ISCTE IUL

University Institute of Lisbon

Department of Information Science and Technology

Engineering Evolutionary Control for Real-world Robotic Systems

Miguel António Frade Duarte

A Thesis presented in partial fulfillment of the Requirements for the Degree of
Doctor in Information Science and Technology

Supervisor
Prof. Dr. Anders Lyhne Christensen, Assistant Professor
ISCTE-IUL
Co-Supervisor
Prof. Dr. Sancho Moura Oliveira, Assistant Professor
ISCTE-IUL

April 2016

University Institute of Lisbon

Department of Information Science and Technology

Engineering Evolutionary Control for Real-world Robotic Systems

Miguel António Frade Duarte

A Thesis presented in partial fulfillment of the Requirements for the Degree of
Doctor in Information Science and Technology

April 2016

Resumo

A Robótica Evolutiva (RE) é a área de investigação que estuda a aplicação de computação evolutiva na conceção de sistemas robóticos. Dois principais desafios têm impedido a aplicação da RE em tarefas do mundo real: a dificuldade em solucionar tarefas complexas e a transferência de controladores evoluídos para sistemas robóticos reais. Encontrar soluções para tarefas complexas é desafiante para as técnicas evolutivas devido ao *bootstrap problem* e à *deception*. Quando o objetivo é demasiado difícil, o processo evolutivo tende a permanecer em regiões do espaço de procura com níveis de desempenho igualmente baixos, e consequentemente não consegue inicializar. Por outro lado, o espaço de procura tende a enrugar à medida que a complexidade da tarefa aumenta, o que pode resultar numa convergência prematura. Outro desafio na RE é a *reality gap*. O controlo robótico é tipicamente evoluído em simulação, e só é transferido para o sistema robótico real quando uma boa solução tiver sido encontrada. Como a simulação é uma abstração da realidade, a precisão do modelo do robô e das suas interações com o ambiente é limitada, podendo resultar em controladores com um menor desempenho no mundo real.

Nesta tese, apresentamos uma abordagem de síntese de controlo hierárquica que permite o uso de técnicas de RE em tarefas complexas com hardware robótico real, mitigando o *bootstrap problem*, a *deception* e a *reality gap*. Decompomos recursivamente uma tarefa em sub-tarefas, e sintetizamos controlo para cada sub-tarefa. Os comportamentos individuais são então compostos hierarquicamente. A possibilidade de transferir o controlo incrementalmente à medida que o controlador é composto permite que problemas de transferibilidade possam ser endereçados localmente na hierarquia do controlador. A nossa abordagem permite o uso de diferentes técnicas de síntese de controlo, resultando em controladores híbridos. Demonstramos a nossa abordagem em várias tarefas que vão para além da complexidade das tarefas onde a RE foi aplicada. Também mostramos que o controlo hierárquico pode ser aplicado em sistemas de um robô ou sistemas multi-robô. Dado o nosso objetivo de longo prazo de permitir o uso de técnicas de RE em tarefas no mundo real, concebemos e desenvolvemos uma plataforma de robótica de enxame, e mostramos a primeira transferência de controlo evoluído e hierárquico para um exame de robôs fora de condições controladas de laboratório.

Palavras-chave: Robótica evolutiva, sistemas de controlo hierárquicos, controlo híbrido, robótica de enxame, sistemas multi-robô, redes neuronais artificiais.

Abstract

Evolutionary Robotics (ER) is the field of study concerned with the application of evolutionary computation to the design of robotic systems. Two main issues have prevented ER from being applied to real-world tasks, namely scaling to complex tasks and the transfer of control to real-robot systems. Finding solutions to complex tasks is challenging for evolutionary approaches due to the *bootstrap problem* and *deception*. When the task goal is too difficult, the evolutionary process will drift in regions of the search space with equally low levels of performance and therefore fail to bootstrap. Furthermore, the search space tends to get rugged (deceptive) as task complexity increases, which can lead to premature convergence. Another prominent issue in ER is *the reality gap*. Behavioral control is typically evolved in simulation and then only transferred to the real robotic hardware when a good solution has been found. Since simulation is an abstraction of the real world, the accuracy of the robot model and its interactions with the environment is limited. As a result, control evolved in a simulator tends to display a lower performance in reality than in simulation.

In this thesis, we present a hierarchical control synthesis approach that enables the use of ER techniques for complex tasks in real robotic hardware by mitigating the bootstrap problem, deception, and the reality gap. We recursively decompose a task into sub-tasks, and synthesize control for each sub-task. The individual behaviors are then composed hierarchically. The possibility of incrementally transferring control as the controller is composed allows transferability issues to be addressed locally in the controller hierarchy. Our approach features hybridity, allowing different control synthesis techniques to be combined. We demonstrate our approach in a series of tasks that go beyond the complexity of tasks where ER has been successfully applied. We further show that hierarchical control can be applied in single-robot systems and in multirobot systems. Given our long-term goal of enabling the application of ER techniques to real-world tasks, we systematically validate our approach in real robotic hardware. For one of the demonstrations in this thesis, we have designed and built a swarm robotic platform, and we show the first successful transfer of evolved and hierarchical control to a swarm of robots outside of controlled laboratory conditions.

Keywords: Evolutionary robotics, hierarchical control systems, hybrid control, swarm robotics, multirobot systems, artificial neural networks.

Acknowledgements

First of all, I would like to acknowledge my supervisors, Anders Christensen and Sancho Oliveira. While the journey toward a Ph.D. is very challenging, having them as my supervisors made it enjoyable, fun, and rewarding. I will never forget their friendship, dedication, scientific integrity, commitment to excellence, and how they transmit these values to their students. I don't think it would have been possible to have two better supervisors, and I thank them for their mentorship.

Being part of the BioMachines Lab from the very beginning and watching it grow to what it is today has been really exciting. I've had the opportunity to work with some amazing people over the years that have contributed to my scientific and personal growth. I would like to acknowledge Carlos Duque, Piotr Szczawiński, Hao Zong, Amr Hamouda, Gustavo Martins, Inês Mamede, Rita Ramos, and Pedro Romano. Special thanks go out to Vasco Costa, without whom I would not have been able to develop our aquatic robots, and who taught me so much about electronics, hardware and rock climbing, Tiago Rodrigues, my daily research colleague for over two years, and Fernando Silva and Jorge Gomes, for all the interesting and stimulating discussions, and their willingness to collaborate and share knowledge.

The Vitruvius FabLab crew, João Sousa, Maria João de Oliveira, Bárbara Varela, and Alexandra Paio, have been essential during my research. FabLab was the place where I learned the ins and outs of digital (and classical!) fabrication, and where we found many of the tools and much of the knowledge that allowed a few software guys to develop a robotics platform from scratch.

I would like to acknowledge my fellow IEEE volunteers. When we started our local IEEE Student Branch at ISCTE, I never imagined the amazing things that we would end up accomplishing: from participating in robotics competitions, to kickstarting numerous technical workshops, organizing huge events, and taking the best of what is done here to many national and international venues. It was fantastic to be able to learn from and collaborate with so many talented and driven individuals.

I would also like to leave a word of appreciation to the administrative staff at ISCTE and IT that made many of the unavoidable bureaucracies so much easier to handle: Fátima Silva, Marisa Manteigas, and Alice Espada from ISTA, Fátima Estevens from ISTAR, Sónia Viveiros from GAI, and Sara Correia, Tereza Traquinas and Ana Rodrigues from IT.

I thank all my friends, who have always encouraged me, particularly Fábio Francisco, Ricardo Gomes, Gabriel Veiga, João Machado, Carlos Lima, and David Jardim.

I would like to express my gratitude to my parents, António and Isabel, and my sister, Mariana, for the unwavering support and love, and who always pushed me to follow my passions. Who I am today and what I have accomplished is only a consequence of all they have given and taught me.

Finally, I thank Margarida, who has been my pillar of strength, and whose love and emotional support have given me the motivation to follow through with this adventure.

This work has been supported by the Portuguese Foundation for Science and Technology (Fundação para a Ciência e Tecnologia) under the grants SFRH/BD/76438/2011, EXPL/EEI-AUT/0329/2013, and by Instituto de Telecomunicações under the grant UID/EEA/50008/2013.

Contents

List of Figures

Acronyms

AI	Artificial Intelligence
ANN	Artificial Neural Network
AUV	Autonomous Underwater Vehicle
CPPN	Compositional Pattern Producing Networks
CTRNN	Continuous-time Recurrent Neural Network
EA	Evolutionary Algorithm
ER	Evolutionary Robotics
ESP	Enforced Sub-Populations
FSM	Finite State-machine
HyperNEAT	Hypercube-based NeuroEvolution of Augmenting Topologies
NS	Novelty Search
SRS	Swarm Robotic Systems

Chapter 1

Introduction

The first records of complex, moving machines, or *automata*, date back to the Hellenistic scientific period (323 BC to 31 BC), where inventors and scientists such as Archytas, Ctesibius and Heron designed and built moving mechanical devices (Bedini, 1964; Rossi et al., 2009). The interest in such machines has increased as technology advanced, and the concept of automata has morphed into what we now call *robots*. A robot is, according to the Oxford English Dictionary, "a machine capable of carrying out a complex series of actions automatically, especially one programmable by a computer". Advanced robots were suddenly made possible with the computer revolution of the mid 20th century, and research in this area grew rapidly. Nowadays, robots can be found in almost every sector of society (Bekey and Yuh, 2008), such as industrial robots that can operate continuously in factories, medical robots that assist doctors in complicated surgeries, domestic robots that clean up our home's floor, and consumer-grade robots that can be controlled using our phones.

The apparent ubiquity of smart robots, however, is deceiving. While current robots are extremely technologically advanced machines, they are not very intelligent. Advances in Artificial Intelligence (AI) have consistently failed at achieving the so-called "strong AI", a general type of intelligence intellectually equivalent to human intelligence. The development of strong AI robots would allow them to be applied to whole new classes of tasks in our complex, dynamic world, and be actively involved in our daily lives. While robots are very good at carrying out precise instructions, they are unable to adapt to unforeseen situations or operate in arbitrary real-world task environments.

Most robots are manually programmed by engineers to perform a particular task. In many cases, the algorithms that govern the behavior of the robots are not flexible, and the robot's behavior is limited to specific manually programmed rules. Many different types of "narrow AIs", extremely specialized decision systems, have been developed for robotic applications. While these control systems work in very specific situations, such as classifying objects from a video feed or recognizing speech, the ability to solve many everyday tasks in general is still lacking.

In order to overcome this inflexibility in traditional robotic control systems, alternative control synthesis methodologies have emerged. Evolutionary Robotics (ER) is a research field inspired by Darwin's theory of evolution (Darwin, 1859). It employs artificial evolution with the aim of automatically synthesizing robotic control (Nolfi and Floreano, 2000), and sometimes even the robots' morphology (Hiller and Lipson, 2012). The robot's control system is typically encoded in a genotype, which is then subject to an iterative evaluation, selection and variation process. ER techniques have the potential to automate the design of control systems without the need for manual and detailed specification of the desired behavior (Floreano and Keller, 2010) and to exploit the way in which the world is perceived through the robot's (often limited) sensors. Numerous studies have demonstrated evolved control systems that enable robots to solve simple tasks in surprisingly elegant ways (Floreano and Mondada, 1996; Nolfi and Floreano, 2000; Nakamura et al., 2000; Hornby et al., 2005; Christensen and Dorigo, 2006a; Duarte et al., 2011).

ER techniques have been applied to single-robot systems and to multirobot systems. Swarm robotics is a promising approach to collective robotics, where large groups of relatively simple robots with the ability to display collectively intelligent behavior are used, as opposed to relying on a single, complex robot (Brambilla et al., 2013). Inspiration for Swarm Robotic Systems (SRS) comes from biological swarms (Şahin, 2005). Colonies of social insects, such as ants, termites or bees, display certain interesting properties: while each individual is relatively simple, the swarm as a whole is capable of solving complex tasks, in a classic example of the collective being more than the sum of its parts.

Control in a SRS is decentralized, meaning that each individual robot operates based on its local observations of the environment, and interaction with neighboring robots. During task execution, the swarm-level behavior emerges from the interactions between neighboring robots, and from the interactions between robots and the environment. The swarm robotics approach is associated with important properties in multirobot systems (Şahin, 2005), namely: (i) robustness, the ability to cope with the faults and loss of individual robots, and (ii) scalability, the ability to operate under a wide range of group sizes. Due to these properties, swarm robotics systems have an enormous potential in several real-world domains, such as search and rescue, exploration, surveillance, and cleaning (Bayındır and Şahin, 2007; Brambilla et al., 2013).

One of the key challenges in SRS is the synthesis of behavioral control for the constituent robots. Manually designing control for each robot requires the decomposition of the macroscopic, swarm-level behavior into microscopic behavioral rules so that the global, collectively self-organized behavior emerges (Brambilla et al., 2013). That is, the designer needs to be able to understand the relation between robot interactions and emerging global properties. There is, however, no known way to derive the microscopic behavioral rules based on a desired global behavior or task description for the general case (Dorigo et al., 2004). In this respect, the use of evolutionary computation techniques has been studied as an alternative to traditional control approaches such as manual programming, see (Nolfi and Floreano, 2000; Harvey et al., 1997; Lipson and Pollack, 2000) for examples.

Despite its potential, a number of critical issues currently prevent ER from becoming a viable mainstream approach for engineers (Silva et al., 2015a). Even though numerous studies have applied evolution for the synthesis of control systems, the approach has so far failed to scale to more complex tasks (Nelson et al., 2009). If a task is too difficult, an initial randomly generated population may be located in a region of the search space without a fitness gradient. The evolutionary process may therefore drift around in such a region and fail to *bootstrap*. Even when a gradient is present, the gradient may lead the evolutionary process toward low-quality local optima, a problem known as *deception* (Whitley, 1991). As the complexity of a task increases, the fitness landscape typically becomes rugged (Nelson et al., 2009), and the evolutionary process becomes more vulnerable to deception (Lehman and Stanley, 2011).

Another issue is related to the transfer of evolved control from simulation to reality. In ER, a large number of candidate solutions typically have to be evaluated. As a consequence, evaluations are usually conducted in computer simulation and not on real robotic hardware. Despite best efforts to accurately simulate the real world, differences are bound to exist between simulation and reality. The differences between simulation and reality are often referred to as *the reality gap* (Jakobi, 1997). The presence of the reality gap means that controllers evolved in simulation may exploit aspects of the simulated world that are different or may not exist in the real world. Controllers evolved in simulation are therefore not guaranteed to maintain their performance when executed on real robotic hardware (Koos et al., 2013).

The bootstrap problem, deception, and the reality gap remain major challenges in ER. While there has been considerable progress in the field of ER in recent years, there have been no significant breakthroughs indicating that ER scales to tasks with the level of complexity found outside strictly controlled laboratory conditions. As discussed by Nelson et al. (2009) in a review of single-robot ER tasks and fitness functions, and Brambilla et al. (2013) and Bayındır (2016) in recent reviews of the field of swarm robotics, all experiments presented in the literature covered have, in fact, been performed either in simulation or in controlled environments, such as enclosed arenas, where the relevant conditions are defined by the experimenter. That is, although ER-based control is ultimately intended to be applied in potentially unstructured, real-world environments, no study has been able to truly leverage and demonstrate the benefits of ER in such environments. Notwithstanding, evolutionary techniques still have significant potential in controller design that can be realized if we depart from the tradition unadulterated application of ER and embrace more practical and engineering-oriented approaches as the work presented in this thesis demonstrates.

In this thesis, we propose an approach for decomposable complex tasks that combines the benefits of ER, namely automatic synthesis of control, and human engineering to circumvent the bootstrap problem, to avoid deception, and to successfully cross the reality gap. If a robotic controller cannot be evolved for a particular task, we manually divide the task into two or more sub-tasks and evolve an independent sub-controller for each sub-task. An additional controller that selects which sub-controller is active at any given time is then synthesized. Behavioral control for complex tasks can thus be obtained in an incremental and hierarchical manner, and issues related to performance on real hardware can be addressed at each increment. Our approach allows for the reuse of previously synthesized controllers and for the combination of different control synthesis techniques. Some tasks, such as those that require the robot to perform actions with a high degree of accuracy, might be difficult to simulate with sufficient fidelity to allow for successful transfer of evolved control to a real robot. Examples include object grasping and manipulation (Okamura et al., 2000), morphogenesis (O'Grady et al., 2009), where robots attach to each other to form specific shapes, and fine sensorimotor coordination (Er et al., 2002; Hehn and D'Andrea, 2011), where accurate sensing, actuation and control is necessary. For such tasks, a manually programmed behavior can be developed directly for the real robotic hardware and integrated in the hierarchical structure of the controller.

Our approach introduces four important features related to the use of ER techniques as an engineering tool: (i) *incremental evolution*: by taking an incremental, divide-and-conquer approach to evolution, bootstrapping issues and deception can be avoided, (ii) *scalability (in task complexity)*: as partial solutions are combined, fitness functions can be derived based on the immediate task decomposition, and an increase in fitness function complexity as increasingly complex tasks are considered is thereby averted, (iii) *incremental transfer*: sub-controllers can be tested incrementally on real robotic hardware and issues related to real-robot performance can be addressed locally in the controller hierarchy, and (iv) *hybridity*: our methodology allows for seamless integration of behavior synthesized with different approaches and even manually programmed behavior.

In summary, the contribution of this thesis is as follows. We present the hierarchical control synthesis approach, which allows evolutionary methodologies to be successfully applied to complex tasks by decomposing the robotic control into a hierarchy of behaviors. We systematically validate our approach in real robotic hardware and in tasks that are beyond the state of the art in terms of task complexity. We go on to demonstrate the first instance of a robotic swarm with evolved control performing tasks outside of strictly controlled laboratory conditions, showing that SRS, ER and hierarchical control systems are viable approaches for complex real-world tasks.

1.1 Problem Statement

The focus of this thesis is to address the main issues in the field of ER that have prevented the application of evolutionary techniques to real-world robotic systems, through the proposal and study of an engineering-centric hierarchical control synthesis approach. The approach consists of decomposing a task into several sub-tasks, and then synthesizing control for each sub-task separately. The behavioral blocks, which can be synthesized using different techniques, such as artificial evolution and manual programming, are combined hierarchically, allowing for increasingly complex tasks to be solved.

The hierarchical control synthesis approach was designed in order to address multiple issues in the field of ER: (i) the bootstrap problem and deception are avoided, since each individual controller only solves part of the task, (ii) the reality gap effect is minimized by adding significant amounts of noise in simulation (Miglino et al., 1996), thus promoting the evolution of general and robust controllers, and (iii) hybridity in the controller structure allows for the synthesis of control for tasks where the application of evolutionary techniques would not feasible, such as in tasks that require fine sensorimotor coordination.

In this thesis, we evaluate the hierarchical control synthesis approach in several different scenarios. The approach is first validated in a single-robot system using different variations of the approach, such as purely-evolved hierarchical controllers and hybrid controllers. After the first validation, we apply the approach to SRS, where it becomes infeasible to manually program robotic controllers for general tasks. We validate our approach by conducting systematic real-robot experiments, including the first demonstration of a SRS with evolved control outside of strictly controlled laboratory conditions.

1.2 Thesis Structure and Contribution of Research

In this section, we provide an overview of the thesis structure and the scientific publications produced over the past four years leading to this thesis.

In Chapter 2, we review the current state of the art in the field of evolutionary robotics and swarm robotics. The chapter is divided in four major sections: an introduction of the field of ER, the challenges in transferring control to real robots, the challenges in evolving control for complex tasks, and an overview of the swarm robotics field. Part of this section was presented in a survey of the key open issues in the field of ER:

- F. Silva, M. Duarte, L. Correia, S. M. Oliveira, A. L. Christensen, "**Open Issues in Evolutionary Robotics**", *Evolutionary Computation*, 24(2):1-32, 2016.

In Chapter 3, we present the hierarchical control synthesis methodology proposed in this thesis. In our methodology, a task is decomposed into sub-tasks, and robotic control is synthesized for each sub-task. The different sub-controllers are then composed hierarchically, allowing more complex tasks to be solved. Actuation nodes are at the lower levels of such a controller, and are called *behavior primitives*, while decision nodes, called *behavior arbitrators*, are higher up the hierarchy and select which sub-controller should be active at any given time. A controller can be hybrid, which means it can be composed of control nodes synthesized with different techniques, such as evolution and manual programming.

In Chapter 4, we validate our hierarchical control synthesis technique in a series of experiments with an e-puck robot (Mondada et al., 2009). Firstly, we demonstrate a hierarchical controller for a complex, sequential task in which all modules are evolved for a rescue task (see Section 4.2). Secondly, we explore the concept of hybrid controllers, where control is composed of both evolved and manually programmed control, in a navigation sub-task of the complete rescue task (see Section 4.3). Finally, we demonstrate a hybrid controller for a complex, non-sequential task where the need for fine sensorimotor control can prevent the use of evolutionary techniques in real robots (see Section 4.4). The work presented in Chapter 4 resulted in the following publications:

- M. Duarte, S. M. Oliveira and A. L. Christensen, "**Evolution of Hybrid Robotic Controllers for Complex Tasks**", *Journal of Intelligent and Robotic Systems*, 78(3-4):463-484, 2015.
- M. Duarte, S. Oliveira and A. L. Christensen, "**Automatic Synthesis of Controllers for Real Robots Based on manually programmed Behaviors**", *Proceedings of the International Conference on Adaptive Behaviour*, Springer, Berlin, Germany, 2012, pp. 249-258.
- M. Duarte, S. Oliveira and A. L. Christensen, "**Hierarchical evolution of robotic controllers for complex tasks**", *in Proceedings of the IEEE International Conference on Development and Learning and on Epigenetic Robotics (ICDL EpiRob)*, IEEE Press, Piscataway, NJ, 2012, pp. 1-6. **Paper of Excellence award**

- M. Duarte, S. Oliveira and A. L. Christensen, "**Structured Composition of Evolved Robotic Controllers**", *in Proceedings of the 5th International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS)*, N. Siebel, Ed., 2012, pp. 19-25.

Although they are not presented in this thesis, the experiments reported in Section 4.4 were extended to a simulated multirobot-system and resulted in the following publication:

- M. Duarte, S. M. Oliveira and A. L. Christensen, "**Evolution of Hierarchical Controllers for Multirobot Systems**", *in Proceedings of the International Conference on the Synthesis & Simulation of Living Systems (ALIFE)*, MIT Press, Cambridge, MA, 2014, pp. 657-664.

Preliminary versions of the experiments presented in Sections 4.2 and 4.3 were presented in my MSc thesis (see reference below). The MSc research overlapped with the first 6 months of doctoral research (starting in January 2012), and resulted in the compilation of the MSc thesis in June 2012. Both the simulated and real-robot experiments were later redone based on the feedback from reviewers' comments during the submission to the abovementioned article to Journal of Intelligent and Robotic Systems.

- M. Duarte, "**Hierarchical Evolution of Robotic Controllers for Complex Tasks**", Master's thesis, University Institute of Lisbon (ISCTE-IUL), 2012.

In Chapter 5, we apply our methodology to robotic swarms. The experiments presented in this chapter were conducted in the context of the CORATAM and HANCAD projects, which focused on swarms of aquatic robots. The robotic platform used was designed and built during the projects, and a total of 10 robots were produced. The first part of the chapter focuses on describing the robotic platform (see Section 5.1.1) and the simulation model (see Section 5.1.2). Then, we present a series of validation experiments based on four canonical swarm robotics tasks (homing, clustering, dispersion and area monitoring, see Section 5.1.3) and the results of the evolutionary process in simulation (see Section 5.1.4). The highest-performing controllers for these tasks are systematically tested in the real robotic platform (see Chapter 5.2), and we conduct an additional set of experiments to validate key swarm robotics characteristics in our platform, such as scalability and robustness (see Section 5.3). To finish the validation experiments, we demonstrate the modularity of the evolved controllers by applying our hierarchical control synthesis approach with a simple time-based manually programmed behavior arbitrator for an environmental monitoring task (see Section 5.4). The work presented in the aforementioned sections has been accepted at the international conference OCEANS and submitted to the journal PLoS ONE:

- M. Duarte, V. Costa, J. Gomes, T. Rodrigues, F. Silva, S. M. Oliveira, A. L. Christensen, "**Evolution of Collective Behaviors for a Real Swarm of Aquatic Surface Robots**", *PLoS ONE*, 11(3):e0151834, 2016.
- M. Duarte, J. Gomes, V. Costa, T. Rodrigues, F. Silva, V. Lobo, M. M. Marques, S. M. Oliveira and A. L. Christensen, "**Application of Swarm Robotic Systems for Marine Environmental Monitoring**", *in Proceedings of the MTS-IEEE OCEANS*, 2016, in press.

In Section 5.5, we apply our hierarchical control methodology to an intruder detection task. In our intruder detection task, a swarm of robots must remain within a previously designated area, delimited by a geo-fence, and pursue intruders that try to cross the area. The robots are initially located on a base station, to which they must periodically return in order to recharge their batteries. The homing and area monitoring controllers previously evolved in Section 5.1.3 are reused for this task, in addition to a newly evolved pursue intruder controller. These behaviors are then combined with a behavior arbitrator consisting of a manually programmed finite state-machine. We further test the scalability and flexibility of the behavior arbitrator, and finally transfer control from simulation to the real swarm of aquatic robots. The experiments presented in Section 5.5 have been published in the following paper:

- M. Duarte, J. Gomes, V. Costa, S. M. Oliveira and A. L. Christensen, "**Hybrid Control for a Real Swarm Robotic System in an Intruder Detection Task**", *in Proceedings of the 19th European Conference on the Applications of Evolutionary Computation (EvoStar)*, 2016, pp. 213-230.

We further extend the intruder detection experiments in order to test the scalability of the controllers in swarms of up to 1000 robots in a monitoring area with an area of 10 km^2 (see Section 5.5.5). A preliminary version of the scalability experiments was presented in:

- M. Duarte, S. M. Oliveira and A. L. Christensen, "**Hybrid Control for Large Swarms of Aquatic Drones**", *in Proceedings of the International Conference on the Synthesis & Simulation of Living Systems (ALIFE)*, MIT Press, Cambridge, MA, 2014, pp. 785-792.

In Chapter 6, we conclude the thesis and discuss future directions of research. All experiments presented in this thesis (except for the scalability experiments with up to 1000 robots in Section 5.5.5) were validated in real robotic hardware.

1.3 Other Scientific Contributions

During our research, we have conducted a series of studies not directly related to the topic of this thesis. These studies are related with traditional evolutionary robotics, and evolutionary techniques in the context of swarm robotics. These contributions have resulted in one book chapter and eight conference publications:

- T. Rodrigues, M. Duarte, M. Figueiró, V. Costa, S. M. Oliveira, A. L. Christensen, "**Overcoming Limited Onboard Sensing in Swarm Robotics**

through Local Communication", in *Transactions on Computational Collective Intelligence XX*, vol. 9420 of Lecture Notes in Computer Science (LNCS), pp. 201-223. Springer, Berlin, Germany, 2015.

- M. Duarte, F. Silva, . T. Rodrigues, S. M. Oliveira, A. L. Christensen, "**JBotEvolver: A Versatile Simulation Platform for Evolutionary Robotics**", *in Proceedings of the International Conference on the Synthesis & Simulation of Living Systems (ALIFE)*, MIT Press, Cambridge, MA, 2014, pp. 210-211.

- F. Silva, M. Duarte, S. M. Oliveira, L. Correia, A. L. Christensen, "**The case for engineering the evolution of robot controllers**", *in Proceedings of the International Conference on the Synthesis & Simulation of Living Systems (ALIFE)*, MIT Press, Cambridge, MA, 2014, pp. 703-710.

- T. Rodrigues, M. Duarte, S. M. Oliveira, A. L. Christensen, "**What you choose to see is what you get: an experiment with learnt sensory modulation in a robotic foraging task**", *in Proceedings of the 16th European Conference on the Applications of Evolutionary Computation (EvoStar)*, Springer, Berlin, Germany, 2014, pp. 789-801.

- T. Rodrigues, M. Duarte, S. M. Oliveira, A. L. Christensen, "**Beyond Onboard Sensors in Robotic Swarms: Local Collective Sensing through Situated Communication**", *in Proceedings of the International Conference on Agents and Artificial Intelligence (ICAART)*, SCITEPRESS, Lisbon, Portugal, 2015, pp. 111-118.

- A. L. Christensen, S. Oliveira, O. Postolache, M. J. d. Oliveira, S. Sargento, P. Santana, L. Nunes, F. Velez, P. Sebastião, V. Costa, M. Duarte, J. Gomes, T. Rodrigues, F. Silva, "**Design of Communication and Control for Swarms of Aquatic Surface Drones**", *in Proceedings of the International Conference on Agents and Artificial Intelligence (ICAART)*, SCITEPRESS, Lisbon, Portugal, 2015, pp. 548-555.

