette Rivault, Masoud Asadpour, Fabien Tâche, I. Said, Virginie Durier, Stéphane Canonge, Jean Marc Amé, Claire Detrain, Nikolaus Correll, Alcherio Martinoli, Francesco Mondada, Roland Siegwart, and Jean-Louis Deneubourg. Social integration of robots into groups of cockroaches to control self-organized choices. *Science*, 318(5853):1155–1158, 2007.

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

[27] John H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control and artificial intelligence*. MIT Press, 1992.

[28] Holger Hoos and Thomas Stützle. *Stochastic local search: foundations and applications*. Morgan Kaufmann, 2005.

[29] Nick Jakobi, Phil Husbands, and Inman Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. In *Advances In Artificial Life: Proc. 3Rd European Conference On Artificial Life*, pages 704–720. Springer-Verlag, 1995.

[30] S. Kazadi, J. R. Lee, and J. Lee. Model independence in swarm robotics. *International Journal of Intelligent Computing and Cybernetics, Special Issue on Swarm Robotics*, 2(4):672–694, 2009.

[31] Thomas H. Labella, Marco Dorigo, and Jean-Louis Deneubourg. Division of Labour in a Group of Robots Inspired by Ants' Foraging Behaviour. *ACM Trans. Auton. Adapt. Syst.*, 2006.

[32] Marie-Éléonore Marminon, Franco Mascia, Manuel López-Ibáñez, and Thomas Stützle. Automatic design of hybrid stochastic local search metaheuristics. In María J. Blesa, Christian Blum, Paola Festa, Andrea Roli, and Michael Sampels, editors, *Hybrid Metaheuristics*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2013.

[33] Franco Mascia, Manuel López-Ibáñez, Jérémie Dubois-Lacoste, and Thomas Stützle. From grammars to parameters: Automatic iterated greedy design for the permutation flow-shop problem with weighted tardiness. In Panos Pardalos and Giuseppe Nicosia, editors, *Learning and Intelligent Optimization*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2013.

[34] Maja Mataric and Dave Cliff. Challenges in Evolving Controllers for Physical Robots. *Robotics and Autonomous Systems*, 1996.

[35] S. Mitri, D Floreano, and L Keller. The evolution of information suppression in communicating robots with conflicting interests. *Proceedings of the National Academy of Sciences*, 106(37):15786–15790, 2009.

[36] Francesco Mondada, Michael Bonani, Xavier Raemy, James Pugh, Christopher Cianci, Adam Klaptocz, Stéphane Magnenat, Jean christophe Zufferey, Dario Floreano, and Alcherio Martinoli. The e-puck, a robot designed for education in engineering. pages 59–65. IPCB: Instituto Politécnico de Castelo Branco, 2009.

[37] Stefano Nolfi and Dario Floreano. *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines.* MIT Press, Cambridge, MA, 2000.

[38] Leonid Peshkin. *Reinforcement Learning by Policy Search PhD thesis MS.* W eizmann Institute of Science, 1995.

[39] Carlo Pinciroli, Vito Trianni, Rehan O'Grady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni Di Caro, Frederick Ducatelle, Timothy Stirling, Álvaro Gutiérrez, Luca Maria Gambardella, and Marco Dorigo. ARGoS: a modular, multi-engine simulator for heterogeneous swarm robotics. In *Proceedings of*

the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2011), pages 5027–5034. IEEE Computer Society Press, Los Alamitos, CA, September 2011.

[40] Giovanni Pini, A Brutschy, M Frison, A Roli, Marco Dorigo, and Mauro Birattari. Task partitioning in swarms of robots: An adaptive method for strategy selection. *Swarm Intelligence*, 2011.

[41] Michael O Rabin. Probabilistic automata. *Information and Control*, 1963.

[42] Vito Trianni. *Evolutionary swarm robotics*. Springer Verlag, 2008.

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

[44] Markus Waibel, Laurent Keller, and Dario Floreano. Genetic team composition and level of selection in the evolution of cooperation. *Evolutionary Computation, IEEE Transactions on*, 13(3):648–660, 2009.

[45] James M Whitacre, Tuan Q Pham, and Ruhul A Sarker. Credit Assignment in Adaptive Evolutionary Algorithms. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, GECCO '06, 2006.

[46] Alan Winfield. Towards an engineering science of robot foraging. In *Distributed Autonomous Robotic Systems 8*. Springer Berlin Heidelberg, 2009.

[47] Steffen Wischmann, Dario Floreano, and Laurent Keller. Historical contingency affects signaling strategies and competitive abilities in evolving populations of simulated robots. *Proceedings of the National Academy of Sciences*, 109(3):864–868, 2012.