- P. Romano, L. Nunes, A. L. Christensen, M. Duarte, S. M. Oliveira, "**Genome Variations: Effects on the robustness of neuroevolved swarm controllers**", *in Proceedings of the Iberian Conference on Robotics (ROBOT)*, Springer, Berlin, Germany, 2015, pp. 309-319.

- F. J. Velez, A. Nadziejko, A. L. Christensen, S. Oliveira, T. Rodrigues, V. Costa, M. Duarte, F. Silva, J. Gomes, "**Wireless Sensor and Networking Technologies for Swarms of Aquatic Surface Drones**", *in Proceedings of the IEEE 82nd Vehicular Technology Conference (VTC Fall)*, 2015, pp. 1-2.

- V. Costa, M. Duarte, T. Rodrigues, S. M. Oliveira and A. L. Christensen, "**Design and Development of an Inexpensive Aquatic Swarm Robotics System**", *in Proceedings of the MTS-IEEE OCEANS*, 2016, in press.

A detailed description of other contributions, such as the participation in the CORATAM and HANCAD projects, the participation in the COHiTEC program, media coverage, and software tools developed can be found in Appendix A.

1.4 Summary

In this chapter, we have provided an overview of the key advantages and limitations in the field of ER. While evolutionary techniques have a significant potential for the automatic synthesis of robotic control, they are currently limited in terms of complexity of the evolved behavior, and of the transfer of control from simulation to reality. In order to address these issues, we introduced the topic of this thesis, namely the hierarchical control synthesis approach. We listed scientific contribution which lead to this thesis, as well as other scientific contributions related to ER and SRS.

Chapter 2

State of the Art

Evolutionary robotics is the research field that studies the application of evolutionary computation to the synthesis of robotic control (Nolfi and Floreano, 2000). Given a specification of the task by the experimenter, an evolutionary algorithm evaluates and optimizes controllers in a holistic manner, iteratively fine-tuning the parameters and the microscopic rules guiding the robots based on the performance of the resulting behavior. Evolution therefore eliminates the need for manual and detailed specification of low-level control (Floreano and Keller, 2010) and facilitates the emergence of self-organized behavior (Nolfi, 1998; Trianni and Nolfi, 2011).

Evolutionary Algorithms (EAs) are a search heuristic inspired by Darwinian evolution (Darwin, 1859) that rely on blind variations and survival of the fittest to find increasingly better solutions for a particular problem. EAs are composed of a population of candidate solutions, which are randomly selected from the complete search space in the first stage of evolution. The performance of each candidate solution is assessed, and the highest-performing solutions are then selected and subject to variations (mutation and/or crossover). The process then continues iteratively until a termination condition is reached, such as a specific number of generations or a performance threshold.

Each candidate solution is represented by a genome. An encoding mechanism is then used to transcribe a particular *genotype*, which describes the solution's hereditary information, to the corresponding *phenotype*, which represents the solution itself. In the case of ER, the phenotype is typically the robot's controller. Genotype-phenotype mappings can be direct, where each value of the genotype directly translates to a feature of the phenotype (Nelson et al., 2009), or indirect, where each gene can be used multiple times to construct different parts of the phenotype (Stanley and Miikkulainen, 2003).

In ER, it is common to use Artificial Neural Networks (ANNs) as robotic controllers. ANNs are computational models based on biological neural networks. They are composed of neurons and inter-neuron connections, also known as *synapses*. Neurons are computational units, which integrate a number of inputs and compute the corresponding output. ANNs are composed by an input and an output layer, and can have any number of hidden layers, although certain networks are not directly organized in distinct layers (Stanley and Miikkulainen, 2002).

By using an evolutionary process, the parameters of the neural network (such as the number of neurons, the synaptic weights, or their activation threshold function) are changed from one generation to the next. Evolution therefore allows for the self-organization of the controller, in contrast with the traditional approaches of behavior-based robotics in which the designer has to program the robots' behaviors manually. The input neurons of the ANN receive activations from the robot's sensors, and the output neurons are mapped to the robot's actuators. The usage of ANNs in ER is widespread because they (Floreano and Mondada, 1994): (i) provide evolution with a relatively smooth search space, (ii) are able to tolerate noisy input inherent to most real-world sensors, and (iii) have been shown capable of representing general and adaptive solutions.

After two decades of research in ER, controllers have been successfully evolved for robots with varied functionality, from terrestrial robots to flying robots (Floreano et al., 2005). Although there has been significant progress in the field, it is arguably on a scale that still precludes the widespread adoption of ER techniques (Silva et al., 2014b). Two of the main issues (Silva et al., 2015a) that have prevented ER from becoming a mainstream topic in robotics (Stanley, 2011) are the difficulty in transferring control from simulation to real robots (known as the reality gap), and limitations in terms of task complexity (caused by the bootstrap problem and deception). Below, we discuss these two issues, and we provide an overview of the field of swarm robotics.

2.1 Crossing the Reality Gap

In traditional ER approaches, controllers are synthesized *offline*, in simulation, to avoid the time-consuming nature of performing all evaluations on real robotic hardware. When a suitable controller is found, it can be deployed on real robots. One of the central issues with the simulate-and-transfer approach is the reality gap (Jakobi, 1997), a frequent phenomenon in ER experiments. Controllers evolved in simulation can become inefficient once transferred onto the physical robot due to their exploitation of features of the simulated world that may be different or that do not exist in the real world.

Several authors have addressed such transferring issues. Miglino et al. (1996) proposed three complementary approaches: (i) using samples from the real robots' sensors to more accurately model them in simulation, (ii) introducing a conservative form of noise to promote the evolution of robust controllers, and (iii) continuing evolution in real hardware to tune controllers to the differences between simulation and reality. Using samples from real sensors increases the accuracy of simulations by using a more realistic sensor model, which in turn can decrease the difference between the sensory input experienced in simulation and in reality. Noise can be applied to promote the evolution of robust controllers that can better tolerate variations in the sensory inputs during task execution. Finally, if the performance of the controller decreases after transfer, continuing evolution on real hardware can potentially enable the synthesis of a well-adapted controller in a timely manner.

Jakobi (1997) introduced the concept of *minimal simulations*, in which the experimenter only implements features of the real world deemed necessary for successful evolution of controllers. All remaining features are hidden in an "envelope of noise" in order to minimize the effects of simulation-only artifacts that can prevent successful transfer of evolved control to real robotic hardware. The approach was demonstrated in three tasks. The first task was a T-maze navigation task where a wheeled robot had to choose whether to turn left or right at an intersection depending on the location of a light source in the initial corridor. The second task was a shape discrimination task, in which a gantry robot had to distinguish between two shapes and move towards one of them. The third task was a locomotion and obstacle avoidance task for an eight-legged robot.

It is not clear if Jakobi's approach scales well to complex tasks, since such tasks: (i) typically involve richer robot-environment interactions, and therefore more features, and (ii) require that the experimenter can determine the set of relevant features and build a task-specific simulation model. For example, if the tasks considered involve a large number of robots or robots with high-resolution sensory capabilities such as vision, minimal simulations call for considerable engineering effort because the critical simulation features become more difficult to ascertain and to model (Watson et al., 2002).

Koos et al. (2013) proposed *the transferability approach*, a multi-objective formulation of ER in which controllers are evaluated both by their performance in simulation and their performance on real robots. Contrarily to approaches that simply use individual fitness comparisons of reality versus simulation as a feedback to adapt the simulation model (Zagal and Ruiz-Del-Solar, 2007), the goal of the transferability approach is to *learn the discrepancies* between simulation and reality, and to constrain evolution in order to avoid behaviors that do not cross the reality gap effectively. The transferability approach relies on a surrogate model that is updated periodically by evaluating candidate solutions in real hardware.

The authors tested the approach in a T-maze navigation task with a differential drive robot, and in a locomotion task with a quadruped robot. In both tasks, the transferability approach was able to find a solution to the task in relatively few generations (100 or less). However, the approach can become unfeasible if several hundreds or thousands of generations are required. Moreover, the difficulty in automatically evaluating controllers in real hardware represents an additional challenge.

Lehman et al. (2013) propose a different approach where the evolutionary process explicitly rewards the evolved behaviors for displaying a high reactivity, as measured by the mutual information between the magnitude of changes in a robot's sensors and effectors. The authors showed that reactive controllers transfer better to real robots, regardless of the noise conditions with which the controllers were evolved.

As opposed to the simulate-and-transfer discussed above, *online evolution* executes the evolutionary algorithm on the robots themselves, while they perform their tasks. The main components of the evolutionary algorithm (evaluation, selection, and reproduction) are carried out autonomously by the robots without any external supervision. If the environmental conditions or task requirements change, the robots can modify their behavior to cope with the new circumstances.

Online evolution in a real, ANN-driven robot was first studied by Floreano and Mondada (1994, 1996). The authors evolved navigation and homing controllers for a Khepera robot (Mondada et al., 1999) using an online generational evolutionary algorithm, which was executed on a workstation due to limitations of the robot hardware. The evolution of controllers for a standard navigation and obstacle avoidance task required almost three full days, at a rate of 39 minutes per generation (total of 100 generations), showing that evaluation time is a key aspect in real-robot experiments.

Watson et al. (2002) attempted to accelerate online evolution by distributing the evolutionary process across a group of robots, in an approach known as *embodied evolution*. Robots can test a large number of candidate solutions in parallel and then transmit the best controllers to other robots in the group. Following Watson et al.'s studies on embodied evolution, a number of algorithms for online evolution in multirobot systems were introduced. Examples include an approach for self-assembling of robots (Bianco and Nolfi, 2004), the combination of embodied evolution and reinforcement learning (Wischmann et al., 2007), $(\mu+1)$-online (Haasdijk et al., 2010), mEDEA (Bredeche et al., 2012), MONEE (Noskov et al., 2013), and odNEAT (Silva et al., 2015b).

Online evolution still remains unfeasible to apply in real robotic hardware due to the the large number of evaluations needed until adequate solutions are found, and consequently the large amount time required (Silva et al., 2014b).

2.2 Task Complexity in ER

Driving the evolutionary process towards high-quality solutions, thereby avoiding local optima, is another challenge in ER besides the specific shortcomings of offline evolution and online evolution. When the task to which solutions are sought reaches a certain level of complexity, traditional evolutionary approaches are prone to suffer from bootstrap problems and deception (Doncieux and Mouret, 2014).

Numerous studies have demonstrated evolved controllers for basic tasks. Nelson et al. (2009) surveyed different types of fitness functions used in the field of evolutionary robotics. In the discussion of their findings, they state that evolutionary robotics may possibly "*generate autonomous systems with limited general abilities at some point in the future*". In their survey of more than one hundred different ER studies, there were only reports on the successful application of ER techniques to relatively simple tasks, such as locomotion and obstacle avoidance (Floreano and Mondada, 1996), goal homing (Harvey et al., 1994), foraging (Nakamura et al., 2000), and phototaxis (Watson et al., 2002). In a different survey, Meyer et al. (1998) argued that "*the challenge is to move from basic robot behaviors to ever more complex, non-reactive ones*". The lack of successful applications to complex tasks can partly be attributed to the bootstrap problem and deception.

An approach to avoiding deception was proposed by Celis et al. (2013), which allows for non-expert users to interact with the evolutionary process by allowing them to guide evolution away from local optima. The approach was demonstrated on a simple navigation task with a deceptive fitness function, where the robot has to reach a goal by moving around an obstacle. The non-expert user can aid evolution by defining an intermediate waypoint that the robot should pass through on its way to the target location.

Different approaches have been proposed to solve increasingly more complex tasks. In *incremental evolution*, the experimenter decomposes a task to bootstrap evolution and to circumvent deception. There are numerous ways to apply incremental evolution (Mouret and Doncieux, 2008), such as dividing the task into sub-tasks that are solved sequentially, or making the task progressively more difficult through environmental complexification (Christensen and Dorigo, 2006b). Although incremental evolution can be seen as an approach in which engineering and evolution are combined, it is typically performed in an unstructured manner. The experimenter has to perform a manual switch between the execution of each component of the evolutionary setup, such as different sub-tasks, which can significantly affect the global performance of solutions evolved (Mouret and Doncieux, 2008). In addition, if the components of the setup are highly integrated, incremental evolution can be difficult to apply successfully (Christensen and Dorigo, 2006b).

With the advent of behavior-based robotics, Brooks (1986) introduced the subsumption architecture, in which a controller is divided into a number of manually programmed modules that are organized in a layered structure reflecting their relative priorities. The architecture allows modules to inject data in layers of lower priority and to subsume control. The division of control into modules has since been applied in other approaches. Neural ensembles (Hansen and Salamon, 1990) are compositions of ANNs that collectively decide on a particular action, typically by averaging their output. While modules in neural ensembles all contribute to a particular output simultaneously, the modules in our hierarchical controllers are specialized to perform different tasks.

Moioli et al. (2008) used a homeostatic-inspired GasNet to control a robot in two sub-tasks: obstacle avoidance and phototaxis. The authors used two different sub-controllers that were inhibited or activated by the production and secretion of virtual hormones, and they were able to evolve a controller that activated the appropriate sub-controller depending on external stimulus and internal stimulus. An alternative hormone-based control system was introduced by Hamann et al. (2012) for decentralized control for cooperative multirobot systems. Nolfi and Parisi (1995) experimented with dividing a neural network into different modules in a garbage collection task. The robot had to grasp objects and release them outside of the environmental bounds. For their experiments, they used a Khepera robot with a gripper module. They divided the network's output layer into two modules that competed for activation. The controller evolved one of the modules to find and pick up the objects, and one to release them outside the bounds of the environment.

Lehman and Stanley (2011) introduced Novelty Search (NS) as a means to avoid premature convergence and to overcome bootstrapping issues. In NS, behaviors are not scored based on a traditional fitness function, but based on behavioral novelty with respect to previously evaluated individuals. Since NS does not have a static objective, bootstrapping is typically not an issue, and the constant evolutionary pressure toward behavioral innovation means that NS is unaffected by deception. The features chosen by an experimenter to characterize behavior might not be directly related to the task. If NS exploits such features, the evolutionary process may be unable to find suitable solutions for the task (Gomes et al., 2013). To avoid excessive exploration of regions of the behavior space that are unrelated to the goal task, NS is often combined with fitness-based evolution (Gomes et al., 2014). Despite the promising results obtained with NS, to the best of our knowledge, there has still been no demonstration of control evolved with NS on real robotic hardware that is significantly beyond what has been shown using fitness-based evolution.

Stanley and Miikkulainen (2002) introduced Neuroevolution of Augmenting Topologies (NEAT), an evolutionary algorithm that not only optimizes the weights of the neural controller's connections, but also incrementally augments the controller's topology with new connections and new neurons. NEAT has been shown capable of producing solutions with minimal complexity and to outperform traditional evolutionary techniques in some instances, such as in double pole balancing task, a roving eye for the board game Go, and in the evolution of plastic networks that are able to adapt in a foraging task where food items can become poisonous (Stanley and Miikkulainen, 2004).

Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT) extends NEAT by evolving connective Compositional Pattern Producing Networks (CPPN) (Stanley et al., 2009). Evolved connective CPPNs are then used to encode large-scale ANNs by exploring geometric regularities of the network's topology. Neurons are localized in space, and the connection weights are determined for the entire network by inputing the coordinates of pairs of neuron in the evolved CPPN. The spatial sensitivity of the resulting HyperNEAT network allows the network's topography to be exploited by the evolutionary algorithm. HyperNEAT is able to find the geometric aspects of a task, and can typically lead to highly regular networks (Clune et al., 2011), a feature that can be observed in many biological networks.

Enforced Sub-Populations (ESP) (Gomez and Miikkulainen, 1997) is a cooperative coevolutionary method in which individual neurons are evolved in separate populations. Neurons from the different sub-populations are combined to form a complete ANN controller during evaluation. In approaches such as HyperNEAT and ESP, modularity is thus exploited in different ways and for different parts of the controller. In our approach, each module is an independent controller that solves a particular sub-task and the combination of different modules is decided by the experimenter. Moreover, modules can be reused across different tasks, as opposed to approaches such as HyperNEAT and ESP in which modules are exclusively reused within the context of a single controller or not at all.

Silva et al. (2014a) introduced an approach where a neural controller can have both simple neurons, and more complex *macro-neurons*. Both the structure and the parameters of the macro-neurons and of the network as a whole are subject to variation during an online evolutionary process, which means that the weights and the topology of the ANN can be optimized simultaneously. The authors show how introducing macro-neurons allows for the evolutionary process to find solutions in a significantly lower amount of time, while leading to higher performance in a deceptive phototaxis task with multiple light sources.

Behavior composition has been advocated as an engineering-centric approach when single, monolithic controllers are unable to solve a task (see Floreano, 1998; Correia, 1998 for examples). Several studies have applied behavior composition with different control synthesis techniques, such as genetic programming (Lee, 1999), neuroevolution (Larsen and Hansen, 2005; Becerra et al., 2005), or fuzzy logic control (Tunstel, 1996; Abreu and Correia, 1999, 2001). The tasks in which the approaches have been demonstrated are relatively simple, and have been shown to be solvable by traditional evolutionary techniques. The hierarchical control synthesis approach presented in this thesis is distinct in a number of key aspects: (i) our approach allows for hybrid controllers in which different control synthesis methods are used. Hybrid controllers can take advantage of the benefits of ER, namely the automatic synthesis of behavior, and at the same time the use of manual programming for behaviors that would be infeasible to evolve; (ii) we test our approach in tasks that go beyond the state of the art in terms of complexity. We further demonstrate transfer of behavioral control from simulation to real robotic hardware without loss of performance; (iii) *derived fitness functions* are introduced as a solution to prevent an increase in fitness function complexity as the tasks considered become more complex; and (iv) we demonstrate our approach in a swarm robotic system.

2.3 Swarm Robotics

The field of swarm robotics, as well as the more general field of swarm intelligence, takes inspiration from the observation of social insects, such as ants, bees, wasps, and termites (Şahin, 2005). In these animal societies, relatively simple units rely on self-organization to display collectively intelligent behavior. The self-organized behavior of social insects confers the swarms a high scalability, versatility, parallelism of operation, and robustness to individuals failures (Camazine et al., 2003). The key motivation behind swarm robotics is to harness these properties to build large-scale decentralized multirobot systems (Şahin, 2005; Jones and Mataríc, 2006). Due to their properties, swarm robotic systems have significant potential to take on a number of applications requiring distributed sensing and/or action. In the majority of studies on evolutionary SRS, the swarm is homogeneous (Brambilla et al., 2013), meaning that all robots are morphologically identical, and each robot is independently controlled by a copy of the same evolved controller.

Evolutionary robotics has become a popular alternative to manual programming in a variety of swarm robotic systems applications (Brambilla et al., 2013). Evolutionary approaches have been used to solve a large number of swarm robotics tasks such as coordinated motion (Baldassarre et al., 2007; Sperati et al., 2008), chain formation (Sperati et al., 2011), aggregation (Trianni et al., 2003; Bahgeçi et al., 2005; Soysal et al., 2007), flocking (Baldassarre et al., 2003), hole avoidance (Trianni et al., 2006), aerial vehicles communication (Hauert et al., 2009), categorization (Ampatzis et al., 2008), group transport (Groß and Dorigo, 2008, 2009), social learning (Pini and Tuci, 2008), and sharing of an energy recharging station (Gomes et al., 2013).

Several large-scale projects have explored the use of SRS for a variety of applications. The SWARM-BOTS project (Dorigo et al., 2005) focused on the development of hardware and control of a SRS composed of robots known as the s-bots (Mondada et al., 2004). The s-bots are able to physically connect to one another using a gripper, allowing them to form complex, multirobot morphologies. This capability can be used to solve various collective tasks, such as form pulling chains to retrieve heavy objects (Groß et al., 2006), crossing holes in the environment (Trianni et al., 2006), or navigate steep terrain (O'Grady et al., 2005). The project studied several areas of swarm behavior, such as aggregation, coordinated motion, collective and cooperative transport of items, exploration, task-allocation, navigation, and self-assembly.

The SWARMANOID project (Dorigo et al., 2013) extended SWARM-BOTS by studying heterogenous swarms, that is, swarms composed of robots with different morphologies and capabilities. Behavior heterogeneity can allow for a better efficiency through the division of labor, and morphology heterogeneity has the potential of augmenting the swarm's capabilities, while reducing the overall cost by keeping each robot simple. SWARMANOID extended the application of SRS to three-dimensional human-centric environments by using three types of robot: the hand-bot, which was able to climb structures, the eye-bot, a flying robot that can attach to ceilings, and the foot-bot, small differential drive ground robots. In the final demonstration of the system, the robots performed a search and retrieval task, where a book had to be located and collected from a bookshelf. In the task, a group of foot-bots carried a hand-bot to the bookshelf, which in turn climbed and retrieved the book. During the task, various eye-bots guided the ground robots (Dorigo et al., 2013).

The SYMBRION project (Kernbach et al., 2008) explored large-scale robotic organisms composed of multiple simple robots. By physically docking with one another, the robots can symbiotically share computational resources and energy, as well as perform tasks that would be impossible for a single robot to perform. The authors advocate that such a robotic system can have several advantageous features, such as being self-configuring, self-healing, self-optimizing and self-protecting from both the software and the hardware perspectives

The CoCoRo (Schmickl et al., 2011) project, which focuses on synthesis of control for underwater robots, and the ASSISIbf project (Halloy et al., 2013), which aims to develop communication channels between robots and animals (fish and honeybees, in particular), have studied the application of SRS to aquatic environments. While these projects rely on bio-inspired control systems, they are only tangentially related with evolved ANN-based control, such as the one presented in this thesis. Although the use of ER in aquatic robots has not yet been widely explored, a few studies have approached the subject. Some examples include the evolution of locomotion behaviors (Meyer et al., 2003; Ijspeert et al., 2007), station keeping for an underwater robot (Moore et al., 2013), and a neuroaugmenting approach to the evolution of centralized control for a small team of Autonomous Underwater Vehicles (AUVs) (Praczyk, 2014).

Swarm robotic systems have not leveraged their full potential up to this point, and are still far from real-world applications. In recent and broad surveys of SRS research, Brambilla et al. (2013) states that "*in all real-robot experiments presented in the analyzed literature, the experiments are performed in controlled environments*", and Bayındır (2016) advocates that "*to this date the use of robotic swarms in real-world applications is still lacking: while laboratory experiments can give a sense of what a given robotic system might achieve, large-scale deployments in the field would provide new insights on the different factors affecting the operation of a swarm and would stimulate further research*".

There are multiple reasons for the absence of real-world swarm robotic systems, one of the most prevalent being the difficulty in designing behavioral control for the individual robots that results in the desired swarm-level behavior. This problem is exacerbated when trying to achieve behaviors capable of dealing with the complexity of real tasks and uncontrolled environments, as opposed to the abstract tasks and highly controlled environments that have been used in the majority of previous works. In this thesis, we demonstrate for the first time a SRS with evolved control performing swarm behaviors outside of controlled laboratory conditions. We designed and developed an aquatic swarm robotics platform, synthesized control offline (in simulation), and transferred successfully transferred control to the real robots (see Chapter 5). Furthermore, we go beyond the current state of the art in terms of task complexity for SRS by applying the hierarchical control synthesis approach.

Chapter 3

Methodology

In this chapter, we describe the hierarchical control synthesis approach. The motivation for introducing the proposed approach was to address key issues in the field of ER, namely the bootstrap problem, deception, and the reality gap (see Chapter 1). Even though the hierarchical control synthesis approach was designed with the field of ER in mind, it also allows for non-ER components, such as manually programmed control, to be seamlessly integrated in the control structure.

The chapter is divided in six different sections that explore the following topics: (i) an overview of the hierarchical approach and the definition of key concepts, (ii) how the structure of a controller is derived from the task decomposition, (iii) high-level composition of control, (iv) hybridity and manually programmed control, (v) the types of controllers used throughout this thesis, and (vi) the potential advantages and limitations of the approach as a whole.

3.1 Overview and Definitions

The hierarchical control synthesis approach aims at structuring a robot's control program hierarchically by dividing it into multiple modules and organizing them in a tree structure. The approach is inspired by the general concept of hierarchy, which can be found in many fields, such as engineering, social organizations, computer systems, and biology (Simon, 1991). Hierarchy allows for a divide-and-conquer approach to complex problems by breaking them down to smaller and more manageable blocks.

The application of a hierarchical control system allows for the combination of artificial evolution with more traditional robotics engineering techniques, such as manual programming, by combining different approaches at different points of the controller hierarchy. Our hierarchical controllers are composed of several independent nodes and organized hierarchically, as seen in Figure 3.1. Each node in the hierarchy is either a *behavior arbitrator* or a *behavior primitive* (Lee, 1999) (see Figure 3.1). The different nodes are obtained by decomposing a task into multiple sub-tasks, synthesizing control for each sub-task, and then composing the nodes based on the task decomposition.

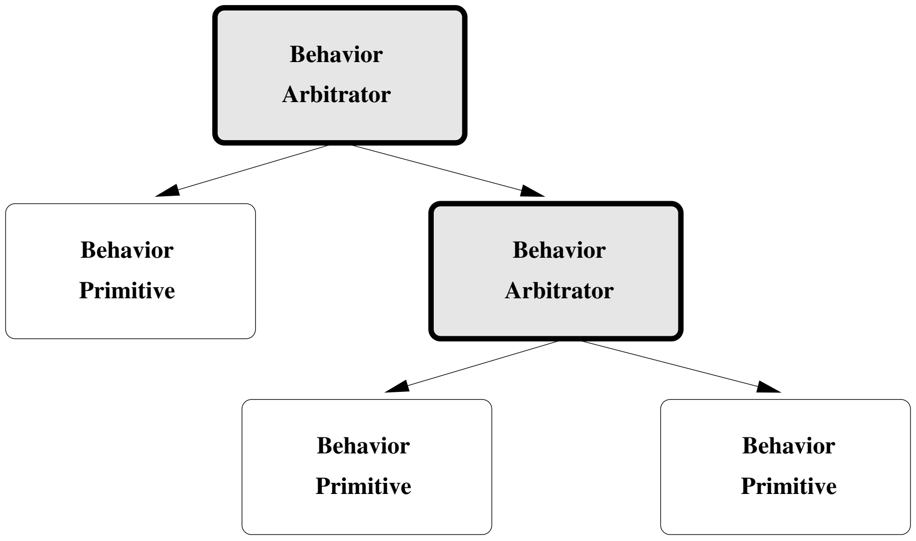


FIGURE 3.1: Representation of a hierarchical controller. A behavior arbitrator delegates the control of the robot to one or more of its sub-controllers. A behavior primitive controls the actuators of the robots directly.

The control mechanism of each node in the hierarchy is completely independent from other nodes. Our approach allows for any type of control synthesis technique to be used, such as artificial evolution, manual programming, fuzzy logic, etc. If evolution is used, different nodes in the same controller can also be evolved under different simulation conditions and even using different evolutionary algorithms. If a hierarchical controller is composed of nodes synthesized with different techniques, we refer to such a controller as *hybrid*. Hybrid controllers allow the experimenter to leverage the advantages of each control synthesis method for specific nodes, for instance artificial evolution for nodes that can benefit from self-organization and automatic synthesis of behavior, and manual programming for nodes that require fine sensorimotor coordination (Er et al., 2002; Hehn and D'Andrea, 2011).

Below, we identify and define key concepts associated with our approach.

Behavior node: A behavior node is a single block of behavior in the controller hierarchy. It can be either a single behavior primitive or a single behavior arbitrator. Each behavior node is independent from other nodes in the controller hierarchy.

Behavior primitive: Behavior primitives are self-contained control nodes that directly actuate the robot. These nodes are the leafs of the controller, which means that they have no child nodes. Behavior primitives have access to the robot's sensors, and can control the actuators of the robot, such as wheels or grippers. A behavior primitive can access either all or a sub-set of the robot's sensors and actuators, allowing for different robot configurations for different behavior primitives.

Behavior arbitrator: Behavior arbitrators are decision nodes that delegate control to one or more of its child nodes. Each behavior arbitrator has two or more sub-controllers, which can, in turn, be either behavior arbitrators or behavior primitives. The term sub-controller refers to a specific node and all of the node's child nodes. Similarly to behavior primitives, a behavior arbitrator can have access to either all or a sub-set of the robot's sensors. Since the behavior arbitrator delegates control to one of its sub-controllers, it does not directly control the robot's actuators.

Controller: A controller is a behavior node and all its child nodes. When we consider a behavior primitive as a controller, we are only referring to the behavior primitive itself, since it is a leaf node. When we consider a behavior arbitrator as a sub-controller, we are referring to that behavior arbitrator and all its descendants. We can also refer to a controller as a *sub-controller*, if it is part of the structure of a more complex controller.

3.2 Task Decomposition

In the hierarchical control synthesis approach, we resort to manual division of a task into simpler sub-tasks when an evolutionary process is unable to find a solution for the task. For each sub-task, a sub-controller is defined, which means that the hierarchy of the controllers is defined by the task decomposition. If it is possible to find an *appropriate fitness function* for a given task, a behavior primitive composed of a single ANN is evolved to solve the task. An appropriate fitness function is one that (i) allows evolution to bootstrap, (ii) leads to controllers that are able to solve the task consistently and efficiently in simulation, and (iii) leads to controllers that maintain performance when transferred to real robotic hardware.

In case an appropriate fitness function cannot be found, the task is manually and recursively divided into sub-tasks until an appropriate fitness functions has been found for each sub-task, resulting in multiple hierarchical levels. Even though we ideally rely on evolutionary techniques to solve every sub-task, different control synthesis can be used if deemed more suitable for a particular sub-task. In this thesis, we resort to manual programming of behavior primitives if an appropriate fitness function cannot be found for a sub-task that cannot be further divided, or if fine sensorimotor behaviors are required (see Section 4.4).

3.3 High-level Composition of Control

As we move up the controller hierarchy and attempt to evolve behavioral control for increasingly complex tasks, fitness functions that lead to the evolution of adequate solutions may be challenging to define. The challenge can stem from the need to carry out a various high-level objectives, which can lead to convoluted fitness functions that present evolution with multiple local optima. In such cases, the fitness function can be *derived* based on the task decomposition. The derived fitness function is constructed based on the immediate task decomposition to reward the arbitrator for activating a sub-controller that is valid for the current sub-task, rather than for solving the global task. The experimenter may not always know which sub-controller is the optimal one for a given situational context, and for such cases, more than one sub-controller can be specified as valid for a context. A general example of a derived fitness function can be see in $f_{derived}$, where the controller is rewarded for the number of control cycles where it executed a valid sub-controller, and penalized for the number of control cycles where it executed an invalid sub-controller:

$$f_{derived} = \sum_{s=1}^{n} \left(\frac{t_s - w_s}{T_s} \right) + \begin{cases} 1 - \dfrac{c}{C} & \text{if task completed} \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

where the sum is over all the started sub-tasks, n is the total number of sub-tasks, t_s is the number of control cycles in which the controller chose a valid sub-controller for sub-task s, T_s is the number of cycles that the controller has spent in sub-task s, w_s is the number of cycles in which the controller chose an invalid sub-controller for sub-task s, C is the maximum number of control cycles for the given complete task, and c is the number of control cycles spent during task execution.

Derived fitness functions can be applied in offline evolution, where it is possible to have global knowledge of the states of the robots and the environment, potentially leading the evolutionary process towards high-quality solutions. The set of valid sub-tasks can be defined using simple rules, such as the position of the robot, the proximity to a particular environmental feature, or based on onboard sensory readings. Derived fitness functions therefore rely on the evolutionary process to find controllers that are able to differentiate the different sub-tasks automatically based only on the robots' onboard sensory readings.

3.4 Hybridity and Manually Programmed Control

Although the hierarchical control synthesis approach allows for the evolution of simpler behavior nodes by decomposing the task into simpler sub-tasks, it can still prove challenging to apply evolutionary methods in some situations, such as tasks that require fine sensorimotor coordination, behaviors that must always execute for a fixed amount of time, or very complex behavior arbitrators. In such cases, it can be beneficial from an engineering point of view to forego the use of evolutionary techniques.

Manually programmed behavior primitives can be an alternative to evolved behavior primitives when fine sensorimotor behaviors are required. Fine sensorimotor behaviors are those that require the robot to perform actions with a high degree of accuracy, and that depend on the precise information obtained through the robot's sensors. When such behaviors are necessary for solving the task, evolved control may be combined with manually programmed behaviors, which can be fine-tuned manually for the real robotic hardware.

The use of manually programmed behaviors is particularly beneficial for subtasks and behaviors that are infeasible to simulate with sufficient accuracy to allow for the successful transfer to real robotic hardware. Behaviors that have only been implemented for the real robotic hardware can be integrated in simulation in different ways. We propose two alternative approaches: (i) *sensor playback*, where samples can be collected from the real robot performing the manually programmed behavior and then played back in simulation, or (ii) *offline behavior*, where the normal control cycle is stopped completely while the robot executes the manually programmed behavior. In either case, it is only necessary to change the state of the environment according to the robot's actions, and not to simulate the detailed robot-environment interaction. We explore the *offline behavior* technique in the experiments described in Section 4.4.

When using manually programmed control, it might be necessary to allow a particular behavior to finish executing. Examples include changing behavior half-way through its execution in such a way that the robot would be in an undesirable or unsafe state (for instance, a legged robot climbing an obstacle), or when using the offline behavior approach in which only the outcome of an action is defined in simulation. For such behavior nodes, we introduce the concept of *locking* the controller. While a locking manually programmed behavior is executing, no other behavior primitive can be executed. The locking mechanism helps to guarantee transfer of control from simulation to a real robot by allowing manually programmed behaviors to complete before another behavior is executed. Locking behaviors are demonstrated in the task presented in Section 4.3.

Control for high-level behavior arbitrators can be challenging to synthesize, especially if the task is sequential and/or composed of many different sub-tasks. In such cases, simple decision mechanisms, such as the Finite State-machines (FSMs), can be a flexible way of achieving the desired macroscopic behavior. As shown in Chapter 5, FSMs behavior arbitrators can allow the high-level robotic behavior to be fine-tuned to suit particular constraints of the task.

3.5 Studied Hierarchical Controllers

For the experiments presented in this thesis, we use the following control synthesis techniques to generate control nodes: evolution of Continuous-time Recurrent Neural Networks (CTRNNs) (Beer and Gallagher, 1992) (Chapter 4) and NEAT networks (Stanley and Miikkulainen, 2002) (Chapter 5), manually programmed instructions (Chapter 4), and finite state-machines (Chapter 5).

In our experiments, behavior arbitrators delegate control to a single sub-controller at any given time. ANN-based behavior arbitrators have one output neuron for each sub-controller that they can activate. The output with the highest activation determines which sub-controller is activated at each control cycle. Alternative methods for activating sub-controllers could be used, such as allowing for multiple sub-controllers to be active at the same time, or combining the outputs of different behavior primitives. The activation of multiple sub-controllers could be useful, for instance, to activate locomotion and communication sub-controllers simultaneously, or to generate control based on the combination of the locomotion patterns from multiple behavior primitives.

The state of a node, which can be represented by the states of hidden neurons in the case of ANNs or variables in the case of preprogramed controllers, is reset whenever it is deactivated. Reseting sub-controllers allows for predictable action patterns when a behavior arbitrator switches from one sub-controller to another, since uncertainty regarding the controller's current state (and therefore its behavior) is eliminated. Alternatively, sub-controllers could be simply resumed when they are handed control, or even allowed to constantly integrate new sensory readings even when deactivated.

3.6 Discussion

The division of control in our approach has significant advantages when compared with traditional monolithic ER-based approaches. On the one hand, we can still take advantage of the benefits of ER techniques, such as the automatic synthesis of self-organized behavior. On the other hand, we are not limited by the key issues of ER, such as the difficulties associated in scaling to complex tasks. By maintaining a repertoire of previously synthesized behaviors, it becomes possible to reuse these behaviors in subsequent experiments if sub-tasks are sufficiently similar or if the behavior is sufficiently general.

Transferability issues can be addressed incrementally during the development of the control system, that is, as soon as a sub-controller has been synthesized, it can be tested in real robotic hardware. Transferability can also be improved by promoting the evolution of robust and general behaviors through the inclusion of noise during evolution and the evaluation of controllers in a multitude of different scenarios. Such an approach would be detrimental for the evolution of complex, monolithic controllers, since it generally makes the evolutionary process more challenging.

The potential cost of applying our approach is that evolution is constrained. In ER, the solution space is restricted by various factors, such as the characteristics of the robot, the type of controller, and the fitness function. By dividing a controller into various sub-controllers, we are further limiting the set of potential solutions. While it may be argued that manual decomposition could potentially prevent certain solutions from ever being discovered by the evolutionary process, the experimenter only decomposes a task when evolution is unable to find a solution. It is only in cases where the task cannot be decomposed that our approach cannot be applied, since it would be necessary to synthesize a monolithic solution with an alternative technique.

Chapter 4

Synthesis of Hierarchical Control for Single-robot Systems

In this chapter, we apply our approach (see Chapter 3) for decomposable complex tasks that combines the benefits of ER, namely automatic synthesis of control, and human engineering to circumvent the bootstrap problem, to avoid deception, and to successfully cross the reality gap. We demonstrate our approach in two distinct tasks, which are beyond the complexity level of current state of the art in ER: a rescue task involving a double T-Maze, and a dust cleaning task where a robot must press a button to move between two rooms. In both tasks, controllers evolved in simulation maintain their performance when transferred to a real robot. This chapter is structured as follows: in Section 4.1, we describe the experimental setup; in Section 4.2, we demonstrate the synthesis of an evolved hierarchical controller for a complex task; in Section 4.3, we combine evolved behavior arbitrators and manually programmed behavior primitives; and in Section 4.4, we show how our approach can be used to solve a non-sequential, integrated task where both evolved and manually programmed behavior primitives are used.

4.1 Experimental Setup

In simulation, we run a total of ten evolutionary runs with a population size of 100 genomes for each evolved node in a controller hierarchy. The fitness score assigned to each genome is the average score obtained in 50 simulations with different initial conditions. The five highest scoring genomes are copied directly to the next generation. Another 19 copies of each genome are made and mutation is applied to each gene with a probability of 10%. A Gaussian offset with a mean of zero and a standard deviation of one is applied when a gene undergoes mutation. The evolutionary runs for each node are terminated after an empirically determined number of generations. Upon termination, we conduct a post-evaluation of the highest scoring controller of each evolutionary run in a total of 100 samples for each different configuration of the environment. Genomes consist of floating-point alleles that encode the parameters of a CTRNN with one hidden layer of fully-connected neurons. The neurons in the hidden layer are governed by the following equation:

$$\tau_i \frac{dH_i}{dt} = -H_i + \sum_{j=1}^{J} \omega_{ji} I_j + \sum_{k=1}^{K} \omega_{ki} Z(H_k + \beta_k) \tag{4.1}$$

where τ_i is the decay constant, H_i is the neuron's state, J is the number of input neurons, ω_{ji} the strength of the synaptic connection from neuron j to neuron i, I is the set of input neurons, K is the number of hidden neurons, β is the bias term, and $Z(x) = (1 + e^{-x})^{-1}$ is the sigmoid function. The bias terms β_i, the decay constants τ_i, and the connection weights ω_{ji} are genetically controlled network parameters. The possible ranges of these parameters are: $\beta_i \in [-10, 10]$, $\tau_i \in [0.1, 32]$ and $\omega_{ji} \in [-10, 10]$. Circuits are integrated using the forward Euler method with an integration step-size of 0.2 and the states of hidden neurons are set to 0 when the network is initialized.

For our real hardware experiments, we used an e-puck robot (Mondada et al., 2009) (see Figure 4.1). The e-puck is a small circular (diameter of 75 mm) differential drive mobile robotics platform designed for educational use. The e-puck's

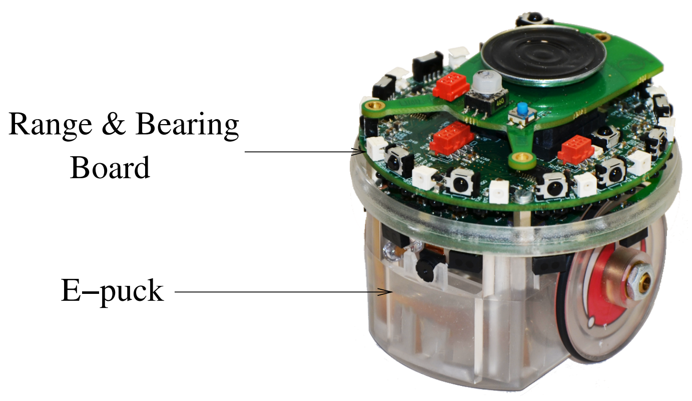


Figure 4.1: The e-puck with a range & bearing board.

set of actuators is composed of two wheels that enable the robot to move at speeds of up to 13 cm/s, a loudspeaker, and a ring of eight LEDs. The e-puck is equipped with several sensors: (i) eight infrared proximity sensors which are able to detect nearby obstacles and changes in light conditions, (ii) three microphones (one near each wheel of the robot, and one toward the front), (iii) a color camera, and (iv) a 3D accelerometer. Additionally, our e-puck robots are equipped with a range and bearing board (Gutiérrez et al., 2008), which allows them to communicate with one another.

We use four of the e-puck's eight infrared proximity sensors: the two front sensors and the two lateral sensors. We collected samples from the sensors on a real e-puck robot in order to model them in simulation, as advocated by Miglino et al., 1996. Each sensor was sampled for 10 seconds (at a rate of 10 samples/second) at distances from an obstacle ranging from 0 cm to 12 cm. We collected samples at increments of 0.5 cm for distances between 0 cm and 2 cm, and at increments of 1 cm for distances between 2 cm and 12 cm.

We use ray-casting to model the infrared sensors in simulation. Seven rays are cast from each sensor in different directions, from $-\frac{\alpha}{2}$ to $\frac{\alpha}{2}$, where α is the sensor's opening angle. Based on experimental data from the robot, we used an α value of 70°. The distances at which the rays intersect with an obstacle are averaged and the final sensor reading is the interpolation of the closest samples collected on the real robot. Finally, noise is added to the reading with an amount based on the variance measured on the real robot for the particular distance.

The e-puck's infrared sensors can also be used to measure the level of ambient light. In the experiments presented in Section 4.2 and Section 4.3, we use ambient light readings from the two lateral proximity sensors to detect light flashes. When a light flash is detected on one of the sides, the activation of its corresponding sensor is set to 1. The sensor remains activated for 15 control cycles (equivalent to 1.5 seconds) to indicate that a flash has been detected. In simulation, we added Gaussian noise to the wheel speeds, with a standard deviation corresponding to 5% of the current wheel speed in each control cycle. The robot's speed is limited to 10 cm/s in our experiments.

The e-puck only has 8 kb of onboard memory. If the control code does not fit within the e-puck's limited memory, we run the control code off-board. When the control code is executed off-board, the e-puck starts each control cycle by transmitting its sensory readings to a workstation via Bluetooth. The workstation then executes the controller, and sends back the output of the controller (wheel speeds) to the robot. Due to memory constraints on the chosen robotic platform, we use off-board execution of control code in the real-robot experiments conducted in Section 4.2 and Section 4.4. We use on-board execution of control code in the real-robot experiments conducted in Section 4.3.

4.2 Evolving and Transferring Controllers for Complex Tasks

In this section, we apply the proposed methodology to a rescue task. The task is relatively complex and requires several behaviors typically associated with ER (Nelson et al., 2009) such as exploration and obstacle avoidance (Floreano and Mondada, 1996), delayed response (Tuci et al., 2004), and the capacity to navigate safely through corridors (Reynolds, 1994).

We purposefully designed the task to be more complex than any previously solved by real robots with evolved controllers: (i) a robot must first find its way out of a room with obstacles, (ii) the robot must then solve a double T-maze (Blynel and Floreano, 2003), and finally (iii) the robot must guide a teammate safely to the room. Variations of the T-maze task have been used extensively in studies of learning and motivation in animals (Tolman and Honzik, 1930), neuroscience (Torta et al., 2008), and robotics. In robotics, T-mazes have been used to study different neural network models such as diffusing gas networks (Husbands, 1998), the online learning capability of continuous time recurrent neural networks (Blynel and Floreano, 2003), and the evolution of transferable controllers (Jakobi, 1997; Koos et al., 2013). While a T-maze task may appear simple from an anthropomorphic perspective, robots' sensors are often limited, which makes it challenging. In our experiment, each sensor only provides the controller with a single scalar or binary value. The information available to the controller about the environment and about the robot's position in the environment is thus very limited. In previous studies in which evolved control has been tested on real hardware, only single T-mazes were used.

A number of obstacles are located in the initial room. The room has a single exit that leads to the start of a double T-maze (see Figure 4.2). In order to find its teammate, the robot should exit the room and navigate to the correct branch of the maze. Two rows of flashing lights in the main corridor of the double T-maze give the robot information about the location of the teammate. Each row of lights indicates the branch leading to the teammate in a junction. For instance, if the left light of the first row and the right light of the second row are activated, the robot should turn left at the first junction and right at the second junction. Upon navigating to the correct branch of the maze, the robot must guide the teammate back to the initial room. We included a boolean *near robot* sensor that indicates if the teammate is within 15 cm. For this sensor, we use readings from the range and bearing board.

For the real-robot experiments, we built a double T-maze with a size of 200 cm × 200 cm (see Figure 4.2). In the real maze, a Lego Mindstorms NXT

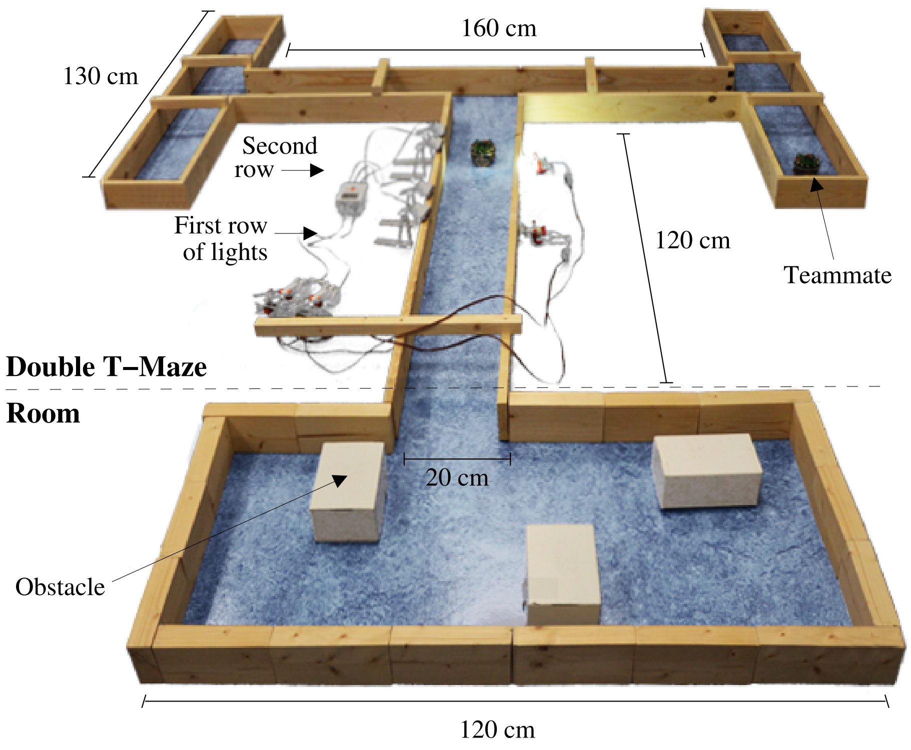


FIGURE 4.2: The environment is composed of a room with obstacles and a double T-maze. The room is rectangular with varying side lengths. The double T-maze has a total size of 200 cm × 200 cm. The two rows with the lights are located in the central maze corridor. The activation of these two rows of lights indicates the location of a teammate.

brick controlled the flashing lights. The brick was connected to four ultrasonic sensors that detected when the robot passed by. Lights were turned on by the first and third ultrasonic sensor and turned off by the second and fourth ultrasonic sensor. The NXT brick controlled the state of the lights using two motors.

We use simple, functional incremental fitness functions (Nelson et al., 2009) for the evolution of behavioral primitives in the experiments presented below. Each fitness function typically has a number of cases that represent different outcomes of an experiment such as whether a robot reached its destination or not. Each case is kept simple and typically has only one or two terms. The cases are weighted by adding a constant and by multiplying by a factor, initially with values of zero and one, respectively. The values of the constants and factors are adjusted through an empirical trial-and-error process when necessary: if a bootstrapping case is exploited over a goal case, guiding evolution toward local optima, the relative weight of the exploited case is either decreased or the weight of the other cases are increased. While the exact weights are not crucial, the relative weight between cases need to be such that solving the task is significantly better than not solving the task. For instance, bootstrapping cases should have significantly lower weights than high-level goal cases. The constants and factors that appear in the fitness functions below are the result of such a process. Given the simplicity of the fitness functions used, this process usually only required a couple of iterations.

4.2.1 Attempting to Evolve a Monolithic Controller

We attempted to evolve a monolithic controller, that is, a controller with a single neural network for the complete rescue task. The chosen neural network was composed of seven input neurons (two light sensors, four infrared sensors, and one robot sensor), ten hidden neurons and two output neurons (two wheels). The robot was placed at a random position and with a random orientation in the room. The complexity of the task translates to a difficulty in finding an appropriate fitness function that allows evolution to bootstrap. In order to evaluate the controller, we chose a functional incremental, gradient-based fitness function with a bonus for reaching three intermediate points of the task: exiting the room, finding the teammate, and returning to the room. The fitness function is defined as follows:

$$f_{monolithic} = \sum_{i=1}^{3} \beta_i + \frac{D_i - d_s}{D_i} \tag{4.2}$$

where β_i is the bonus constant for reaching each intermediate point, D_i is total Euclidean distance from the previous intermediate point of task to the current intermediate point of the task, d_s is the robot's Euclidean distance to the next intermediate point of the task. The bonus constant of each term in the fitness function was chosen based on a derived fitness function (see Section 3). The controllers were evolved for 1000 generations, and were post-evaluated in order to measure how many times they navigated to each of the intermediate points of the task. The controllers were able to solve the first part of the task and leave the room in 92% of the post-evaluation samples, and navigated to the correct exit of the double T-maze in 51% of the samples. The highest-performing controllers in some of the runs specialized in solving the task for a particular maze exit (25% of the samples in which the robot managed to leave the room), and as a result, the average solve rate for the complete task was only 17%.

We also tried to evolve a monolithic controller for the rescue task using NEAT (Stanley and Miikkulainen, 2002). The controllers were evaluated based on $f_{monolithic}$ (see Equation 4.2) in a total of 10 evolutionary runs. Each run was evolved for 1000 generations, and we used the parameter values proposed in (Stanley and Miikkulainen, 2002). We observed that evolution typically got stuck in a local optimum after around 300 generations. On average, the highest-performing controllers of the 10 evolutionary runs were able to leave the room in 77% of the samples, navigated to the correct exit of the room in 19% of the samples, but none of the controllers was able to return to the initial room. We experimented with several variations of the default parameter values, but obtained similar or inferior results.

4.2.2 Decomposition of the Rescue Task

As shown in Section 4.2.1, the rescue task is relatively complex, which meant that we were unable to find an appropriate fitness function that allows evolution of a controller based on a single ANN. We therefore divided the rescue task into three sub-tasks: (i) exit the room, (ii) solve the double T-maze to find the teammate, and (iii) return to the room, guiding the teammate. We evolved sub-controllers to solve each of the sub-tasks. A behavior arbitrator was then given access to each of the sub-controllers and evolved to solve the complete rescue task. Figure 4.3 shows the hierarchical structure of the complete controller, the description of each behavior arbitrator and behavior primitive, including the topology, inputs (sensors) and outputs (actuators) used by of each neural controller, and the performance results in simulation.

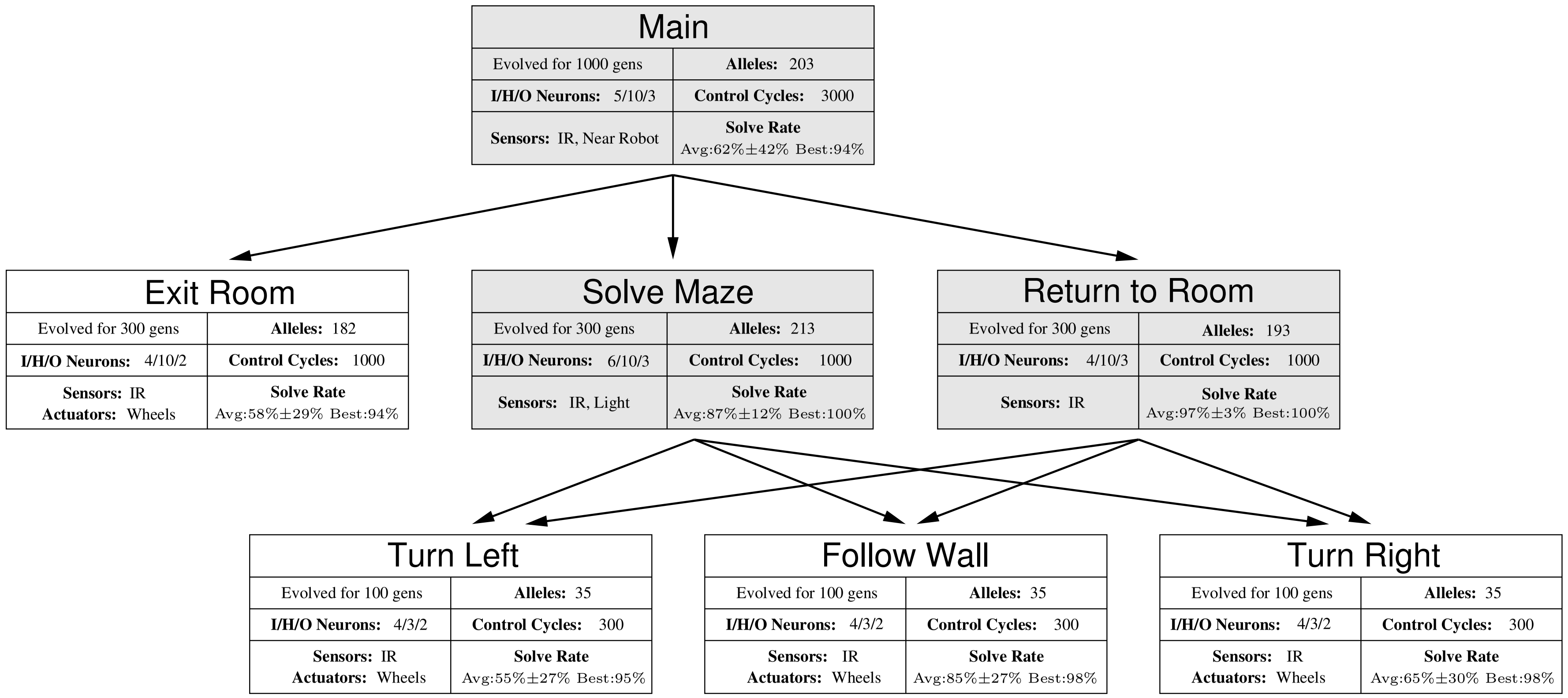


FIGURE 4.3: Hierarchical Rescue Task Controller. The controller used in our experiments is composed of three behavior arbitrators (with darker background) and four behavior primitives. In each node, we list its name, the number of generations for which the sub-controller was evolved, the number of alleles, the number of input, hidden and output neurons, the sensors and actuators, the number of control cycles for each evaluation and the average and best solve rate of the post-evaluation. We performed an initial set of experiments to test different parameter values and used the set of parameters yielding the highest performance for the experiments summarized in the figure.

4.2.2.1 Exit Room Sub-task

The first part of the rescue task is an exploration and obstacle avoidance task. The robot is located in a room and must find a narrow exit leading to the maze. The room is rectangular with side lengths that vary between 100 cm and 120 cm drawn from a uniform distribution. We evolved the "Exit Room" behavior primitive to solve this sub-task. In each sample, we placed either two or three obstacles in the room depending on its size. Each obstacle was rectangular with random side lengths ranging from 5 cm to 20 cm drawn from a uniform distribution. The location of the room exit was also randomized in each trial. The robot was randomly oriented and positioned inside the room at the beginning of each sample. Controllers were evaluated differently depending on whether they found the exit of the room within the allotted time or not, according to f_{exit_room}:

$$f_{exit\_room} = \begin{cases} 5 + \frac{C-c}{C} & \text{, if exit was found} \\ \frac{D-d}{D} & \text{, time expired} \end{cases} \tag{4.3}$$

where C is the maximum number of control cycles (1 second = 10 cycles), c is the number of cycles spent, D is the distance from the center of the room to its exit, and d is the closest point to the exit that the robot reached. f_{exit_room} rewards controllers that move from the center of the room and toward the exit.

In two of the ten evolutionary runs, the highest scoring controller was able to find the exit of the room in over 90% of the post-evaluation samples. The highest performing controller starts by moving away from the center of the room until it senses a wall, which it then follows counter-clockwise until the room exit is found. The remaining eight runs did not produce successful behaviors: the robots would spin around in circles, sometimes finding the exit by chance but often colliding with one of the walls or with an obstacle.

4.2.2.2 Solve Maze Sub-task

In the second sub-task, the robot has to solve a double T-maze in order to find a teammate that needs to be rescued (see Figure 3). Controllers were evaluated according to f_{maze}:

$$f_{maze} = \begin{cases} 1 + \frac{C-c}{C} & \text{, if navigated to destination} \\ \frac{D-d}{3D} & \text{, if collided or chose wrong path} \\ 0 & \text{, if time expired} \end{cases} \tag{4.4}$$

where C is the maximum number of control cycles, c is the number of cycles spent, D is the distance from the start of the maze to the robot's destination, and d is the final distance between the robot to its destination. f_{maze} awards a score of one plus a speed bonus in case the robot reaches the destination. If the robot collides with a wall or chooses a wrong path, a lower fitness is awarded, calculated based on how close to the destination it managed to get. The experiment ends prematurely if the robot either collides with a wall or reaches one of the maze exits. In such cases, we do not count the time as having expired.

We divided the "Solve Maze" sub-task into three different sub-tasks: "Follow Wall", "Turn Left", and "Turn Right", for which appropriate fitness functions could easily be specified. For each sub-task, we evolved a behavior primitive. Although the intended behaviors were quite simple, we evolved the controllers in a wide variety of simple mazes, some of which had a high degree of difficulty due to the starting position and orientation of the robot. The difficulty of some of the mazes had an impact on the solve rates of these controllers, bringing the average down to 55% in the turn left controller and 65% in the turn right controller (see Figure 4.3). We used ten different mazes for the "Turn Left" and "Turn Right" controllers. For the "Follow Wall" controllers, only four different types of corridors were used: (i) a normal corridor, two corridors with respectively (ii) the left and (iii) the right walls missing, so that the controllers learn how to navigate when they can only detect one of the walls, and (iv) a corridor with small gaps both in the left and right walls so that controllers are robust to variations in the wall detection on both sides simultaneously.

We then evolved the “Solve Maze" behavior arbitrator with the three highest scoring behavior primitives as sub-controllers. At the beginning of each trial, the robot was placed at the start of the double T-maze and had to navigate to the correct branch based on the activations of the lights in the central corridor (see Figure 4.2). The robot was evaluated by f_{maze}, and the simulation sample was terminated if the robot collided with a wall or if it navigated to a wrong branch of the maze.

4.2.2.3 Return to Room Sub-task

The final sub-task consists of the robot guiding its teammate back to the room. We initially tried to evolve a behavior primitive for this sub-task, but the evolved solutions proved difficult to transfer successfully to the real robot. We therefore reused the behavior primitives previously evolved for maze navigation (“Follow Wall”, “Turn Left” and “Turn Right”), and evolved a new behavior arbitrator. The new behavior arbitrator was evolved in the double T-maze with the robot starting in one of the four branches of the maze (chosen at random in the beginning of each trial). In the guidance sub-task, the robot had to navigate correctly through the maze, and we therefore used the same fitness function, f_{maze}, as in the “Solve Maze” sub-task. The only difference was the objective: the robot was evaluated based on how close it got to the initial room (see Figure 4.2), and not the distance to the teammate.

For the complete task, we evolved a behavior arbitrator with the highest scoring controllers for the “Exit Room”, the “Solve Maze”, and the “Return to Room” sub-tasks as sub-controllers. The teammate being rescued continuously emitted a signal while waiting for the rescuing robot. We used a derived fitness function, $f_{derived}$ (see Section 3), to evolve the “Main’ behavior arbitrator for the rescue task. The derived fitness function rewards the selection of a valid behavior for the current sub-task and penalizes the selection of an invalid behavior (for instance, selecting the "Exit Room" behavior primitive if the robot was in the T-maze). The controller was awarded a fitness score between 0 and 1 for each sub-task (hence a maximum score of 3 for successful completion of all sub-tasks), depending on the proportion of the time that it selected the valid behavior, plus a time bonus.

The fitness trajectory for the highest scoring controller evolved and the average fitness trajectory of all ten evolutionary runs for the rescue task can be seen in Figure 4.4. The highest scoring controller had a solve rate of 94%. Out of the ten controllers, seven were able to consistently solve the whole rescue task in over 89% of the samples, while the remaining three were not able to solve the three sub-tasks.

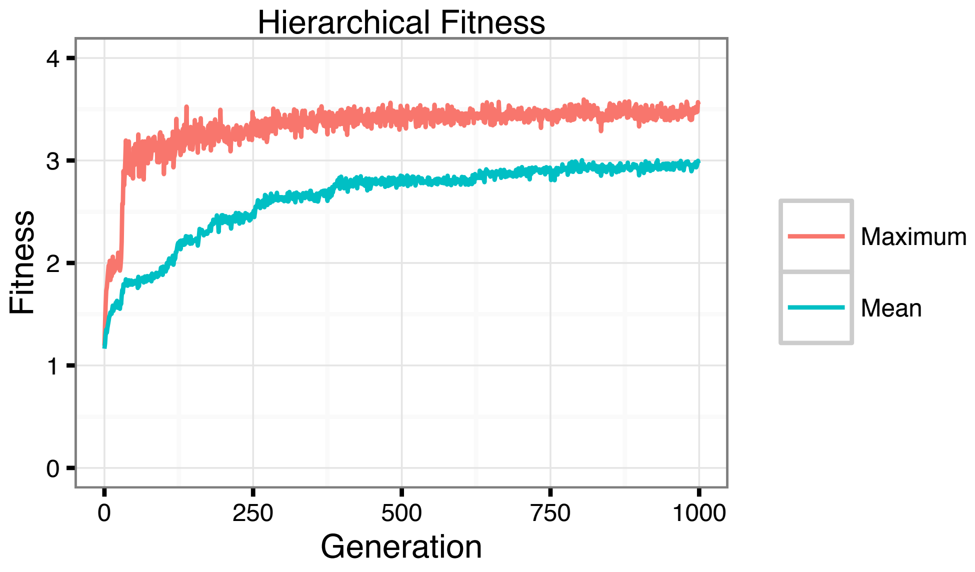


FIGURE 4.4: The average fitness trajectory of each of the highest scoring controllers of all ten evolutionary runs, and the fitness trajectory of the highest scoring controller for the complete rescue task.

4.2.3 Transfer to the Real Robot

We tested each sub-controller incrementally on the real robot, which enabled us to ensure the transferability of the complete controller. After evaluating all the different evolutionary runs of the complete controller, we tested the highest performing controller from the simulation on a real e-puck. The robot had to solve the complete rescue task: find the exit of the initial room, navigate to the correct branch of the double T-maze, and return to the room. We used a room with a size of 120 cm $\times$ 60 cm for our real-robot experiments. Three identical obstacles with side lengths of 17.5 cm and 11 cm were placed in the room as shown in Figure 4.2. We sampled the controllers six times for each light combination, resulting in a total of 24 samples.

To avoid interference between the two robots during the systematic testing of the controllers in real hardware, we excluded the teammate in the real-robot experiments, and manually triggered the near-robot sensor when the robot reached the correct maze branch. We ran additional proof-of-concept experiments in which we included a teammate that was manually programmed to follow the main robot back to the initial room.

The controller solved the composed task on the real robot in 22 out of 24 samples (a solve rate of 92%). The “Main” behavior arbitrator consistently chose the correct sub-controller at each point of the task, and only failed once in the “Solve Maze” behavior (the robot turned the wrong way in the second intersection of the maze), and once in the “Return to Room” behavior (the robot did not turn at the intersection, ending up in a different maze branch).

4.2.4 Discussion

The experiments and results presented above demonstrate how controllers can be composed in a hierarchical fashion to allow for the evolution of behavioral control for a complex task. For the main behavior arbitrator, we used a fitness function directly derived from the immediate decomposition of the task, that is, we used a fitness function that rewards a controller for activating a valid sub-controller given the current situational context. During evolution, an arbitrator (an ANN) was rewarded for (i) activating the “Exit Room” sub-controller while the robot was in the room, (ii) the “Solve Maze” sub-controller while the robot was in the maze, and (iii) the “Return to Room” sub-controller after the teammate had been located. In this way, we avoid that the complexity of the fitness function increases with the task complexity as sub-controllers are combined. The highest scoring controller in simulation was able to cross the reality gap, achieving a performance on real robotic hardware similar to the performance obtained in simulation.

4.3 Hybrid Controllers

In this section, we explore the evolution of hybrid control systems that can take advantage of manually programmed behavior primitives in a different set of experiments. There are examples in literature where researchers have combined evolved control and manually programmed control, but it has been done in an ad-hoc manner (see for instance Groß et al., 2006). Our approach, on the other hand, allows for a structured integration of learned and manually programmed behavior in a hierarchical and incremental manner. We used the double T-maze task (see Section 4.2) with simple manually programmed behavior primitives ("Follow Wall", "Turn Left", and "Turn Right"). The controller for the complete task is composed of an evolved behavior arbitrator that activates one of the three manually programmed behavior primitives. Two manually programmed primitives take more than one control cycle to complete, namely turning 90° left or right. For such primitives, we used locking behaviors (see Chapter 3).

4.3.1 Experiments and Results

The input layer of the ANN-based behavior arbitrator was composed of six neurons: one for each of the four infrared proximity sensors, and one for each of the two light sensors. The highest value of the output neurons of the neural network determines which one of the three possible manually programmed behaviors is activated: follow wall, turn left or turn right. The complete controller and the results are shown in Figure 4.5.

After the evolutionary process was concluded, we conducted a post-evaluation of the evolved controllers in which the fitness of every controller was sampled 100 times for each of the four possible light configurations. The evolved controllers had an average solve rate of 50% (±31%). A solve rate of 80% or more was achieved by three of the ten controllers, with a solve rate of 93% for the highest scoring controller. The solutions produced in different evolutionary runs were similar. The controllers learned to navigate the T-maze correctly, but some were not able to take advantage of the information from the light flashes to consistently make the correct decisions at the junctions, which caused them to navigate to a wrong maze branch.

The highest scoring controller was tested on a real e-puck 24 times, six for each light configuration. The controller was successful in 22 of the 24 samples (a solve rate of 92%). In both failed samples, the robot turned to the wrong maze branch in the second intersection. These results are comparable to the results obtained in simulation and to the real-robot experiments where the behavior arbitrator had access to evolved behavior primitives (see Section 4.2).

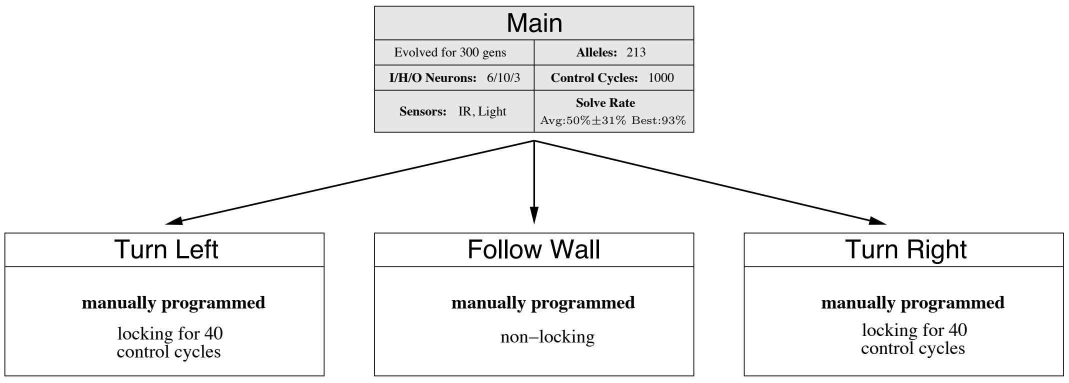


FIGURE 4.5: Hierarchical "Solve T-Maze" Controller. The controller used in our experiments is composed of one behavior arbitrator and three manually programmed behavior primitives. All controllers were manually programmed, with the exception of the main behavior arbitrator. Both the "Turn Left" and the "Turn Right" manually programmed behavior primitives lock the network during execution, in order to ensure that the behavior completes before another primitive can be executed.

4.4 Hierarchical Evolution for Integrated Tasks

In the experiments presented in the previous sections, the behavior primitives were either all evolved, or all manually programmed. Furthermore, in the case of the experiments in Section 4.2, the task was sequential, which meant that the robot had to learn a strict sequence of behaviors: “Exit Room”, “Solve Maze” and “Return to Room”. In this section, we evaluate our approach in a task that is non-sequential and we combine evolved behavior primitives and manually programmed primitives. We use the *offline behavior* technique described Chapter 3, in which a manually programmed behavior is implemented for the real robot, but not in simulation. When the controller activates such a manually programmed behavior in simulation, the normal control cycle is stopped, the end-result of the robot-environment interaction is applied, and controller is then resumed.

We use a task in which the robot must clean dust spots. The dust spots appear in two rooms that are connected by a corridor (see Figure 4.6). The rooms are square-shaped and side lengths vary between 80 cm and 120 cm drawn from a uniform distribution. A new dust spot is randomly placed in one of the two rooms every 10 seconds. The placement of a new spot is determined by a Bernoulli trial with a probability p that the spot is placed in one room and $1 - p$ that it is placed in the other room. The probability p is randomly sampled from the uniform distribution at the beginning of each trial. A maximum of five dust spots can be in the environment at any given time. This means that if the robot keeps cleaning one room for a long time, all dust spots will eventually be in the other room.

In order to traverse the corridor connecting the two rooms, the robot must first push a button to open both doors (see Figure 4.6). We manually programmed a behavior primitive to enable an e-puck to push a button, which opens the doors to the corridor. Pushing a button to open the doors requires fine sensorimotor coordination, since the buttons are difficult to detect and hit. The buttons are only 2.5 cm in diameter, and they must be pushed at an angle under 45°. This is a difficult interaction to model correctly in simulation, and it can also be a challenging behavior to evolve and transfer successfully. The manually programmed

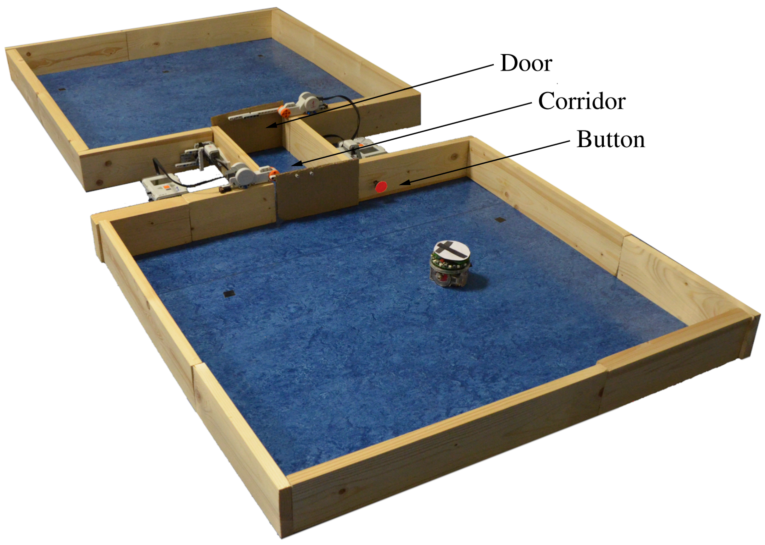


FIGURE 4.6: The environment is composed of two rooms, connected by a corridor. The corridor is blocked by two doors that the robot can open by pushing a red button.

push button behavior was therefore only implemented for the real robot, and activating the manually programmed behavior automatically opens the door instantly in simulation. On the real robot, the manually programmed behavior uses the e-puck's on-board camera to find and move the robot toward the button. When the manually programmed behavior primitive that opens the door is activated, the control cycle of the main behavior arbitrator is stopped. The manually programmed behavior primitive rotates the robot up to a maximum of 360° in order to try to locate the button, which can be identified by its red color. The central horizontal line of each image captured by the e-puck's camera is scanned. If the button is identified, the robot aligns itself, moves forward, and pushes it. The main behavior arbitrator is resumed after the manually programmed behavior terminates. When the manually programmed behavior is activated in simulation, the elapsed time of the current sample is increased by the average amount of time taken by the manually programmed push button behavior on the real robot.

4.4.1 Experiments and Results

In the real-robot experiments, virtual dust spots are implemented using a visual tracking system. The position and orientation of the robot is tracked using an overhead camera. The robot has a cross-shaped marker on top (see Figure 4.6), which is extracted from the video feed using the OpenCV library (Bradski, 2000). This information allows virtual dust spots to be detected by the robot using virtual sensors. The controller is executed in a workstation, and the resulting movement commands are sent to the robot every 100 ms. The robot is able to sense the dust spots up to 30 cm away with eight virtual sensors, positioned on the perimeter of the robot's body at equal angular intervals. The robot must activate a virtual actuator when it is within 5 cm of a dust spot to remove the spot. The robot is also equipped with two button sensors that are placed at angles of $-30°$ and $30°$, providing the controller with information on the direction and distance (up to 1 m) to the nearest button. The controller we used for these experiments can be seen in Figure 4.7. We decomposed the task into two main sub-controllers: "Change Room" and "Clean". The "Change Room" sub-controller is a behavior arbitrator that can choose between the "Open Door" behavior arbitrator and the "Enter Corridor" behavior primitive.

The robot must push a button to open the doors to the corridor before it can move from one room to the other. Once the doors have been opened, the robot has 40 seconds to traverse the corridor that leads to the other room before the doors close. We divided the "Change Room" sub-controller into a behavior primitive and a behavior arbitrator: the "Open Door" behavior arbitrator, which moves the robot toward the current room's button ("Move to Button" behavior primitive) and pushes the button ("Push Button" manually programmed behavior primitive), and the "Enter Corridor" behavior primitive, which navigates to the corridor and crosses to the other room.

We first evolved the "Move to Button" behavior primitive. The controller was evaluated by f_{two_rooms} (see Equation 4.5) according to a distance gradient to

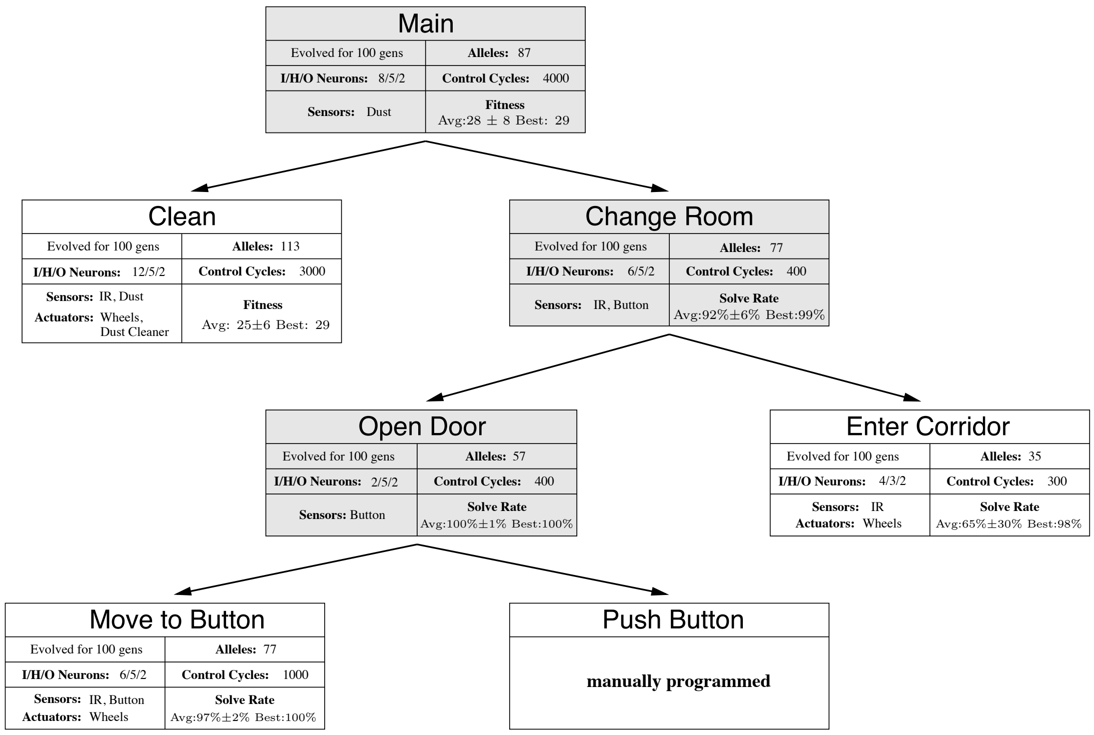


FIGURE 4.7: Hierarchical Dust Cleaning Controller. The controller synthesized for the dust cleaning experiment is composed of three behavior arbitrators and four behavior primitives.

the button, with a time-dependent bonus upon correctly activating the manually programmed behavior primitive.

$$f_{two\_rooms} = \begin{cases} 1 + 10 \cdot \frac{C-c}{C} & \text{, if achieved objective} \\ \frac{D-d}{D} & \text{, otherwise} \end{cases} \tag{4.5}$$

where C is the maximum number of control cycles, c is the number of cycles spent, D is the distance from the robot's starting point to the button, and d is the closest point to the button that the robot reached. In the case of the "Move to Button" behavior primitive, the objective was to get within 5 cm of the button.

The "Open Door" behavior arbitrator has access to the button sensor. In simulation, the manually programmed "Push Button" behavior opens the doors if the robot is within 20 cm of the button, otherwise the robot stops moving while the behavior is activated. Since the push button behavior takes some time to execute on the real hardware, controllers that activate this behavior too often or too far from the button are therefore indirectly penalized because they have less time to clean. On the real robot, such controllers would make the robot search for the button at a distance that would make it very difficult for the e-puck's camera to detect a small target. The controllers were evaluated by f_{two_rooms} (see Equation 4.5), where the objective was to open the corridor doors. Since the button was placed to the right-hand side of each door, we reused the "Turn Right" behavior primitive from Section 4.2 instead of evolving a new primitive for the "Enter Corridor" sub-controller. This behavior primitive allows the robot to move to the other room after pushing the button.

For the evolution of the "Change Room" behavior arbitrator, the robot was positioned and oriented randomly in one of the rooms at the beginning of each trial, and had to push the button, enter the corridor and move to the other room. Each controller was also evaluated according to f_{two_rooms}, where the objective was to reach the other room. For the "Change Room" arbitrator, D is the distance from the robot's starting point to the end of the corridor, and d is the point closest to the end of the corridor that the robot reached.

The "Clean" behavior primitive was evolved in a single room without any door or exit. The output neurons controlled the speed of the robot's wheels, and the *clean dust spot* actuator was triggered if its corresponding output had an activation value higher than 0.5. If no dust spot is nearby when the robot activates the *clean dust spot* actuator, the speed of the robot is set to 0. This penalizes the controller from always activating the actuator, instead of only activating it near a dust spot. The robot was randomly oriented and positioned near the center of the room at the beginning of each sample. The fitness function rewarded the robot based on how many dust spots it cleaned (a score of 1 for each dust spot) in the allotted time or until it collided with a wall.

To obtain a controller for the complete task, the "Clean" behavior primitive and the "Change Room" behavior arbitrator were combined via the evolution of a new behavior arbitrator (see Figure 4.7). All evolutionary runs converged to a behavior where the robot would move between rooms whenever necessary. The main arbitrator evolved behaviors that activated the change room behavior arbitrator if the robot did not sense any dust spot for a certain amount of time. The controllers were evaluated according to the number of dust spots they cleaned, as in the evolution of the "Clean" behavior primitive. The controllers achieved an average fitness of 28 ± 8 in the ten evolutionary runs, and the highest-performing controller achieved an average of 29 ± 6. The controllers chose to cross to the other room an average of 2.86 times per sample.

4.4.2 Real-Robot Experiments

For our real-robot experiments, we built the walls of the two rooms and the corridor using wooden blocks. Each room had a size of 100 cm $\times$ 100 cm, and the corridor had a length of 40 cm and a width of 18 cm (see Figure 4.6). Both doors were opened and closed by motors connected to a Lego Mindstorms NXT brick. A physical button was placed on the wall to the right of the entrance to the corridor in each room.

We transferred the highest scoring controller from simulation to the real e-puck. The performance of the controller on the real robot was sampled five times in five different configurations, for a total of 25 samples. We used fixed rates at which the dust spots were placed in each room, in a combination of average cases and extreme cases to compare performance. In each configuration, the rate of a dust spot being placed in each room was set to one of the following: $1:0$, $3:1$, $1:1$, $1:3$, and $0:1$. In the $3:1$ scenario, for instance, three dust spots are placed in the room where the robot starts, and then one dust spot is placed in the other room, and so on. The dust spots were placed deterministically at these fixed rates in order to allow for direct comparison of performance in simulation and performance in real hardware. We sampled the robot's performance in simulation 100 times per configuration. Each sample lasted five minutes (3000 control cycles). The results can be seen in Figure 4.8. In the real-robot experiments, the controller was able to complete the task with a performance comparable to that obtained in simulation.

Several outliers can be observed in the 0 : 1 configuration. In this configuration, the robot starts in a room where no dust spots are ever placed. After some time, the controller decides to cross to the other room. In most of the samples, the robot keeps cleaning the room in which all the dust spots appear, but sometimes the controller might choose to return to the initial room. This happens when several dust spots are placed in a cluster and the robot does not explore the part of the room containing the cluster for a certain amount of time. In such cases, the number of dust spots that the robot can clean are significantly lower than if the robot remains in the room in which dust spots appear. The robot wrongly returned to the initial room in one of the real-robot samples. As a result the robot only cleaned 13 dust spots in the 0 : 1 experimental setup, and received a low fitness (see outlier for the 0 : 1 setup in Figure 4.8).

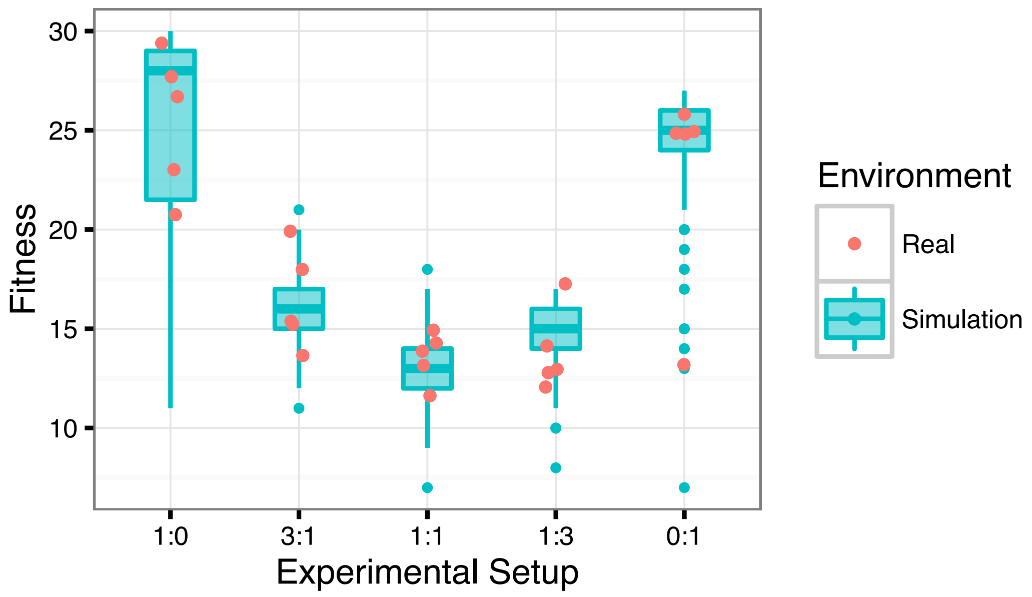


Figure 4.8: Results of the real-robot experiments in the dust cleaning experimental setup. The box plots represent 100 samples in simulation, while the scatter plots represent the fitness obtained in the real-robot experiments. The whiskers extend to the most extreme data point within 1.5× the interquartile range.

4.4.3 Discussion

In this section, we introduced a non-sequential task that required fine sensorimotor coordination. A hybrid approach in which evolved behaviors are combined with manually programmed behaviors can be beneficial when some behaviors are too difficult to simulate accurately. In the case of pushing the button, we did not implement the manually programmed behavior in simulation. When the controllers were being evolved, the doors in the environment would open automatically if the robot activated the push button behavior primitive near the button. On the real robot, activating that same behavior primitive triggered a manually programmed behavior, which used the robot's camera to detect and push a physical button in the environment.

In the experiment, we reused a previously evolved behavior primitive from a different experiment. The turn right behavior primitive from Section 4.2 was used to navigate from one room to the other. In this case, the behavior could be reused because of the characteristics of both the task and the environment: the button was positioned to the right of the door, and the robot had to turn right to enter the corridor after pushing the button. If the position of the button was unknown, it would have been necessary to evolve a new behavior for the sub-task. While classic evolutionary approaches force the designer to evolve new controllers for similar tasks, our approach, allows for the reuse of previously evolved or manually programmed behaviors.

4.5 Summary

The application of classic evolutionary robotics techniques to the synthesis of controllers for complex tasks has proven problematic: the evolutionary process is often difficult to bootstrap and vulnerable to deception when the task is difficult. Successful transfer of evolved control from simulation to real hardware also remains a challenging problem. In this chapter, we demonstrated how hierarchical composition of robotic controllers can overcome these issues. We recursively divide the goal task into sub-tasks until an appropriate fitness function has been found or until a behavior can easily be programmed by hand. In this way, partial solutions can be found incrementally and successful transfer to real robotic hardware can be guaranteed at each increment. Controllers for complex tasks can thus be synthesized in a hierarchical fashion while, at the same time, they can benefit from evolutionary robotics techniques, namely (i) automatically synthesis of control, and (ii) evolution's ability to exploit the way in which the world is perceived through the robot's (often limited) sensors.

The transfer of behavioral control from simulation to a real robot is usually a hit or miss because a controller for the goal task is completely evolved in simulation before any tests are conducted on real hardware. In our approach, the transfer from simulation to real robotic hardware can be conducted in an incremental manner as sub-controllers are evolved. Crossing the reality gap one sub-controller at a time allows the designer to address issues related to transferability immediately and locally in the controller hierarchy. The fact that each sub-controller solves only part of the task also allows for the use of different types of noise and other sub-task specific configurations in simulation, as well as the use of different evolutionary algorithms, such as novelty search (Lehman and Stanley, 2011), fitness-based evolution (Floreano and Mondada, 1996), and particle swarm optimization (Clerc, 2010).

Chapter 5

Synthesis of Hierarchical Control for Swarm Robotic Systems

Artificial evolution has been proposed as a potential solution to the problem of control synthesis for SRS. Evolved control for robotic swarms has been demonstrated in a wide variety of tasks, but most studies are carried out only in simulation. When experiments are validated in real robotic hardware, they are confined to controlled laboratory conditions (Brambilla et al., 2013). While SRS with evolved control have the potential to be applied in real-world tasks, state-of-the-art studies are thus still conducted under simplistic and unrealistic constraints. This disparity can arguably be attributed both to open issues in the field (such as the reality gap) and to the lack of focus of the swarm robotics research community in applying the current state of the art in progressively realistic scenarios.

In this chapter, we apply our hierarchical control synthesis approach to a swarm of aquatic robots. Our experiments are the first demonstration of a SRS system with evolved, distributed control performing self-organized behaviors in a real-world environment. Our experiments rely on a swarm of up to ten small, simple, and inexpensive aquatic surface robots (Costa et al., 2016). Using our simulation framework and our real robotic swarm, we first study the fundamental issues related to evolution of control for swarms of aquatic surface robots. Then, we apply our hierarchical control synthesis approach in order to use the SRS for complex tasks that require a set of different behaviors.

While SRS are typically associated with ground robots (Dorigo et al., 2013; Brambilla et al., 2013) and, more recently, with aerial robots (Lindsey et al., 2012; Basiri et al., 2014), we show that SRS also have the potential to be applied to maritime tasks, which are usually expensive to carry out because they rely on manned vehicles with large operational crews (Lutterbeck, 2006). While significant effort has been made to adapt unmanned vehicle technology to maritime tasks, such systems are currently relatively expensive to acquire and operate, and only a single or a few robots are typically deployed (Yan et al., 2010). The use of SRS at sea is advantageous because: (i) several maritime tasks, such as environmental monitoring, sea-life localization, and sea-border patrolling, require distributed sensing at a high spatial and temporal resolution, and are therefore challenging to carry out with one or few robots only; and (ii) robots operating at sea need to display a high degree of fault-tolerance and robustness (Yan et al., 2010), which can be provided by SRS approaches (Şahin, 2005).

The outline of this chapter is the following. Firstly, we demonstrate our SRS performing four canonical tasks in an uncontrolled aquatic environment: (i) homing, (ii) dispersion, (iii) clustering, and (iv) monitoring. Secondly, using a subset of the controllers, we setup a set of experiments to assess the scalability and robustness of the evolved control. Thirdly, we apply our hierarchical control synthesis approach to an environmental monitoring task and an intruder detection task. Finally, we show that the approach scales to large-scale swarms, with up to 1000 robots.

5.1 Experimental Setup

In this section, we present the methodology for the evolution of control, the evolutionary setups of our experiments, the simulation platform, and the real robot platform that was used to evaluate the controllers. To synthesize the behavioral control, we followed a classic evolutionary robotics approach in which the evolutionary process relies exclusively on simulation for the evaluation of candidate solutions and the highest performing solutions are then transferred to real robots after the evolutionary process terminated.

For all tasks, the following general methodology was applied to synthesize controllers and assess their performance on real robots (see Figure 5.1):

1. The task was defined by specifying the fitness function, which translated the task objective that needed to be achieved. The robots' sensor configuration and the task conditions were also specified.

2. The evolutionary algorithm optimized the neural network-based controllers for the robots. Each controller was evaluated in 10 independent simulation trials, and the fitness of the controller was the mean score obtained in these simulations. In every trial, several parameters were randomly varied: the number of robots (between five and ten), the starting position and orientation of the robots, the sensor and actuator noise parameters, and the drift speed and orientation.

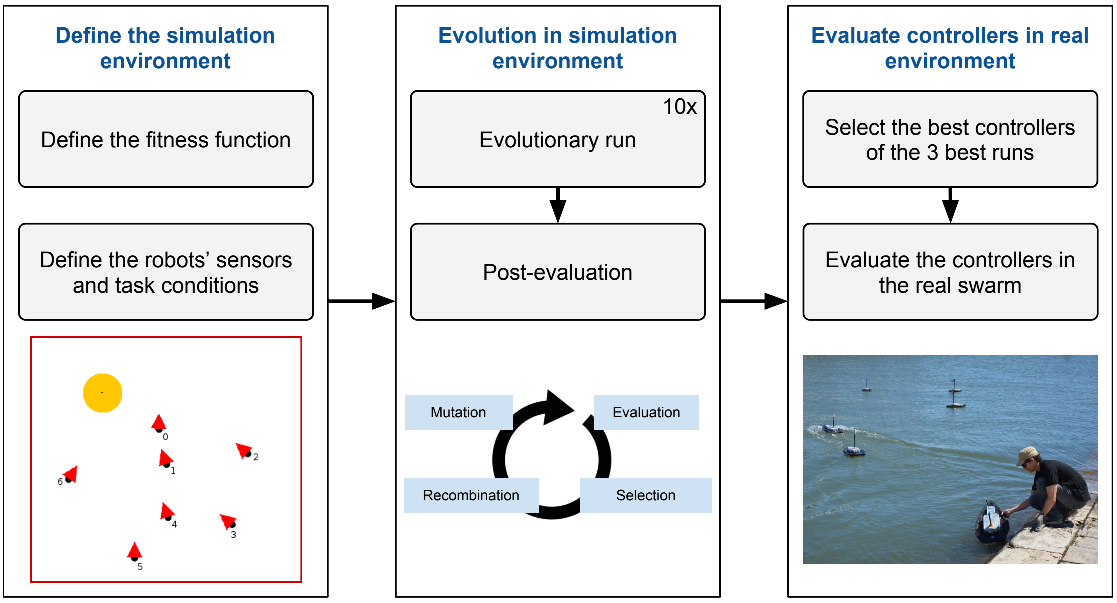

Define the simulation environment
Define the fitness function
Define the robots' sensors and task conditions
Evolution in simulation environment
10x
Evolutionary run
Post-evaluation
Mutation
Evaluation
Recombination
Selection
Evaluate controllers in real environment
Select the best controllers of the 3 best runs
Evaluate the controllers in the real swarm

FIGURE 5.1: The control synthesis and performance assessment process.

3. The same evolutionary setup was repeated for 10 independent evolutionary runs, each with a different initial random seed.

4. After each run, we post-evaluated the top controller of each generation in 100 simulations.

5. We selected the top three controllers that achieved the highest fitness scores in post-evaluation.

6. Using the same set of initial conditions for all controllers, we assessed the real-robot performance of the three controllers, and compared their performance to that obtained in simulation.

In the rest of this section, we present: (i) the hardware components and physical characteristics of the robotic platform, (ii) the sensors and actuators used, (iii) the simulation platform, and (iv) the evolutionary setup.

5.1.1 Robotic Platform

We designed and produced a total of 10 simple, small (65 cm in length) and inexpensive ($\approx$ 300 EUR/unit) robots, using widely available hardware and off-the-shelf sensors and motors. Each robot model is a differential drive mono-hull boat (see Figure 5.2). The maximum speed of the robot in straight line is 1.7 m/s (3.3 kts), and the maximum angular velocity is 90°/s. The on-board control of each robot is supported by a Raspberry Pi 2 single-board computer (SBC). Robots communicate through Wi-Fi by broadcasting UDP datagrams, which are received by the neighboring robots and the monitoring station. The robots form a distributed network without any central coordination or single point of failure. Each robot is equipped with GPS and compass sensors, and broadcasts its position to neighboring robots every second. A detailed description of the robotic platform is available in Appendix B.

Each robot is controlled by an artificial neural network-based controller. The inputs of the neural network are the normalized readings of the sensors, and the

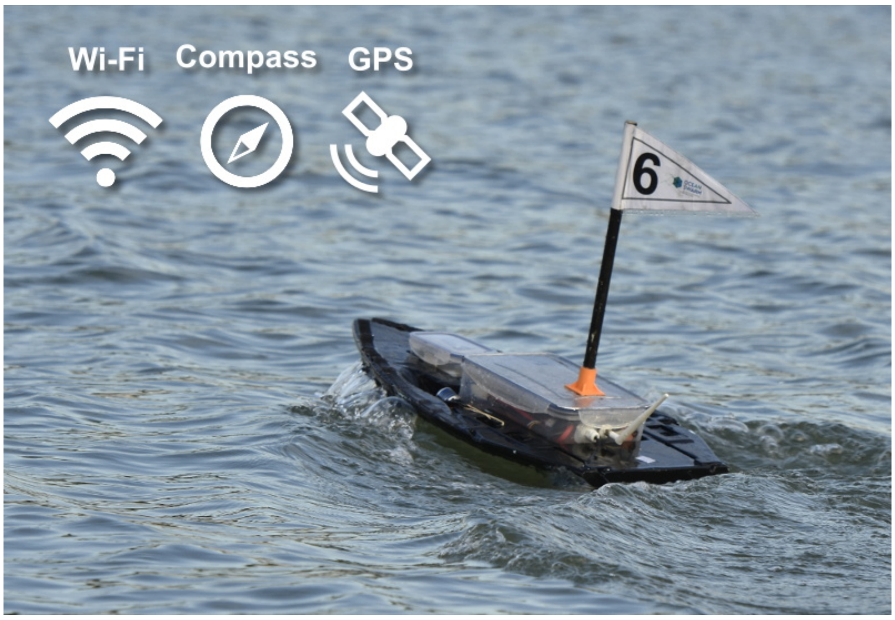


FIGURE 5.2: The robot is an autonomous surface vehicle equipped with Wi-Fi for communication, and a compass and GPS for navigation. It has a length of 60 cm and can move at speeds of up to 1.7 m/s.

outputs of the network control the robot's actuators. The sensor readings and actuation values are updated every 100 ms. The neural network controlling each robot has two actuators, which control respectively the linear speed and the angular velocity. These two values are converted to left and right motor speeds. We implemented three *emulated sensors* for the detection of points and objects of interest in the task environment. The emulated sensor values are obtained by pre-processing the GPS location of the entities in the environment that the robot is currently aware of, and the current heading and position of the robot, as given by the GPS and compass. This pre-processing is conducted onboard each robot, and the resulting sensory inputs are indistinguishable to the controller from any other type of sensor. The following emulated sensors were implemented (see Figure 5.3):

Waypoint sensor: This sensor is used to locate waypoints in the environment, if any. The sensor returns two values: (i) the relative angle from the robot to the waypoint, and (ii) the distance from the robot to the waypoint.

Robot sensor: This sensor measures the distance to nearby robots. The area around the robot is divided into four equally-sized slices. The sensor returns four values, one per slice, linearly proportional to the distance between the robot and the closest neighbor in that slice, or the maximum value if there is none within the sensor range.

Geo-fence sensor: This sensor is used to detect the virtual geo-fence. Similarly to the robot sensor, the geo-fence sensor returns four values, one for each slice around the robot, indicating the minimum distance to the fence, or the maximum sensor value if the fence is outside the sensing range. One additional value is returned, indicating whether the robot is inside the fence or not (the geo-fence is a polygon).

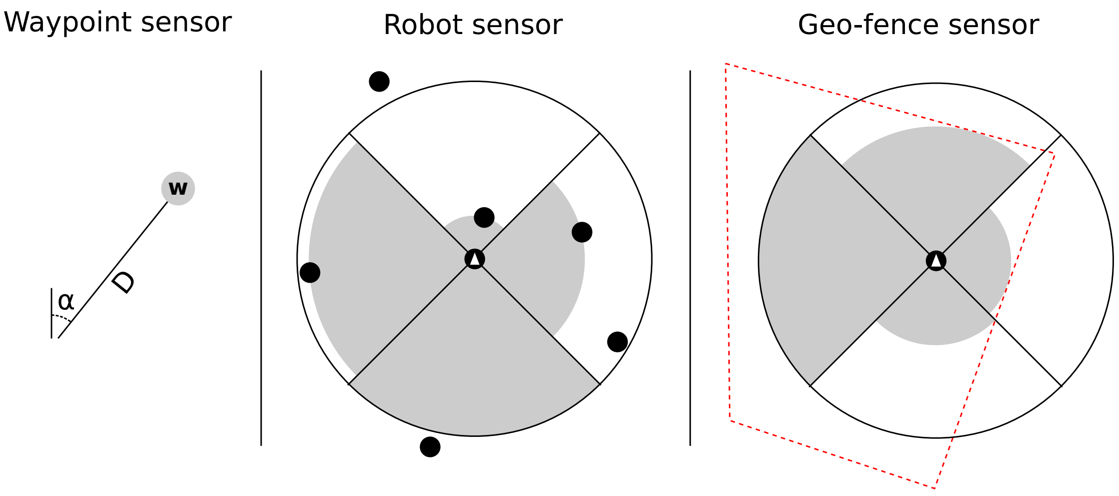


FIGURE 5.3: Illustration of the three types of emulated sensors.

5.1.2 Simulation Model

The controllers for the SRS were synthesized offline, in simulation. We modeled the robots in JBotEvolver, and developed a middle-layer that is shared by both the hardware platform (Raspberry Pi) on-board control software and the simulation in JBotEvolver, enabling the same code-base and controllers to be executed on both the real robots and on robots in simulation.

We used a two-dimensional simulation environment, and the robots were abstracted as circular objects with a certain heading and position. The robot's dynamics were modeled based on data taken from a single robot in a fresh water environment with no currents. We measured the robot's movement in a variety of scenarios, applying multiple combinations of motor speeds, which allowed us to characterize the motion dynamics and main friction components. The general principle behind the simulation was to model the motion of the robots based on real measurements taken in the water, but without including physics simulation and fluid dynamics, which would be complex and computationally expensive. Maintaining a low computational complexity is essential for making the evolutionary robotics process feasible, as a large number of simulations have to be conducted for each evolutionary run.

To compensate for the differences between the motion model and the real dynamics of the robots, as well as for other potential sources of stochasticity (GPS/compass errors or drifts, varying motor performance due to battery level, etc), we introduced a conservative amount of actuator noise during evolution. As previous works have shown (Miglino et al., 1996; Jakobi, 1997), the use of noise in simulation promotes the evolution of general and adaptive solutions, therefore increasing the performance of transferred solutions. The introduction of noise is a computationally-effective way of promoting the evolution of robust controllers, since it becomes more difficult to exploit particular features of the simulator. Random noise and random offsets were added to the sensors, actuators, and even the environment in the form of drift. The amount of noise was chosen based on the values observed in the real hardware (Miglino et al., 1996).

During evolution we also introduced a drift with a random magnitude in the range [0,1] m/s, and a random direction drawn from uniform distributions at the beginning of each simulation sample. The drift constantly affected the position of the robots, and was included to account for the unpredictable drift caused by wind and currents in real aquatic environments. A description of all different types of noise used in simulation can be found in Appendix C.

5.1.3 Evolutionary Setup

Our initial set of experiments focuses on four tasks, which have been previously studied both in simulation and on real robots (but only in strictly controlled laboratory conditions) (Bayındır, 2016): homing in group while avoiding collisions, dispersion, clustering, and monitoring of a pre-defined area. All of the chosen collective tasks can be classified as swarm robotics tasks, according to the classification proposed by Brambilla et al. (2013): (i) robots are autonomous; (ii) robots are situated in the environment; (iii) robots' sensing and communication capabilities are local; (iv) robots do not have access to centralized control and/or to global knowledge; and (v) robots cooperate to solve a given task.

In this section, we describe the four tasks we focused on, and the evolutionary setup that was used to synthesize control for each task, including the fitness functions and relevant parameters. We used NEAT (Stanley and Miikkulainen, 2002) to evolve the neural network controllers for all tasks in this chapter. A detailed listing of the different parameters used for each evolutionary setup can be found in Appendix C.

5.1.3.1 Homing

Navigation and obstacle avoidance is an essential feature for autonomous robots operating in the real world. This type of tasks was among the first to be studied in the field of evolutionary robotics (Floreano and Mondada, 1996). A variation of the task, known as homing, involves one or more robots moving towards a point of interest in the environment, typically with obstacles in the way (Christensen and Dorigo, 2006a).

In the homing task used in our experiments, the swarm of robots had to navigate to a given waypoint while avoiding collisions among the robots, regardless of the size of the swarm. To evolve solutions for this task, we rewarded controllers for minimizing the average distance of the swarm to a waypoint, according to the following equation:

$$f_{homing} = \left(\frac{1}{T} \sum_{t=1}^{T} \frac{1}{R} \sum_{r=1}^{R} \frac{startingDist_r - dist_{r,t}}{startingDist_r} \right) \times S \ , \tag{5.1}$$

where T is the maximum number of time steps, R is the number of robots, $startingDist_r$ is the initial distance from robot r to the waypoint, and $dist_{r_t}$ is the distance of robot r to the waypoint at time t.

Collisions between robots are undesirable in any task, as they can damage the real robots. To avoid collisions, we introduced a *safety coefficient* (S) when assessing the fitness in all task setups, which penalized solutions where robots get too close to one another (less than 3 m). The safety coefficient S is in the range [0.1,1], and is inversely proportional to the minimum distance between any two robots in the current simulation trial ($minDist$). The safety coefficient is defined according to the following equation:

$$S = 0.1 + \frac{max(0, min(3, minDist))}{3} \times 0.9 \ , \tag{5.2}$$

5.1.3.2 Dispersion

In a dispersion task, each individual robot in the swarm has to maintain a predefined distance (*target distance*) from its nearest neighbor. Robots should cover a large area without risking losing contact with the rest of the swarm, which can be an issue in unbounded environments. Examples in the literature include solutions that rely on manually programmed behaviors (Batalin and Sukhatme, 2002), potential fields (Reif and Wang, 1999), and automatically synthesized state-machines (Francesca et al., 2014).

For this task, we chose a target distance of 20 m (half of the robots' communication range). During evolution, controllers were rewarded for minimizing the difference between the current distance to the nearest neighbor and the target distance, according to the following equation:

$$f_{dispersion} = \left(\frac{1}{T} \sum_{t=1}^{T} \frac{1}{R} \sum_{r=1}^{R} 1 - |minDist_{r,t} - targetDist| \right) \times S \ , \qquad (5.3)$$

where T is the maximum number of time steps, R is the number of robots, $minDist_{r_t}$ is the distance of robot r to the nearest neighbor at time t, and *targetDist* is the distance at which the robots should disperse.

5.1.3.3 Clustering

Clustering, also known as aggregation, is a canonical task in swarm robotics (Brambilla et al., 2013). Clustering is a challenging task because it combines several aspects of multirobot tasks (Silva et al., 2015b), including distributed individual search, coordinated movement, and cooperation. Furthermore, clustering plays an important role in robotics because it is a precursor of other collective behaviors such as group transport of heavy objects (Groß and Dorigo, 2009).

In our instance of the clustering task, the robots started randomly spread in a square-shaped area with a side-length of 100 m, and had the objective of finding one another so as to form a single cluster. However, since the communication range was limited and the environment was unbounded, certain initial conditions could lead to more than one cluster. During evolution, controllers were scored based on the number of clusters that they formed. Two robots belonged to the same cluster if the distance between them was inferior to 7 m. The fitness function was defined based on a weighted sum of the number of clusters throughout time:

$$f_{clustering} = \frac{\sum_{t=1}^{T} t \times \frac{R-c_t}{R-1}}{\sum_{t=1}^{T} t} \times S \ , \qquad (5.4)$$

where T is the total number of time steps in each trial, R is the number of robots, and c_t is the number of clusters formed by the robots at step t.

5.1.3.4 Area Monitoring

Swarms of robots are ideal for tasks where large areas need to be covered for monitoring or surveillance purposes. This task category has been explored in the past with several different control techniques, such as pheromone traces (Wagner et al., 1999), odor sources (Marques et al., 2006) and evolved artificial neural networks (Haasdijk et al., 2011).

In our area monitoring task, a geo-fence is defined to delimit an area of interest, and the robots should coordinate to continuously cover as much of the area as possible. During evolution, we tested the controllers in a variety of randomly generated geo-fences with different shapes, in order to obtain general behaviors. For the purpose of this task, we considered that each robot covers a circular area with a radius of 5 m (V) around it. The challenge in this task is to find a general movement pattern that takes into account many different shapes of the monitoring area and varying number of robots.

In order to assess the performance of the controllers, we divided the monitoring area into a grid with a fixed cell size of 1 m^2. Each robot could visit cells within the coverage radius V, setting its value to 1. The value of previously visited cells decayed linearly over a time frame of 100 s, down to 0. Controllers were scored based on how much of the grid was covered over time, according to the following equation:

$$f_{monitor} = \left(\frac{1}{T} \sum_{t=1}^{T} \frac{1}{C} \sum_{c=1}^{C} val(c_t) \right) \times S \ ,$$

$$val(c_t) = \begin{cases} 0 & , \ t = 1 \\ max(0, val(c_{t-1}) - \Delta) & , \ minDist_{c,t} > V \\ 1 & , \ minDist_{c,t} \leq V \end{cases} \tag{5.5}$$

where T is the maximum number of time steps, R is the number of robots, C is the number of cells, $minDist_{c,t}$ is the minimum distance of any robot to cell c at time t, and Δ is the decay for any cell (0.001 per time step).

5.1.4 Evolutionary Results

The results of the evolutionary algorithm for all tasks can be found in Figure 5.4. Solutions for homing, dispersion and monitoring were found within 100 generations, while clustering required up to 400 generations for good solutions to evolve. This can be explained by the challenge in forming a single cluster, depending on the initial positions of the robots: since the environment is unbounded, some robots might find themselves outside of each other's communication range, making the task more difficult.

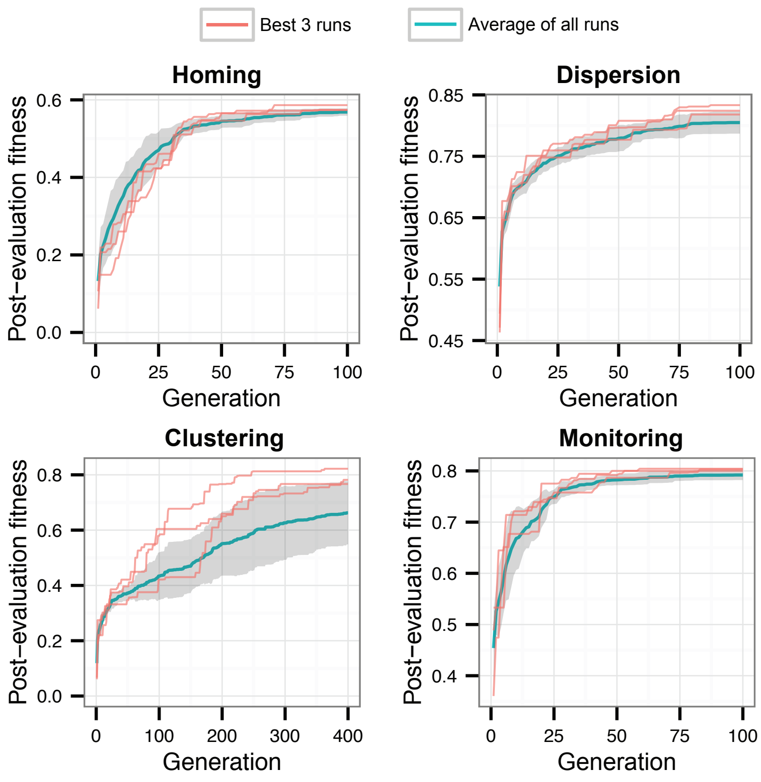


FIGURE 5.4: Fitness plot for the four different tasks. The plot shows the highest fitness scores found so far at each generation. The red lines depict the three highest-scoring evolutionary runs, while the blue line depicts the average of the ten runs, with the respective standard deviation shown in gray.

For the most part, the definition and configuration of the evolutionary process was straightforward. We relied on the NEAT algorithm, which does not require the specification of the neural network topology for the controllers. Fine-tuning the parameters of NEAT was also not necessary, as the default parameters yielded good results. The only challenge in configuring the evolutionary algorithm was the definition of the fitness functions. Defining a fitness function for a given task was typically done within two or three iterations, usually due to unforeseen local optima that only manifested themselves throughout evolution. The fact that we used common swarm robotics tasks also facilitated the definition of the fitness functions, as we could rely on fitness functions proposed in previous studies.

5.2 Transferring Controllers to Real Robots

In this section, we assess the evolved controllers on real robots and compare their performance on real robots to the performance in simulation. The experiments reported in this section were conducted over the course of four days, at Parque das Nações, Lisbon, Portugal, in a semi-enclosed area in the margin of the Tagus river, see Figure 5.5. The wind speed ranged from 15 km/h to 40 km/h. The average drift speed of the robots was 0.2 m/s, twice as high as we had used in simulation (see Section 5.1.2).

For each tasks, the top three controllers were tested (referred to as Controllers 1, 2 and 3 in this section). Each controller was evaluated in three independent experiments (referred to as samples A, B and C in this section). For every sample, the initial positions of the robots were randomized: a set of positions was generated inside a user-defined region, and each robot navigated to the respective position. When all robots were in place, we started the experiment by activating the controller in all robots simultaneously. When evaluating the three different controllers for a given task, the same set of initial positions was used, ensuring that all controllers were tested under similar initial conditions. Each sample ended after a fixed amount of time.

5.2.1 Homing

We setup a scenario where the swarm had to navigate sequentially to four waypoints to assess the performance of the evolved homing controllers. Each waypoint was placed at a distance of 40 m from the previous one, and the active waypoint was changed periodically every 60 s. Each controller was tested once in this scenario, for a total of 4 minutes. All three controllers tested in the real robots were able to navigate successfully to the waypoint while avoiding collisions with the neighboring robots. Once the swarm arrived at one of the waypoints, the robots would start moving around the waypoint at slow speeds in concentric circles (see Fig 5.6, bottom).

Figure 5.6 (top) shows the average distance to the active waypoint, compared to the same controller in the simulation environment. The controllers display a similar behavior and level of performance in reality and in simulation, arriving at the intermediate waypoints at approximately the same time, and keeping the same average distance to them (around 10 m). Controller 3 was the exception, reaching the waypoints faster than the other controllers, and performing better on the real robots than in simulation.

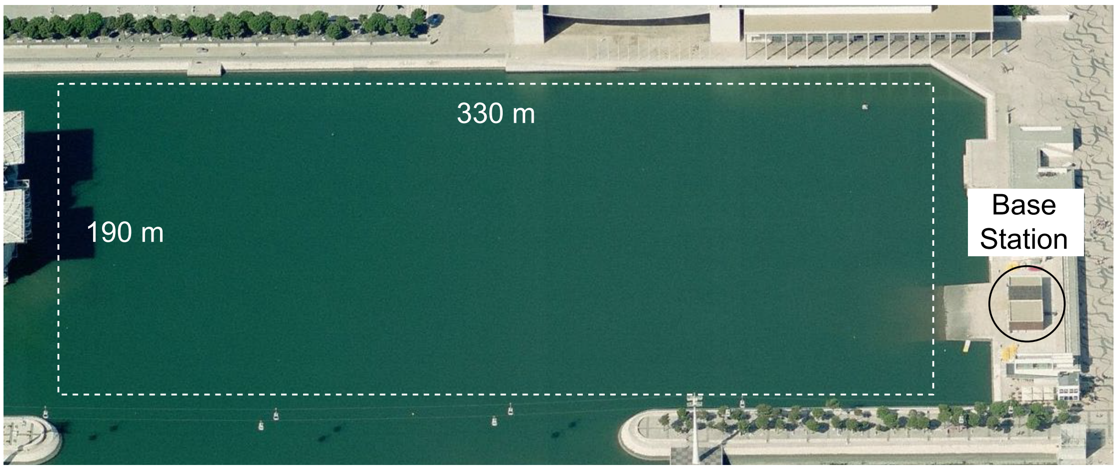


FIGURE 5.5: Aerial photograph of the location of our experiments at Parque das Nações, Lisbon, Portugal. The waterbody used has an area of 330 m × 190 m, and is connected to the Tagus river.

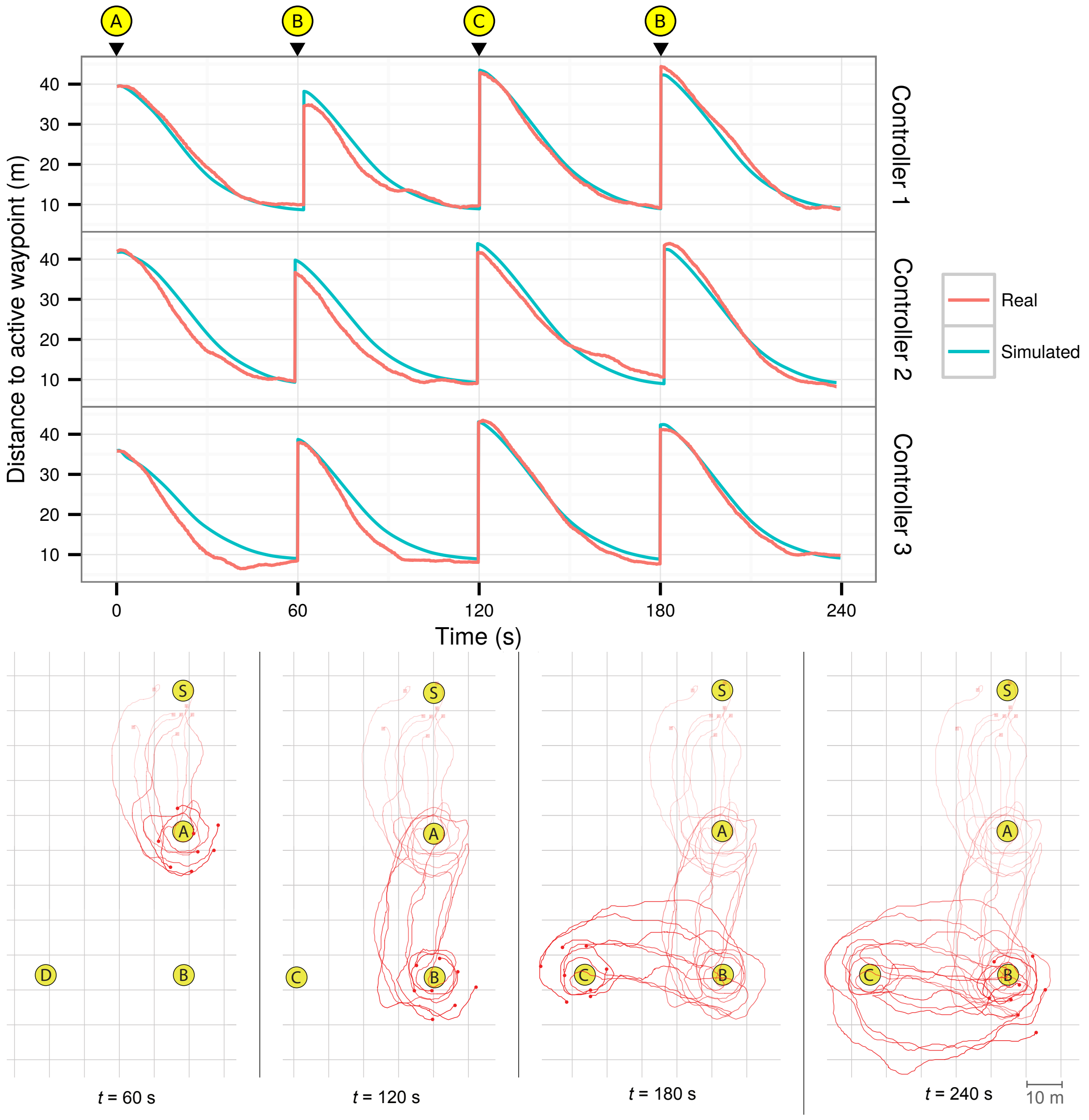


FIGURE 5.6: Real-world homing experiments with eight robots. The robots started around S. The active waypoint was then changed at 60 second intervals, in the order A→B→C→B, for a total of four minutes per experiment. Top: comparison between the real and simulated robots, showing the average distance to the active waypoint, for similar conditions. The top of the figure shows the current active waypoint. Bottom: trajectory traces of the real robots for Controller 3. The waypoints are marked with yellow circles.

5.2.2 Dispersion

In the real-robot dispersion tests, eight robots started in a cluster, and had to disperse to a distance of 20 m from their nearest neighbor. The robots were initially placed randomly in a square-shaped area with a side-length of 28 m, at a minimum distance of 5 m from the nearest neighbor. Each experiment lasted 90 seconds.

The results in Figure 5.7 show some differences in the performance of controllers on the real robots and in simulation. While performance was relatively homogeneous across all controllers in the simulation environment, with an error close to 1 m to the target distance, no controller in the reality was able to achieve the same level of performance on real robots. Controller 3 achieved an average error of 2 m in reality, but the other two controllers performed considerably worse, despite all of them displaying similar performance in simulation. Upon inspection of the behaviors, we observed that the robots in simulation relied on moving in circles at very low speeds to maintain their positions after dispersion. The same movement pattern was observed on the real robots. However, the circles performed by the real robots were larger, meaning that they therefore tended to move away from their position, which is explained by the differences between the motion model in simulation and the motion of the real robots.

5.2.3 Clustering

The clustering controllers were tested by randomly placing the real robots in an area of 100x100 m, up to a maximum of 40 m from the nearest robot. Each experiment lasted 180 seconds. Results for the clustering experiments greatly varied depending on the initial conditions, but this variation was also observed in the simulation environment (see Figure 5.8, top). If one or more robots were initially positioned far away from the remaining robots (and therefore not within communication range), the swarm would sometimes aggregate in more than one cluster, see example in Figure 5.8 (bottom, Controller 2).

The results show that the controllers were generally able to cross the reality gap successfully. In the case of Controller 1, it had a better performance in the real-robot experiments than in simulation, by successfully aggregating in a single cluster in two of the three samples. Controller 2, on the other hand, performed worse than in simulation, and Controller 3 displayed a similar level of performance.

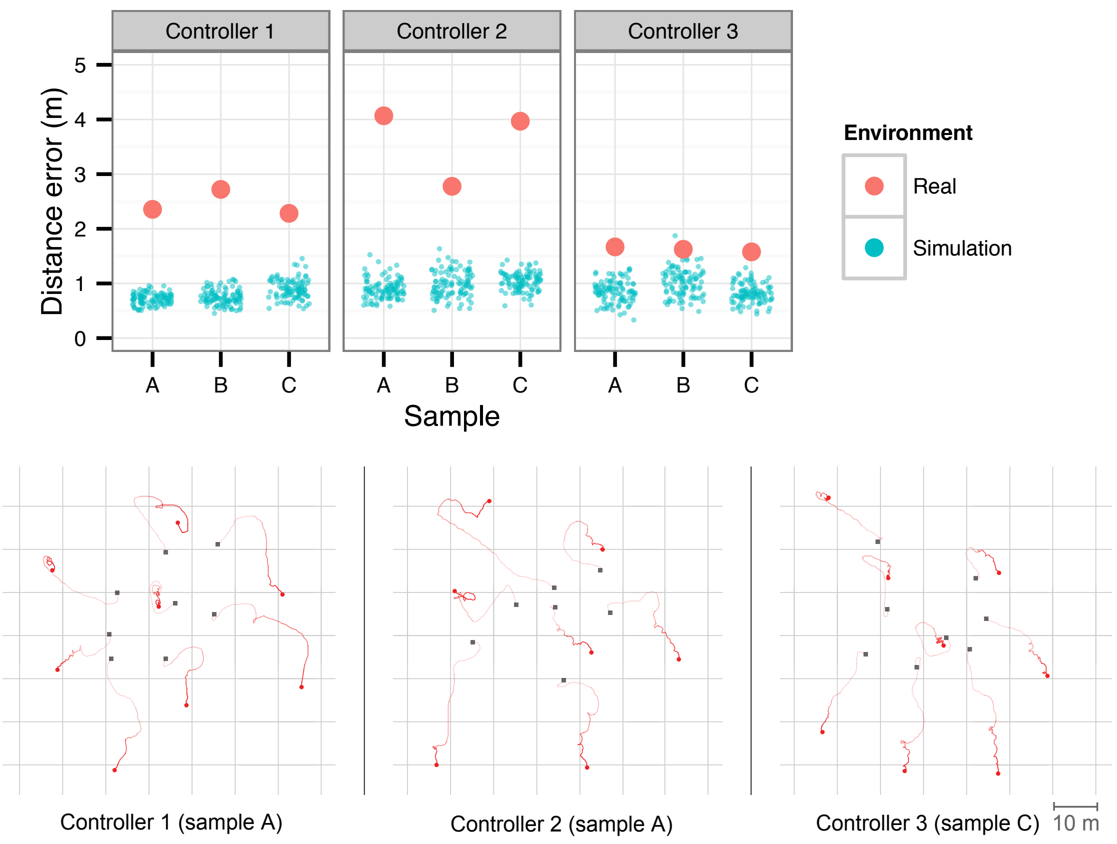


FIGURE 5.7: Real-world dispersion experiments with eight robots, one for each controller tested, over a period of 90 seconds. Top: average error to target distance (20 m) of the nearest robot in the last 10 s of each dispersion experiment. Bottom: trajectory traces of the real robots. The black squares mark the starting positions, and the red circles mark the final positions.

5.2.4 Area Monitoring

We assessed each of the three highest-performing area monitoring controllers in real robots in three areas: square, L-shaped, and rectangular. Each of the areas covered a total of 10,000 m^2. It should be noted that the evolutionary process did not optimize the controllers for areas with these specific shapes, but rather for randomly generated shapes in order to promote the evolution of general behaviors. Only the highest-performing controller was transferred, since we did not observe any significant behavioral or performance differences in the other best controllers. Each experiment lasted for a total of five minutes.

The controllers performed significantly better in reality than in simulation (see Figure 5.9, top). These results are explained by speed differences between

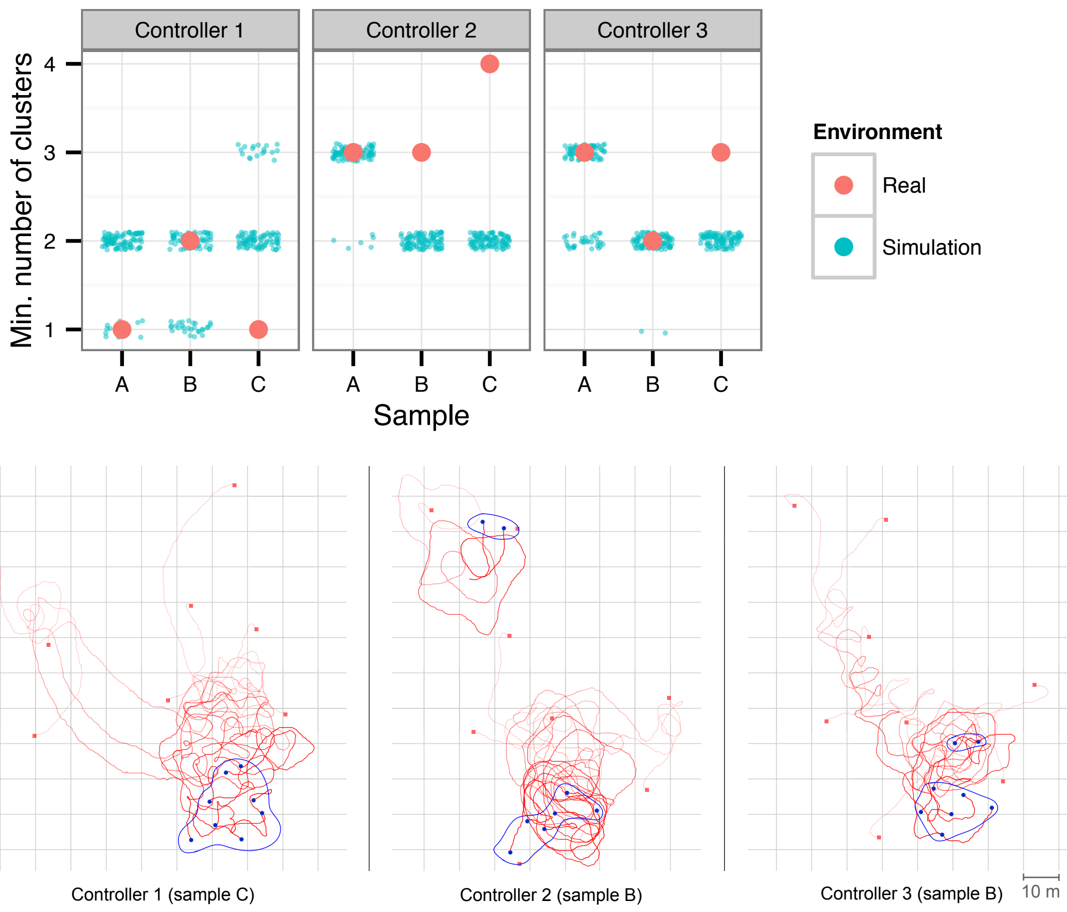


FIGURE 5.8: Real-world clustering experiments with eight robots, over a period of 180 seconds. Top: minimum number of clusters obtained in each sample. Bottom: trajectory traces of the real robots. The final clusters are highlighted in blue.

the simulated robots and some of the real robots. The speed differences directly influence the coverage measure, since a faster robot will be able to cover a larger area in the same amount of time. In terms of behavior (Figure 5.9, bottom), it is possible to see that the robots were able to effectively cover the square and rectangular areas, staying within the boundaries and moving away from the other nearby robots. The L-shaped area was also covered reasonably well, with the robots passing over all regions, but the intensity of the coverage was not uniform across the whole area. Although some robots would eventually move outside of the pre-defined boundaries (both in simulation and on the real robots) either because of wind/currents, GPS inaccuracies, or too many robots nearby, they would quickly return to the area inside the boundaries and resume monitoring.

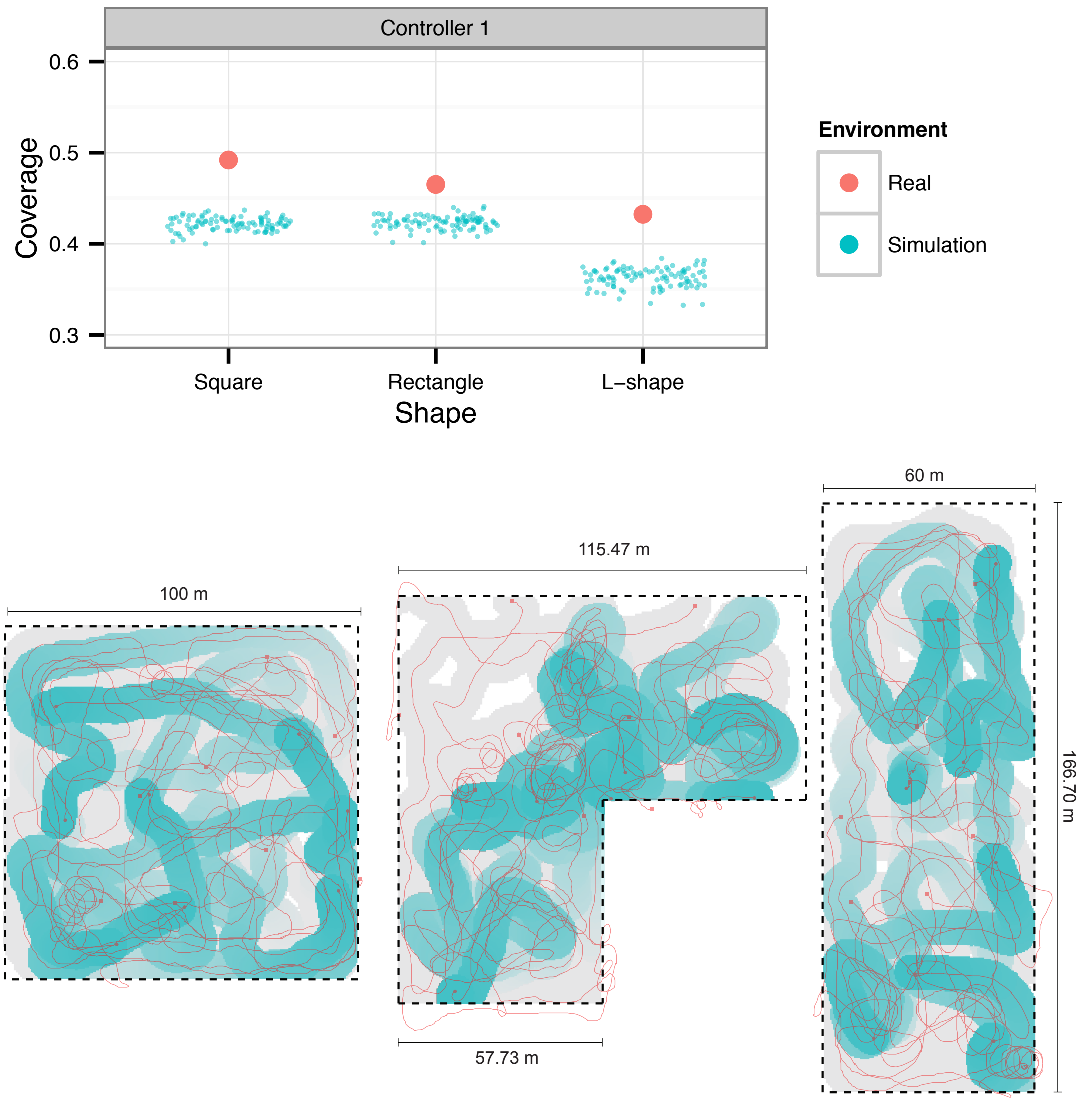


FIGURE 5.9: Real-world monitoring experiments with eight robots for Controller 1, over a period of five minutes. Top: coverage of the three different monitoring areas. Bottom: coverage maps in the experiments with the real swarm. The coverage of the area is presented in blue, and has a decay of 100 s. Trajectories for the full duration of the task are presented in red, and all the areas visited by the robots are filled in gray.

5.3 Scalability and Robustness in Real Robotic Hardware

Desirable characteristics of swarm behaviors include scalability and robustness (Şahin, 2005). The swarm should be able to perform well independently of the number of robots (until issues such as congestion and overcrowding become a limiting factor), and failures in individual units should not compromise the performance of the rest of the swarm. To evaluate if our swarm of real robots with evolved control would display these properties, we ran an additional series of experiments: (i) scalability experiments for the dispersion and clustering tasks with four and six robots, (ii) robustness experiments for the dispersion task where a second group of robots is added to the swarm after a period of time, and (iii) robustness experiments for the monitoring task where robots are physically removed during task execution (simulating faults), and then reintroduced after a period of time. Robustness experiments introduce changes in the number of robots during task execution, while scalability experiments maintain the number of robots fixed. All experiments were conducted in the real environment with the controllers that showed the highest performance in real hardware in the previous experiments.

5.3.1 Scalability

The dispersion and clustering controllers were tested with four and six robots (in addition to the first experiments with eight robots) to determine if the evolved controllers were able to perform well independently of the number of robots in the swarm. Figure 5.10 summarizes the results for these experiments.

In the dispersion experiments, while the controllers were able to perform the task successfully with swarms of different sizes, the results show a slight performance decrease in the experiments with six robots. It should be noted, however,

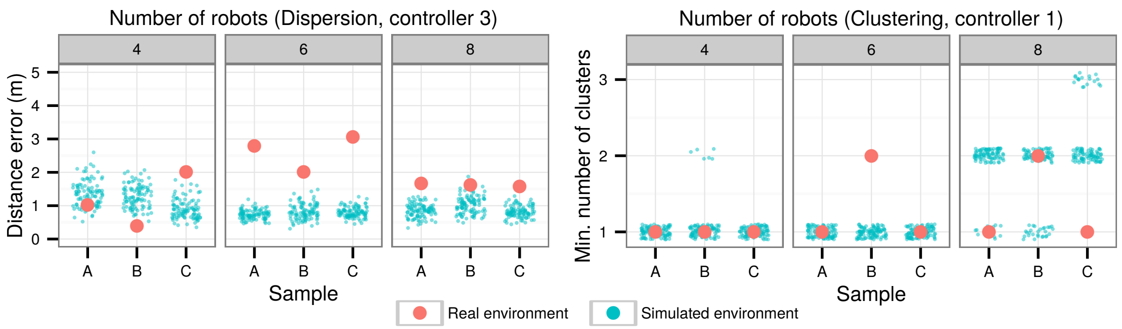


FIGURE 5.10: Scalability experiments with dispersion (Controller 3, left) and clustering (Controller 1, right) controllers. In each task, the same controller was used in a swarm of four, six, and eight robots, with three samples for each setup.

that the error obtained is still relatively low, when compared to the target distance of 20 m.

In the clustering experiments, the swarm was able to aggregate into a single cluster in all three samples with four robots, and in two out of three samples with six and eight robots. As more robots are used, the probability that more than one cluster will be formed tends to increase since there are more possibilities of choice as to where a particular robot might choose to go. Since the communication range is limited and a robot can only sense the closest neighbor in each emulated sensor slice, groups of robots might be unable to see each other, resulting in two or more isolated clusters.

5.3.2 Robustness

We setup a dispersion task that starts with a group of four robots (G_a). In the beginning of the experiment, G_a start dispersing. After 60 s, four additional robots (G_b) start moving towards the center of the swarm. When all robots of G_b are at the center ($t = 130\,\mathrm{s}$), their presence disturbs the dispersion of G_a, forcing G_a to separate even further. After the robots in G_b are in the center for some time, they start the dispersion behavior along with the robots in G_a (at $t = 180\,\mathrm{s}$).

Figure 5.11 shows the results for the adaptability experiments. When G_b moves to the center of G_a, the robots in G_a are forced to move away from one another since they try to stay at a fixed distance from the closest robot. When G_b starts dispersing, robots from G_a move further away to accommodate the movement pattern of G_b. When the whole swarm is dispersing, the distance error decreases with time as the robots continue to adjust their positions. This experiment shows that the controllers are able to adapt to conditions that they were not exposed to during evolution, particularly the addition of new robots during task execution, and still carry out the task successfully.

In a different scenario, we physically removed and added units during the execution of the monitoring task. The area to monitor was a square with $100 \times 100\,\mathrm{m}$.

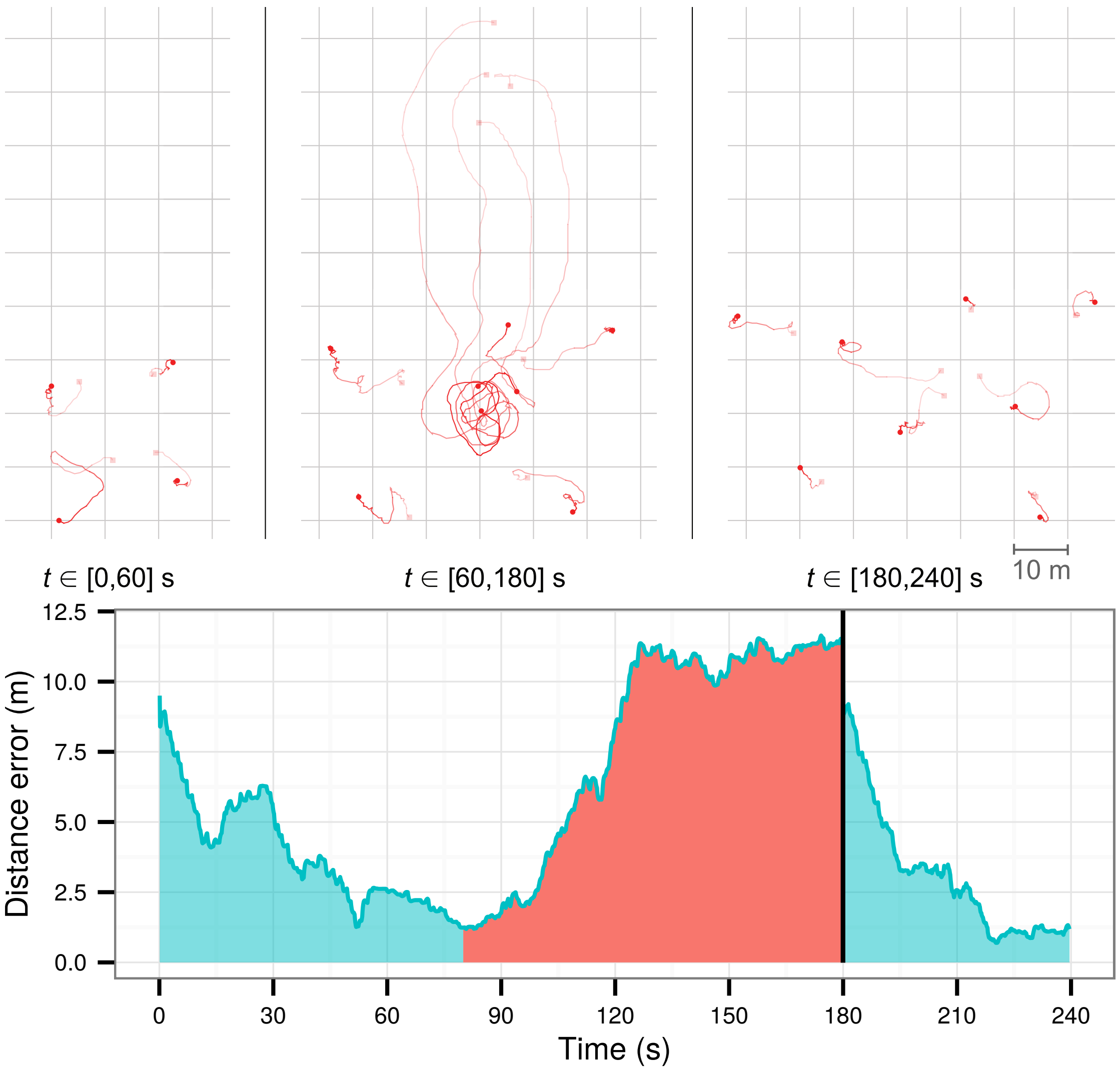


FIGURE 5.11: Robustness experiments with Controller 3 of the dispersion behavior. The red area represents the period where the robots of G_b are disturbing the dispersion of G_a, and the black vertical line indicates the point where the robots in G_b start dispersing, and where the distance error starts being measured for all eight robots.

The monitoring controller was executed for a total of 15 minutes, starting with eight robots. At the 5-minute mark, we removed four of the robots, and at the 10-minute mark, two robots were added. Figure 5.12 shows the coverage of the monitoring area at any given time during the experiment. The performance decrease after the 5-minute (300 s) mark, and the consequent increase at the 10 minute (600 s) mark show that the performance of the evolved controllers tends to scale linearly with the number of robots executing the task (see black line in Fig 5.12). These results indicate that the evolved controller is robust to variations in the composition of the swarm during task execution.

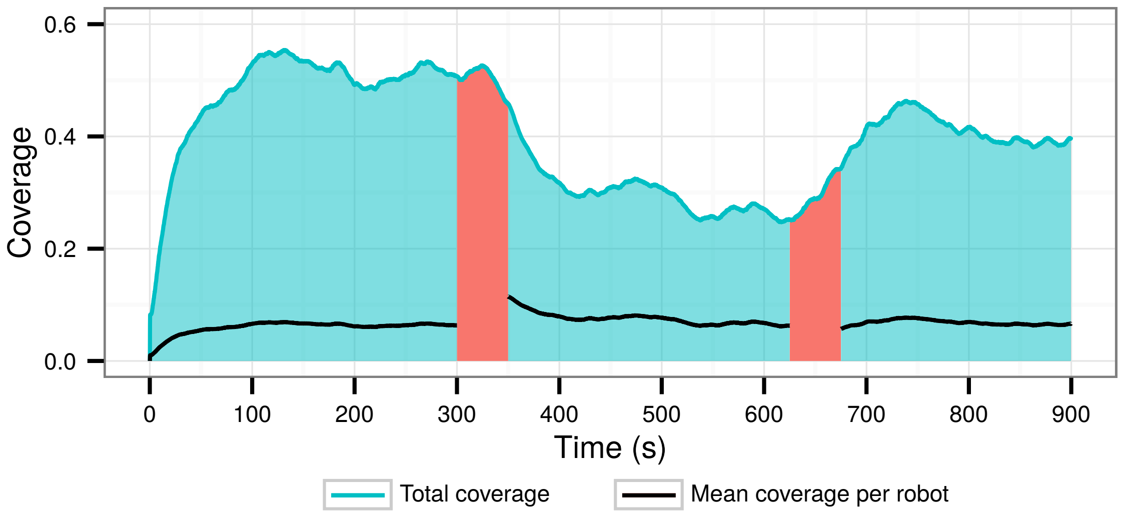


FIGURE 5.12: Robustness experiments with Controller 1 of the monitoring behavior. The time regions highlighted in red correspond to the periods when robots where either entering or leaving the monitoring area.

5.4 Sequential Environmental Monitoring Task

In this section, we conduct the first experiment where the hierarchical control synthesis approach is applied to the aquatic SRS. As a first step, we manually program a simple behavior arbitrator that activates the previously evolved behavior primitives in order to produce a sophisticated global behavior. The arbitrator for this task does not use the robots' sensory inputs in order to decide which behavior primitive should be active at any given time, and instead relies on a predefined time-based mechanism to select the appropriate behavior. Since all robots start the task simultaneously and the top-level behavior arbitrator relies on time to switch the active behavior primitive, synchronization among the swarm is implicitly guaranteed.

We integrated the behaviors described in the previous sections to accomplish a mission of sampling the water temperature over a given area of interest. The robots all started near a base station, outside the area of interest, and the mission was decomposed in five sequential sub-tasks: (i) collectively navigate to the center of the area of interest (homing); (ii) disperse; (iii) monitor the area of interest; (iv) aggregate, using the clustering behavior; and finally (v) return to the starting point (homing). The robots start collecting temperature data after reaching the center of the monitoring area, before sub-task (ii). The behavior arbitrator triggers the appropriate behavior primitive for the current sub-task sequentially to produce the control for complete mission, with each primitive executing for a predefined amount of time.

The complete controller was tested in the real robotic swarm which had to monitor an area with a size of 100×100 m, and the task lasted for a total of 11 minutes. The trajectories of the real robots executing the sequential controller mission and the spatial interpolation of the temperature data collected can be seen in Figure 5.13. Measurements taken by the robots' temperature sensors were

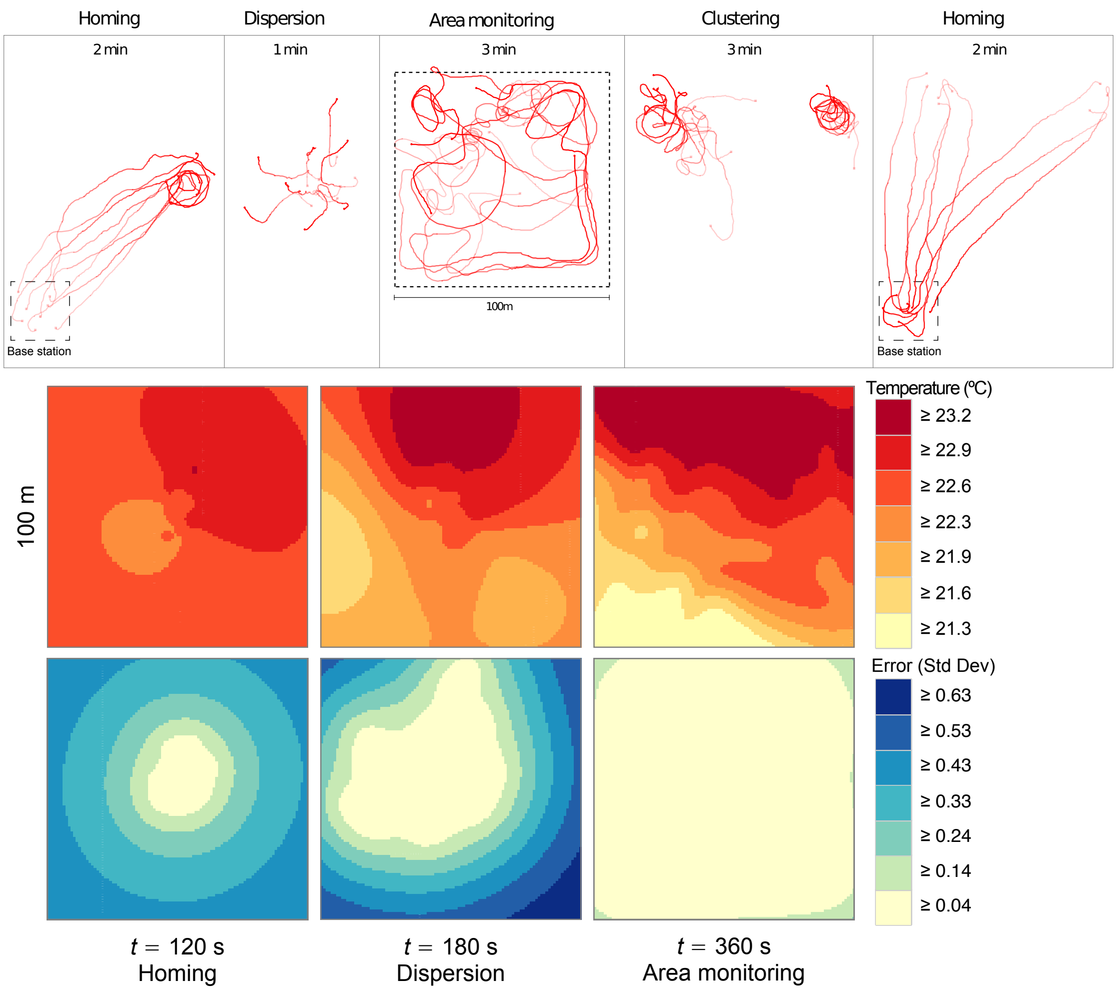


FIGURE 5.13: Results for the sequential controller mission. Top: robot trajectories for each sub-task. Middle and bottom: temperatures in the monitoring area. Data collection started after the robots arrived at the waypoint ($t = 100$ s). The middle row shows the predicted temperatures, while the bottom row shows the estimated error in the predictions.

spatially interpolated using Kriging (Stein, 2012). The uncertainty of the data deceases as the robots cover the area, resulting in a more accurate temperature profile. At the end of the monitoring period ($t = 360\,\mathrm{s}$), the estimated standard deviation of the error was < 0.14 for the whole area except in the corners where it was slightly higher.

5.5 Hierarchical Control for SRS

To test our hierarchical control synthesis approach (Duarte et al., 2015) in the aquatic environment, we chose a maritime intruder detection task where the swarm must remain within a previously designated monitoring area and pursue intruders that try to cross it (see Figure 5.14). The robots are initially located on a base station, to which they must eventually return in order to recharge their batteries.

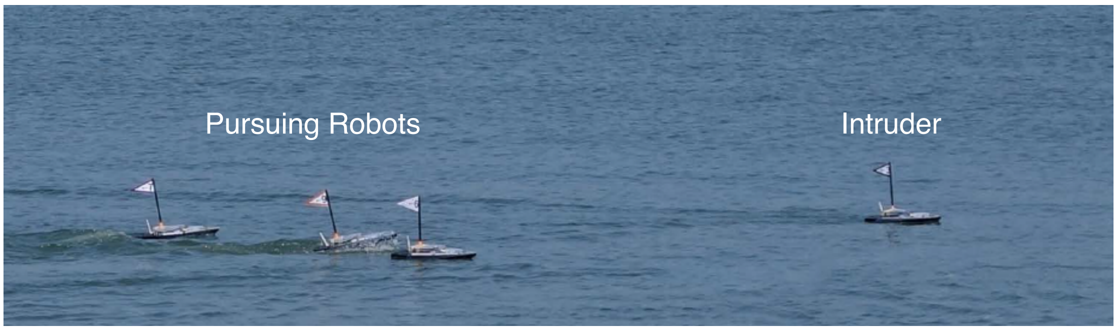


Figure 5.14: A photo of a group of robots in the area where the experiments were performed.

5.5.1 Experimental Setup

The role of the intruder is carried out by a robot running a path-following controller. The robots are aware of the intruder's position up to the range of an emulated intruder sensor. When a robot detects an intruder using its emulated sensors, it shares the intruder's position with neighboring robots. Neighboring robots, in turn, use the received positions to compute the reading for their *collective sensors* (Rodrigues et al., 2015a). Collective sensors are based on the integration of mutually shared information obtained from onboard sensors between neighboring robots. The shared information is then used to compute readings for collective sensors, which can give the individual robot information that would not be available through its onboard sensors, effectively extending their sensing capabilities. The readings for the collective sensors are calculated by taking into account the robot's own location and orientation, as well as the sending robot's location and orientation. For the robotic controller, the collective sensor is indistinguishable from regular physical or emulated sensors, since the communication and integration of received positions is part of the low-level firmware and not of the controller itself. We used a range of 40 m (equivalent to the communication range of the robots) for the emulated robot sensors, the collective intruder sensors, and the emulated geo-fence sensors, and a range of 20 m for the emulated intruder sensors.

To evolve the pursue intruder behavior primitive, we used an emulated intruder sensor with a range of 20 m, and a collective intruder sensor with a range of 40 m. During evolution, robots were randomly spread in a given area, and one intruder traversed the area with a randomly generated trajectory. When a robot detected an intruder, the robot should attempt to remain within range of it. In simulation, we rewarded controllers according to the following equation:

$$f_{intruder} = \left(\frac{1}{T} \sum_{t=1}^{T} \frac{1}{R} \sum_{r=1}^{R} found(r_t) \right) \times S \ , \tag{5.6}$$

$$found(r_t) = \begin{cases} 1 & , \ within\ range \\ 0 & , \ else \end{cases} \tag{5.7}$$

where T is the maximum number of time steps, R is the number of robots, and $found(r_t)$ indicates whether or not robot r is detecting the intruder at timestep t. The fitness trajectory for the pursue intruder controller can be seen in Figure 5.15.

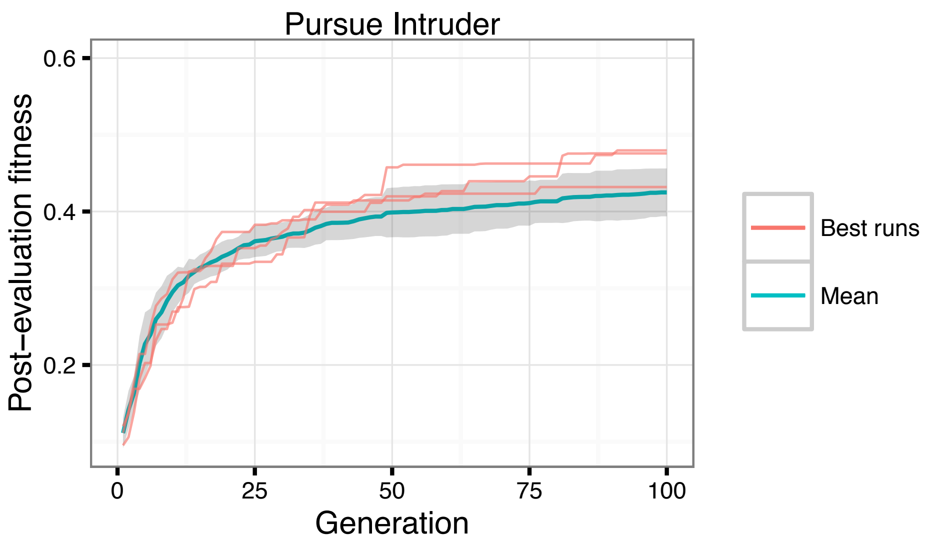


Figure 5.15: Fitness plot for the pursue intruder task. The plot shows the highest fitness cores found so far at each generation. The red lines depict the three highest-scoring evolutionary runs, while the blue line depicts the average of the ten runs, with the respective standard deviation shown in gray.

5.5.2 Top-Level Behavior Arbitrator

In this section, we detail the task setup for the complete intruder detection task and conduct a series of experiments to: (i) study how changes to the high-level behavior arbitrator impact the overall swarm behavior, (ii) study how the the swarm size and environment's size affect the performance of system, and (iii) validate the the transferability of the controllers by testing them on a real robotic swarm. Finally, we test our approach in a large-scale version of the task with swarm sizes of up to 1000 robots.

5.5.2.1 Task Setup

The intruder detection task is carried out in a rectangular monitoring area. A base station, where the robots start the task, is located 20 m from the monitoring area (see Figure 5.16). In each trial, an intruder makes a total of four crossings of the area, and the robots should detect and remain within range of their onboard intruder sensors (20 m). The task terminates after the intruder's fourth crossing. The robots have a limited battery time of 8 minutes, and need to return to the

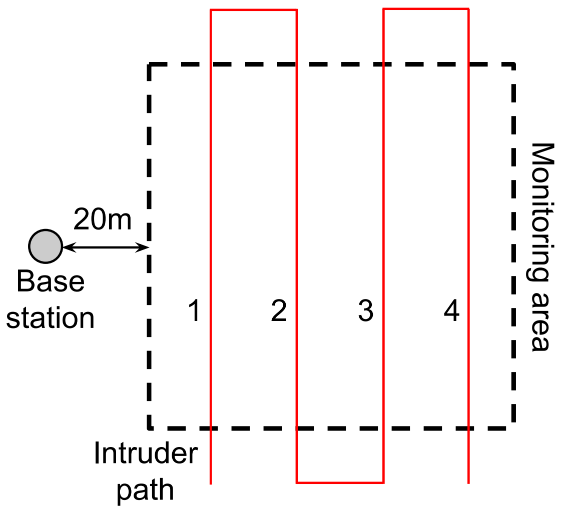


FIGURE 5.16: Representation of the experimental environment. The robots are deployed from a base station, to which they must regularly return in order to recharge their batteries. An intruder makes a total of four crosses (1-4, in red) through the monitoring area (dashed lines), and the experiment ends after the last cross.

base station in order to recharge the battery when it reaches 10%. The battery level of each robot is set between 50% and 100% in the beginning of the task.

5.5.2.2 Testing Different Arbitrators

After the we evolved the pursue intruder behavior primitive, we combined it with the two previously evolved homing (for going to the monitoring area and recharging) and area monitoring behavior primitives using a simple pre-programmed arbitrator (see Figure 5.17). The arbitrator determines which behavior should be active at any given time, depending on the current state of the robot, and the robot's sensory inputs. We designed a configurable arbitrator as a FSM, where each state corresponds to the activation of one of the arbitrator's sub-controllers. In this case, since the hierarchical controller only has a depth of two levels, the complete controller can be represented as a single FSM (see Figure 5.18).

To study how changes to the high-level behavior arbitrator impact the overall swarm behavior, we assessed the performance of four increasingly complex configurations of the FSM in simulation:

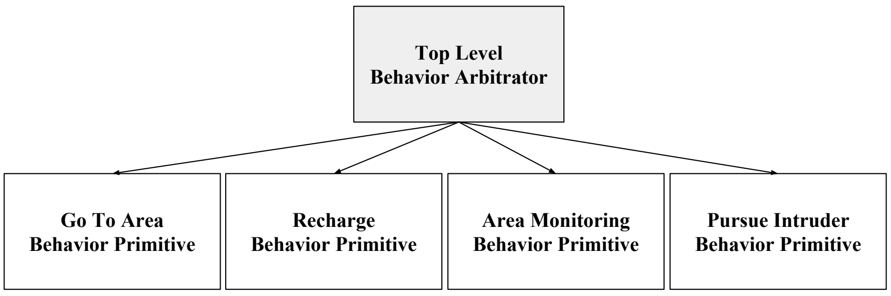


FIGURE 5.17: Conceptual representation of the hierarchical controller used in the intruder detection task.

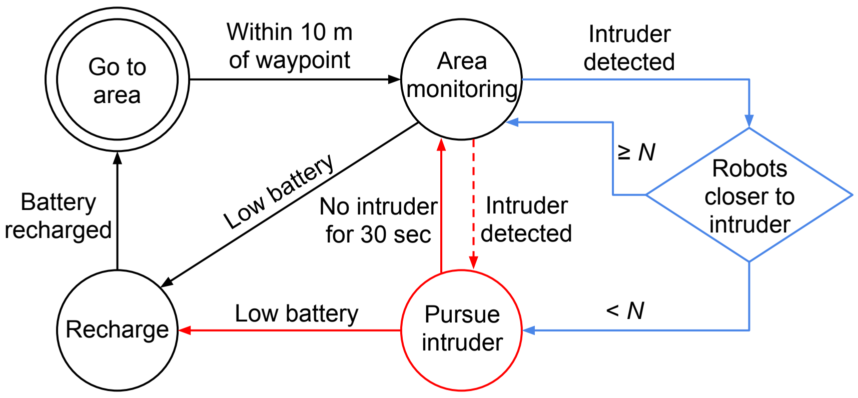


FIGURE 5.18: An FSM representing the manually programmed top-level arbitrator. The differently colored states and transitions represent incremental extensions to the arbitrator. **Monitor:** the controller will monitor the area and manage the battery level (black states and transitions). **Pursue All:** the controller will actively pursue a detected intruder (red extension). **Pursue *N*:** we limit the number of robots that can actively pursue the intruder (blue extension). In this last case, the dashed red transition from the "Pursue Intruder" state to the "Area Monitoring" state is no longer used.

Monitor: Monitor and recharge behavior with no intruder pursuit.

Pursue All: Pursue intruder with no restrictions.

Pursue 1: Pursue intruder only if there is no one else pursuing (N=1).

Pursue 3: Pursue intruder if there are less than 3 robots closer to the intruder (N=3).

We evaluated the arbitrators in six experimental setups with different number of robots and monitoring area sizes, maintaining the number of robots proportional to the area, see Table 5.1. To analyze the collective behaviors obtained with the different FSMs, we relied on three metrics: (i) *detection time*: the proportion of time that the intruder is detected when inside the area, (ii) *near robots*: the average number of robots within range of the intruder, and (iii) *coverage*: the area coverage, using a radius of 20 m for each robot and a linear decay of 100 seconds. Results are shown in Figure 5.19.

As the results in Figure 5.19 show, the *Monitor* arbitrator performs worse than the remaining arbitrators regarding the *detection time* and *near robots* metrics (Mann-Whitney U test, $p < 0.05$), since the robots do not actively pursue the intruder. The arbitrator with no restrictions in the number of robots pursuing (*Pursue All*) achieves the highest number of *near robots*. Adding restrictions to the number of pursuing robots (*Pursue 1* and *Pursue 3*) significantly decreases the number of *near robots* ($p < 0.001$, Mann-Whitney), confirming that the value of N has a significant impact in the behavior of the swarm. Regarding the *detection time*, all the arbitrators that use the pursue behavior show no statistically significant differences ($p = 0.09$, Kruskal-Wallis), indicating that as long as a single robot actively pursues an intruder, the performance of the swarm remained identical.

The *coverage* metric reveals that there is a trade-off between the number of pursuing robots and the coverage of the space: by allowing more robots to pursue the intruder, the overall coverage of the monitoring area decreases (the *coverage* differences between *Pursue 1*, *Pursue 3*, and *Pursue All* are statistically significant, $p < 0.01$, Mann-Whitney). This trade-off is explained by the higher concentration of robots near the intruder, negatively affecting the coverage of the remaining area. The lower coverage of the space did not decrease the performance of the swarm in our setups, since there was only one intruder. A higher coverage, however, is

Table 5.1: Parameters of each experimental setup.

Experimental setups	S_1	S_2	S_3	S_4	S_5	S_6
Monitoring area side-length (m)	100	141	200	245	283	316
Number of robots	5	10	20	30	40	50
Experiment duration (minutes)	8	11	15	19	22	24

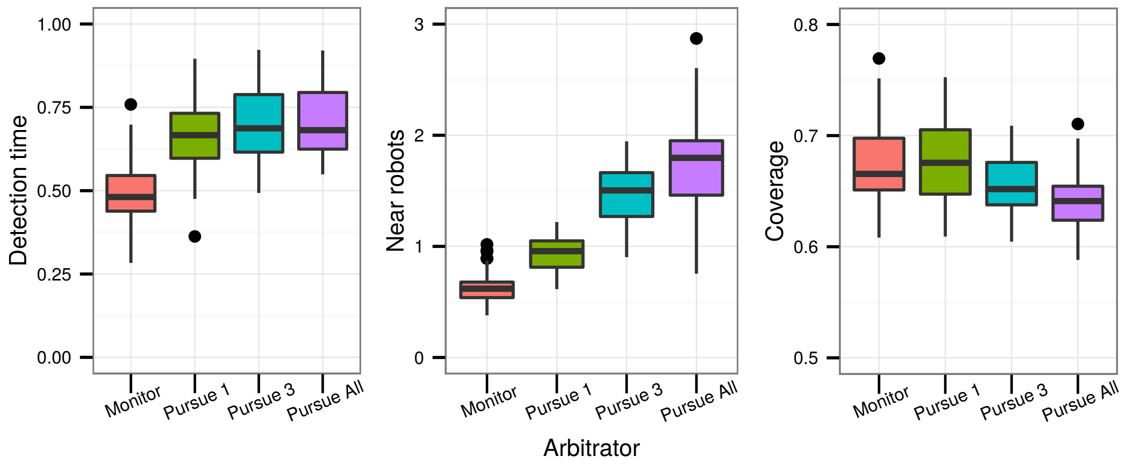


FIGURE 5.19: The plots show three different metrics (*near robots*, *detection time*, and *coverage*) for four variations of the behavior arbitrator. Each boxplot corresponds to 60 data points (10 samples per monitoring area size).

advantageous for scenarios where multiple intruders might cross the monitoring area simultaneously.

These results show how different swarm behaviors can be achieved by making simple modifications to the behavior arbitrator, reusing the same behavior primitives. For the remaining experiments in this paper, we chose the *Pursue 3* behavior arbitrator for its higher behavioral complexity, and because of its potential fault-tolerant characteristics in the case of a failure in one of the pursuing robots.

5.5.3 Arbitrator Scalability

We assessed the scalability of the *Pursue 3* behavior arbitrator using different combinations of swarm size and monitoring area size. We extracted the *detection time* metric for a total of 36 setups, each tested in 100 simulations. Figure 5.20 shows the results of these experiments. The results show that the chosen controller was scalable both across the swarm size, as well as the monitoring area size. There is always a performance increase as the swarm size increases, or as the monitoring area size decreases. In Section 5.3.2, we have shown that the area monitoring and the homing behaviors were scalable with respect to the swarm size. These results suggest that the composition of different scalable swarm behaviors resulted in a scalable composed behavior.

5.5.4 Transferring Control to Real Robots

The the real robot experiments, we used experimental setup S_1, a square-shaped monitoring area with a size of 100 m by 100 m (see Figure 5.16) and a swarm composed of five robots. We conducted three repetitions of the task with the *Pursue 3* controller. In order to compare the performance with the same controller in simulation, we ran 100 simulation samples using the same experimental setup. The results are shown in Figure 5.21.

In the experiments with the real robots, the swarm was able to find the intruder in all crossings, as shown in Figure 5.21 (top). The plot shows that up to four robots followed the intruder simultaneously, and once an intruder was detected, it was always followed continuously until it left the monitoring area. In five of the 12 crossings, the intruder was seen almost as soon as it entered the monitoring area, and then followed during the entirety of its crossing (crossings A-3, B-1, B-3, C-1 and C-2). Figure 5.22 shows an example of the swarm's behavior in the real robot experiments. The performance of the controller in the real environment was very similar to the performance in simulation, see Figure 5.21 (bottom, *Detection time*):

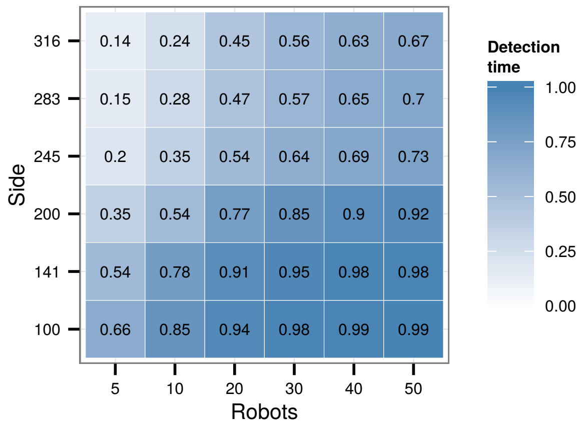


FIGURE 5.20: Proportion of time the intruder was detected when inside the arena, in different task setups with varying number of robots and arena size. Each setup was repeated in 100 simulations.

on average, the intruder was followed during 68% of the time in the real scenario, and 66% in simulation. The average number of pursuing robots is also close to what was observed in simulation. The coverage was slightly higher in reality than in simulation, which is consistent with the results reported in Section 5.2 for the evolved monitoring controller. Overall, these results show that the hybrid controller successfully crossed the reality gap, achieving a similar behavior and a similar performance on the real robots as in simulation.

5.5.5 Scaling to Large-scale Swarm Robotic Systems

We conducted scalability tests with the final controller in a large-scale intruder detection task using a monitoring area with a size of 20 km by 0.5 km. Such a patrol zone would allow the coverage of the south coast of the Italian island of

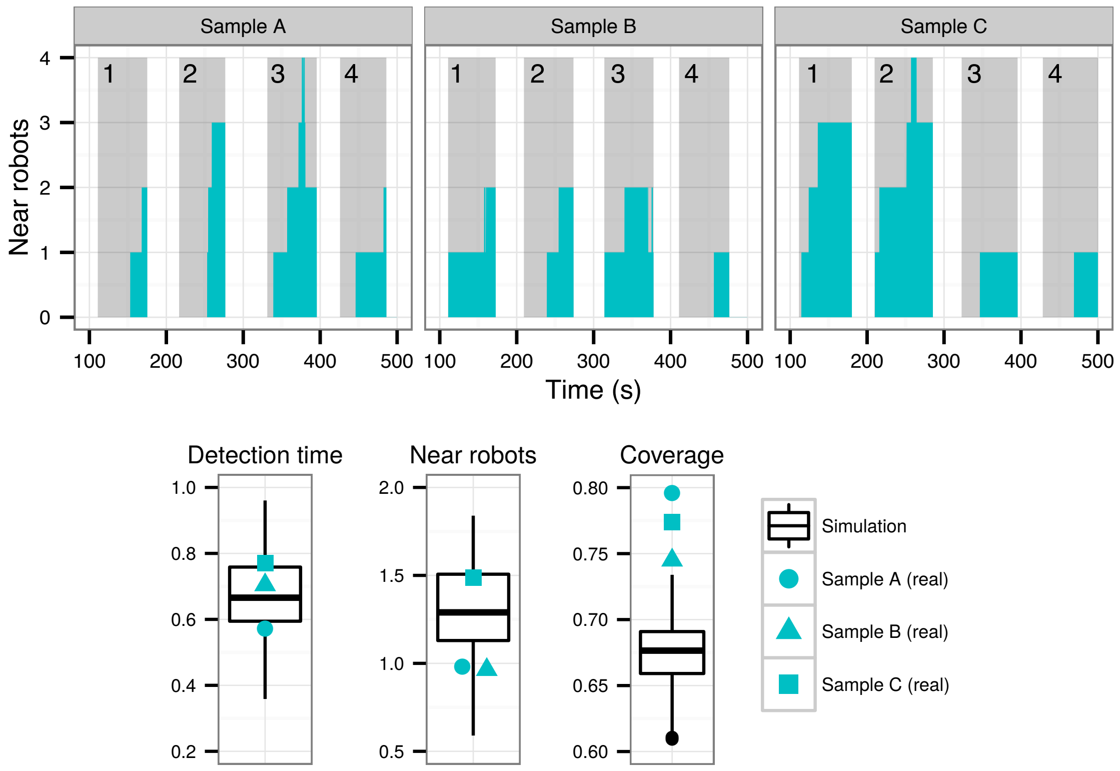


FIGURE 5.21: Top: number of robots pursuing the intruder over time for the three real-robot experiments (crosses 1-4 for each experiment). The period in which the intruder is traversing the monitoring area is shown in gray, and the number of robots pursuing at any given instant is shown in blue. Bottom: comparison of the results in simulation and in the real environment, using the metrics presented before.

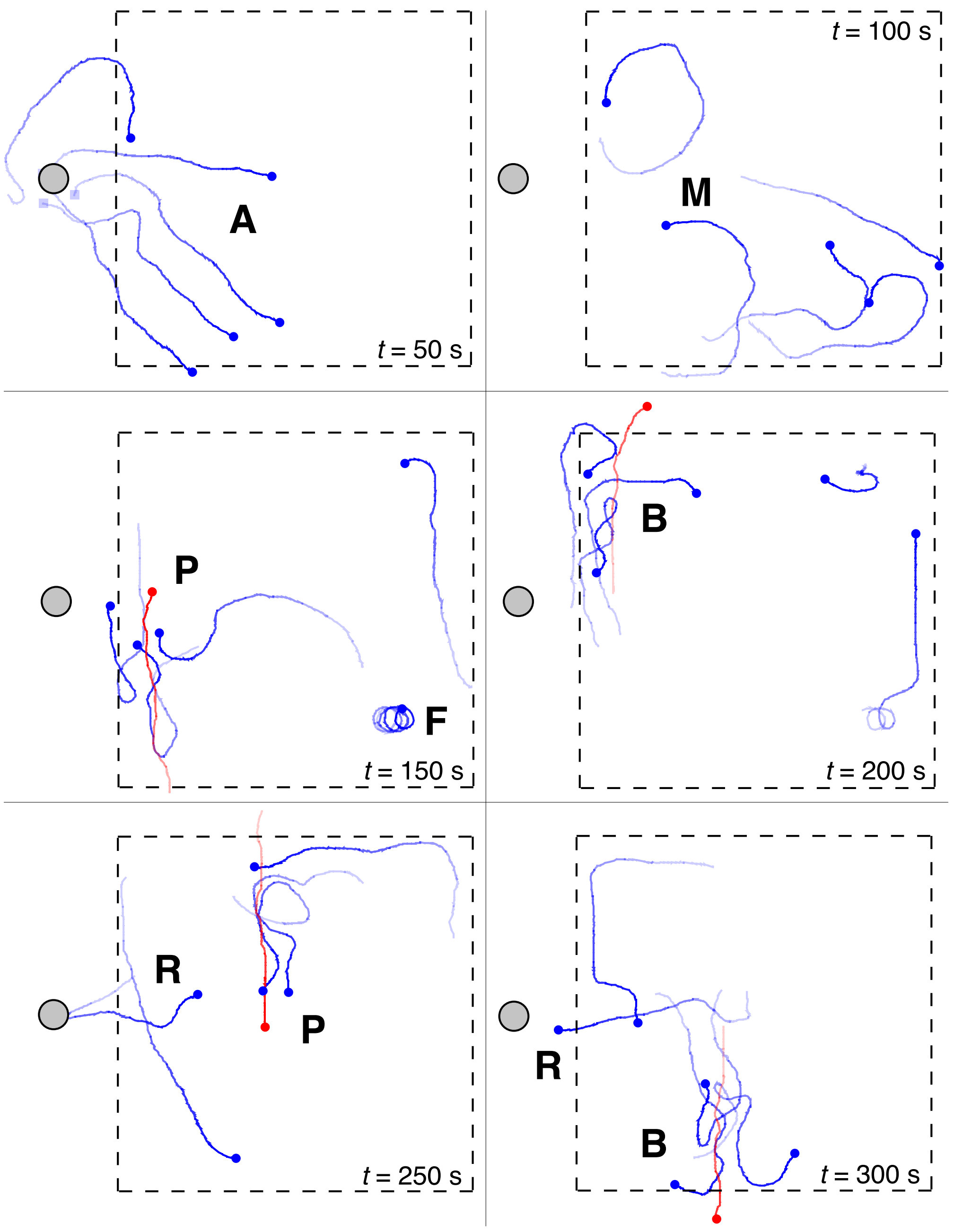


FIGURE 5.22: Traces of the first 300 seconds from sample C of the real-robot experiments, taken at 50 second intervals. The swarm is shown in blue, and the intruder is shown in red. The labels identify specific events that highlight the behavioral capabilities of the swarm. **A:** swarm going to the monitoring area. **M:** swarm monitoring the area. **P:** robots pursuing an intruder. **F:** temporary motor failure in one of the robots. **B:** robots abandoning the pursuit when the intruder leaves the monitoring area. **R:** robot going to the base station to recharge.

Lampedusa (see Figure 5.24), a major hub for illegal migration from Tunisia and Libya to Italy (Coppens, 2013). In the beginning of the experiment, a robot is deployed from each of the two base stations to a random position inside the monitoring area every ten seconds. Each base station deploys robots to one half of the monitoring area. One hour into the experiments, intruders begin crossing the monitoring area at random locations every 30 minutes (a total of 46 crossings). The simulation is run for a total of 24 simulated hours.

In order to conduct the scalability experiments, we improved some characteristics of the robots in simulation, which would be advantageous in such a large-scale mission. We changed the robots' battery life to five hours, the communication range increased from 40 m to 100 m and the sensor ranges were increased to 100 m for the robot, geo-fence, and collective intruder sensors, and to 50 m for the intruder sensors. We ran experiments in ten different scenarios with a number of robots varying from 100 to 1,000 at increments of 100 robots.

The performance observed for the different swarm sizes can be seen in Figure 5.24. By increasing the number of robots, the number of detected intruders increases until it reaches a maximum of 100% detection rate with 900 robots, which corresponds to a density of 90 robots/km^2. Another indicator of performance is the amount of time that the intruders are being pursued by either (i) one robot, or (ii) two or more robots. We can observe an improvement in the proportion of time

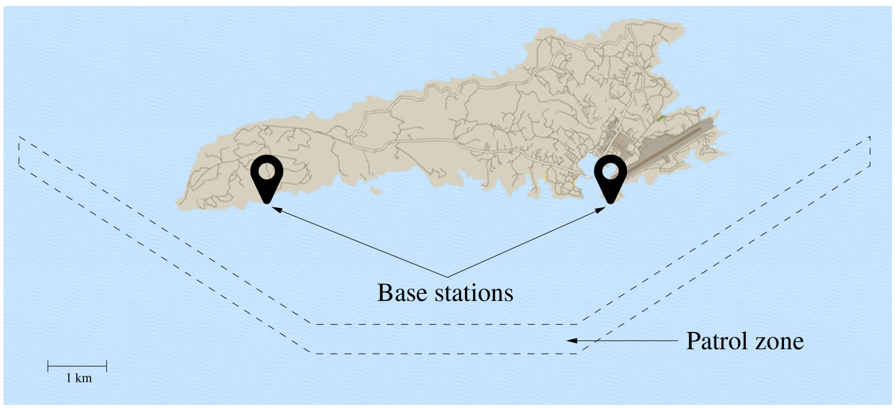


FIGURE 5.23: Map of the island of Lampedusa in the Mediterranean Sea, with a 20 km by 0.5 km monitoring area. Robots are deployed from two base stations to a random location inside the monitoring area.

that two or more robots pursue an intruder throughout the range of experimental setups.

In the scenario with 1,000 robots, the hybrid controllers successfully completed the task by detecting all 46 intruders. In total, intruders were present in the monitoring area for 279 minutes, of which they were detected by a single robot for 32 minutes, and by two or more robots for 213 minutes (11% and 76% of the total time, respectively). The system reached an equilibrium in which a mean of 60-80% of the robots were patrolling, 10-20% were returning to the base or recharging, 10-20% were going from the base to the monitoring area, and 1% were pursuing an intruder (see Figure 5.25).

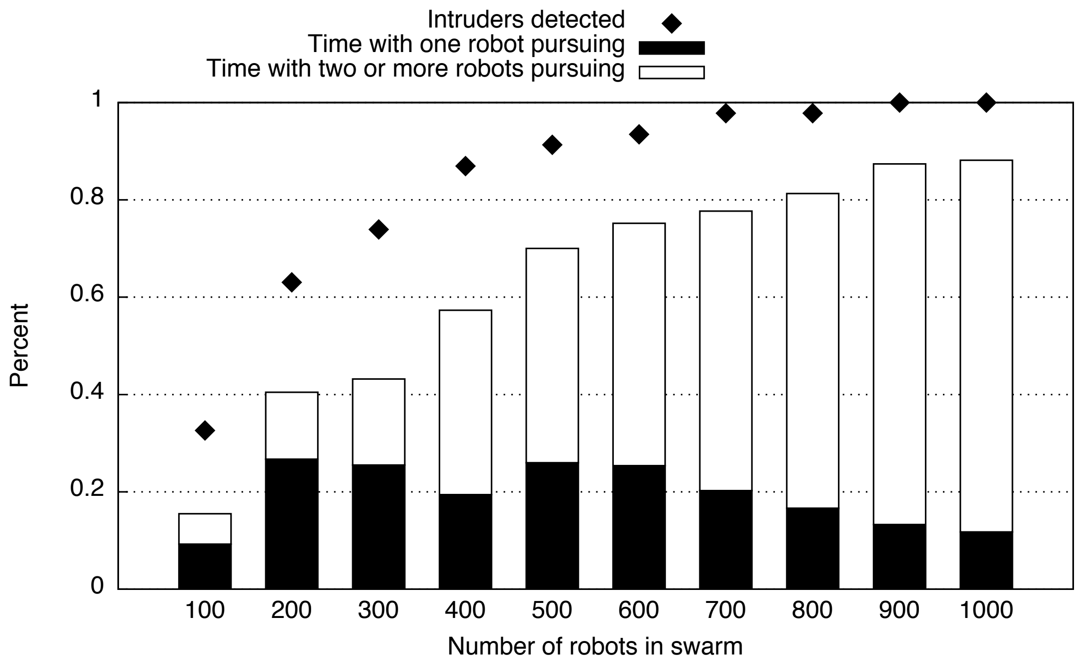


FIGURE 5.24: The figure shows both the percentage of detected intruders as points, and the percentage of time that the intruders were pursued by (i) one robot, and (ii) two or more robots, as histograms for the different simulated scenarios.

5.6 Discussion

In this chapter, we explored how hybrid hierarchical control can be applied to SRS. Hybrid controllers combine multiple behavioral blocks generated with different control synthesis approaches, thus leveraging the benefits of each and enabling complex tasks to be solved. We applied the approach to two tasks: (i) an environmental monitoring task where the swarm had to collect water temperature data from a predefined area, and (ii) an intruder detection task with realistic constrains, where the swarm had monitor an area, follow any intruders that crossed it, and regularly return to a base station in order to recharge their batteries.

For the intruder detection task, we reused two evolved behavior primitives (homing and area monitoring) from the experiments conducted in Section 5.2, and evolved an additional behavior primitive for intruder pursuit. These behaviors were then combined with a behavior arbitrator consisting of a manually programmed FSM. We first tested multiple variants of the behavior arbitrator, and

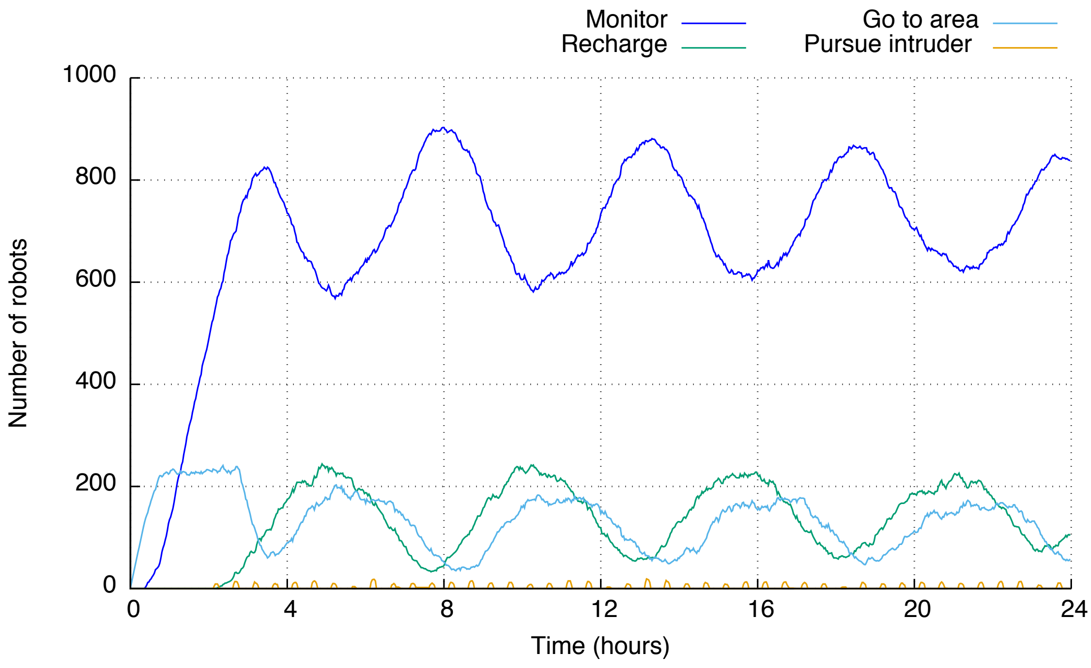


FIGURE 5.25: Plot of the states of the robots' manually programmed behavior arbitrators in the scenario with 1000 deployed robots over a period of 24 hours of simulation.

showed how simple modifications to the arbitrator resulted in different swarm behaviors, using the same behavior primitives. We then tested the scalability of the composed controller, showing that it scales predictably with the number of robots and the size of the monitoring area. We tested the controller in real robots, using a swarm of five aquatic surface robots and one intruder. The controllers successfully crossed the reality gap by achieving a similar performance on the real robots as in simulation. Finally, we tested the controllers in a large-scale simulation-based study with swarm sizes of up to 1000 robots.

The composition of different evolved sub-controllers using a FSM-based behavior arbitrator allowed us to leverage evolution's automatic synthesis of self-organized behavior, while at the same time provided high-level control of the swarm behavior through simple manually programmed rules. Our results show that hybrid control introduces a series of advantages for the synthesis of control for robotic swarms: (i) multiple evolved behaviors can be combined to produce control for complex swarm robotics tasks, (ii) previously evolved control can be reused in different applications, simplifying the synthesis of control for new tasks, (iii) the flexibility of manually programmed behavior arbitrators can enable realistic task-specific constraints to be addressed quickly, without the need to re-evolve control, and (iv) the reality gap can be effectively overcome in complex tasks by relying on general and robust behavior primitives.

A number of previous works have shown that evolutionary computation is a powerful tool for the automatic synthesis of control for SRS (Trianni and Nolfi, 2011). Our experiments confirm not only confirmed this hypothesis, but also demonstrated for the first time that evolved swarm behavior can be applied outside of controlled laboratory conditions. However, once we started considering tasks that require more complex behaviors (as the temperature monitoring task in Section 5.4 and the intruder detection task in Section 5.5), the use of standard evolutionary techniques became problematic in the sense that defining an effective fitness function for the complete version of this task was too cumbersome. We therefore applied our hierarchical control synthesis approach in order to synthesize control for these tasks. We were able to reuse previously evolved behaviors, which proved to be modular and straightforward to integrate with one another. In summary, our experiments showed how evolutionary techniques can be combined with a human expertise to design and synthesize complex behaviors that enable SRS to carry out tasks under realistic constraints.

5.6.1 Transferring Control to Real Hardware

One of the challenges of synthesizing control for robots in uncontrolled scenarios is dealing with the unpredictable environmental conditions or unexpected circumstances. In the case of our experiments, many variables have to be taken into account, such as wind, currents, waves, the movement dynamics of the robots, GPS and compass inaccuracy, and the effect of all these in the sensory readings and actuation of the robots. Some of these elements can be extremely difficult, or even impossible, to model accurately. For instance, waves can impact the communication range of the robots, while water currents or waves can have strong effects on the movement dynamics of the robot.

To make the evolutionary process computationally viable, the complexity of the simulations must remain relatively low. It is, therefore, important to rely on a simplified physics engine. We implemented a preliminary version of the dynamics by taking measurements of the robot moving with different combinations of left and right motor speeds. Afterwards, we collected GPS data of a controller executing on real hardware, and tried to match simulation to reality by adjusting the motion model. Nevertheless, this model was still a simplification of reality, as it did not explicit take into account a number of physical properties such as hull balance and waves generated, which can impact motion patterns. To minimize the impact of the model simplification on the transfer of behavior, we introduced noise into the simulation model, particularly in the sensors, actuators, and in the environment, as advocated by Miglino et al. (1996) and Jakobi (1997).

The results from our experiments showed that the evolved controllers were generally able to cross the reality gap, while maintaining the desirable characteristics of SRS, such as robustness and scalability (Şahin, 2005). In a number of experiments, a speed difference between some of the real robots and the simulated robots (up to 20%, in some cases) contributed to differences in terms of performance. In the case of the area monitoring controllers, for instance, the real swarm obtained a consistently higher performance due to its ability to more quickly cover a larger area. While the behaviors of the real robots were generally similar to those observed in simulation, some motion patterns did not transfer well. With some of the simulated dispersion controllers, for instance, the robots moved in small circles to maintain their current position. In reality, however, these circles were larger, which resulted in a lower performance. This difference highlights the limitations of the simplified physics model we adopted in the simulation environment. We are considering two different approaches to address these issues in future work: (i) further improve the simulated movement dynamics, thereby decreasing the mismatch between simulation and reality, and (ii) select for controllers that rely on motion patterns that transfer well from simulation to reality, as proposed in previous works (Koos et al., 2013; Lehman et al., 2013; Cully et al., 2015).

5.6.2 Robustness to Hardware Faults

The use of robots in real-world conditions will often imply hardware malfunctions. Prolonged used and environmental factors can lead to faults in hardware, especially when considering swarms of robots, where each robot must be relatively inexpensive and, therefore, equipped with lower-quality components. In our experiments, the main source of faults were the robots' motors, which sporadically and temporarily stopped, causing the robot to stop completely or start moving in circles. This fault would solve itself when the motor stopped receiving power and started again, which often happened naturally as a result of the normal controller operation. Other problems that we observed were the decrease of motor power with lower battery levels, and the often erroneous GPS readings.

The aforementioned faults occurred often during the execution of all experiments reported in this chapter. Even though these faults were not contemplated in simulation, and therefore were not taken into account during the evolutionary process, they did not compromise the overall performance of the swarm. The swarm generally exhibited the same group-level behavior when temporary individual faults occurred. In future work, we will study the fault-tolerance properties of the system more systematically by injecting different types of faults (Christensen et al., 2008), and by measuring the impact of these faults in the performance of the swarm.

An additional source of uncertainty, related to the hardware, was the slight heterogeneity among the robots of the swarm. Different robots often had different battery levels, and even slightly different motors, which caused individual robots to perform differently. We measured this heterogeneity in the monitoring task, where all robots supposedly have very similar behaviors. In the real swarm, the difference between the slowest and the fastest robot was up to 0.6 m/s at certain times. For the same task, in simulation, the difference between the slowest and the fastest robot was just 0.1 m/s. This suggests that the real swarm had to deal with a much higher degree of heterogeneity than what was contemplated during the evolutionary process.

5.7 Summary

In this chapter, we reported experiments with a swarm of small, simple, and inexpensive aquatic surface robots, which relied on evolved, distributed control and local communication. Using simulation, we evolved behaviors for four common collective tasks: homing, clustering, dispersion, and area monitoring. The evolved controllers were then systematically evaluated in the real robots in a large and uncontrolled aquatic environment. Our results showed that the controllers displayed similar behaviors and levels of performance in the real swarm as those observed in simulation. Finally, we successfully applied our hierarchical control synthesis approach for an environmental monitoring task and for a maritime intruder detection task with realistic task constrains. The robots consistently displayed the key desirable properties of swarm robotic systems, namely robustness and scalability, in the different experiments.

Chapter 6

Conclusions and Future Work

The application of classic evolutionary robotics techniques for the synthesis of control for complex tasks has proven problematic: the evolutionary process is often difficult to bootstrap and vulnerable to deception as task complexity increases. Furthermore, the reality gap has proven a major obstacle when transferring control from simulation to reality. These issues have, so far, effectively prevented the application of ER-based control synthesis techniques for real-world robotic systems. In this thesis, we addressed these issues by proposing and studying the hierarchical control synthesis approach.

In our approach, a task is decomposed into several sub-tasks, and robotic control is then synthesized for each sub-task. The different sub-controllers are then composed hierarchically. Actuation nodes (behavior primitives) are at the lower levels of such a controller, while decision nodes (behavior arbitrators) are higher up the hierarchy and select which behavior primitive should be active at any given time. A controller can be hybrid, which means it can be composed of control nodes synthesized with different techniques, such as evolution and manual programming.

We first validated our hierarchical control synthesis approach in a series of experiments with an e-puck robot (Mondada et al., 2009): (i) a rescue task with a purely evolved hierarchical controller, (ii) a navigation task with a hybrid hierarchical controller (evolved arbitrator and manually programmed primitives), and (iii) an integrated task that required fine sensorimotor coordination with a hybrid hierarchical controller (evolved arbitrators and evolved/manually programmed primitives). In these experiments, our approach allowed for the synthesis of ER-based control for tasks that were beyond the state of the art in the field, both in simulation and in real robotic hardware.

We then applied our approach to SRS. First, we presented a series of validation experiments based on four canonical swarm robotics tasks (homing, clustering, dispersion and area monitoring) where we compared the performance of controllers in simulation and in the real world, and validated that our robotic platform displayed key swarm robotics characteristics, such as scalability and robustness. Finally, we applied our hierarchical control methodology to an environmental monitoring task, and to a maritime intruder detection task, in which a swarm of robots must monitor a pre-defined patrol zone and detect and follow a robot that acts as an intruder. The experiments conducted in Chapter 5 are the first demonstrations of evolution-based control in a swarm of robots outside strictly controlled laboratory conditions.

The performance of evolved control is typically lower when transferred to a real robot, since the evolutionary process takes place offline. Using our hierarchical control synthesis approach, real-robot performance can be assessed incrementally as sub-controllers are evolved, which allows transferability issues to be addressed immediately and locally in the controller hierarchy. Since each sub-controller solves only part of the task, it becomes feasible to introduce different types of noise in simulation and, therefore, increase the robustness of the solutions. Furthermore, different control synthesis methods, such as evolution or manually programmed control, can be leveraged depending on the requirements of the sub-task.

In summary, our contribution is threefold: (i) we presented the hierarchical control synthesis approach, which allows different control synthesis techniques to be combined in a single controller and enables the application of evolutionary methodologies for complex tasks, (ii) we validated our approach in real robotic hardware and in tasks that are beyond the state of the art in terms of task complexity, and (iii) we demonstrated for the first time a robotic swarm with evolved control performing task outside of strictly controlled laboratory conditions, showing that SRS, ER and hierarchical control systems are viable approaches for complex real-world tasks.

6.1 Future Work

Below, we discuss potential avenues of research with respect to the application of ER methodologies to complex real-world tasks.

6.1.1 Automatic Behavior Composition

One of the limitations of the approach proposed in this thesis is the manual process of decomposing a task into sub-task, and then defining the structure of hierarchical controller and the individual experimental setups. While the different nodes of the controller can take advantage of evolution's automatic control synthesis, the complete control synthesis process is dependent on expert knowledge about the task. One way to overcome this limitation is to explore automatic behavior composition, in which the controller can be synthesized hierarchically automatically, without the intervention of the experimenter. In this scenario, the experimenter would only need to define a high-level fitness function for the complete task.

Automatic behavior composition could potentially be achieved by generating a repertoire of many different behaviors (Cully et al., 2015), using techniques such as novelty search (Lehman and Stanley, 2011). The repertoire would depend on the robotic platform and on the available sensors and actuators, but once generated, the repertoire could be used for any number of tasks. In order to automatically generate hierarchical control, an algorithm would test different combinations of behaviors. One possible approach would be to develop a variation of the NEAT (Stanley and Miikkulainen, 2002) evolutionary algorithm, where complete behavioral nodes would be added, instead of simple neurons. Another approach could be to adapt the technique proposed by Silva et al. (2014a), where a neural controller can have both simple neurons, and more complex macro-neurons, which are also subject to variation during the evolutionary process.

6.1.2 Synchronization and Consensus Achievement

SRS typically do not need explicit inter-robot synchronization methods in state-of-the-art tasks, since such tasks are usually simple and all robots are executing identical behaviors. Once we consider hierarchical control synthesis as an enabler for complex tasks, different robots may be executing completely different sub-controllers simultaneously. In the case of the intruder detection task presented in Chapter 5, for instance, each robot could be executing different behaviors (such as recharging, following an intruder or monitoring) independently. Some tasks might, however, require the swarm as whole, or a sub-set of the swarm, to collectively decide to switch sub-tasks. In such situations, the robots would need to achieve a consensus on which sub-task to perform, and when to start performing it. An example would be the environmental monitoring mission from Section 5.4, where the robots were implicitly synchronized due to the behavior arbitrators' time-based behavior switching mechanism.

Synchronization in the context of SRS has been studied in the past. The use of pulse-coupled oscillators, a synchronization mechanism found in fireflies (Smith, 1935) and other natural systems (Winfree, 2001), has been used to detect faults (Christensen et al., 2009), perform task allocation (Castillo-Cagigal et al., 2014), and evolve synchronized behaviors (Trianni and Nolfi, 2009). This type of synchronization mechanism could potentially be adapted to our approach by means of parallel execution with the robots' hierarchical controllers. Different pulse-coupled oscillators could be bound to different sub-controllers, and each robot might promote the execution of a particular sub-controller by activating the sub-controller's oscillator. If the robots observed consensus achievement, that is, the synchronization of a single oscillator across its neighbors, they would change to the appropriate behavior.

6.1.3 Towards Real-world Applications

The experiments reported in Chapter 5 focused mostly on the spatial organization of the robots of the swarm. The robots only used their position and the position of the neighboring robots for the behavioral decision process. The integration of data from onboard sensors such as cameras, radars, LIDARs or sonars, may be an essential input for the controllers in other tasks. The challenge of integrating these sensors in the control process is twofold: (i) the data from many of these sensors is not trivial to decompose into meaningful information that can be fed to neural controllers, and (ii) these sensors need to be accurately modeled in simulation in order to evolve the controllers.

Another significant challenge in robotics systems operating in real conditions is endowing controllers with the capability to cope with unforeseen circumstances, including both faults within the swarm, as well as the interference of external entities and environmental conditions. Our experiments confirmed that swarm behaviors inherently have some degree of tolerance for individual faults. In real-world applications, however, it is necessary to guarantee that faulty individuals will not compromise the whole swarm. A number of approaches have been proposed for fault detection and fault tolerance in swarm robotic systems (Christensen et al., 2008; Tarapore et al., 2015), and it has also been shown how evolutionary computation can be used to foster fault tolerance at the individual level through online learning and adaptation (Cully et al., 2015). Regarding events caused by external factors, such as the approximation of other vessels, the robots should have the capacity to deal with such events in a way that guarantees the safety of all entities and conforms with the existing regulations. Combining such fail-safe behaviors with the mission controllers may require the combination of evolved and manually programmed behaviors, such as the one shown in Section 5.5.

Appendices

Appendix A

Other Contributions

In this Appendix, we provide an overview of other contributions in the course of the doctoral research presented in this thesis, namely the participation in the CORATAM and HANCAD projects, the participation in the COHiTEC technology transfer program, media coverage, and developed software tools.

A.1 The CORATAM and HANCAD projects

Maritime tasks, such as surveillance and patrolling, aquaculture inspection, and environmental monitoring, typically require large operational crews and expensive equipment. Only recently have unmanned vehicles started to be used for such missions. These vehicles, however, tend to be expensive and have limited coverage, which prevents large-scale deployment. Swarms of small, inexpensive aquatic robots have the potential to take on such maritime tasks in an autonomous way. Robotic swarms with decentralized control based on principles of self-organization display several characteristics which are ideal for maritime tasks, such as distributed sensing, scalability, and robustness to faults (Bonabeau et al., 1999). In the CORATAM (Control of Aquatic Drones for Maritime Tasks) and HANCAD (Heterogeneous Ad-hoc Network for the Coordination of Aquatic Drones) projects, we explored communication mechanisms and the automatic synthesis of control systems such swarms of aquatic surface robots. In order to conduct our experiments, we designed and built a swarm robotics platform (see Appendix B). We built 10 units of the final prototype and used those for extensive experiments conducted at Parque das Nações, Lisbon, and at the REX–Robotics Exercise held at the Lisbon Naval Base, Alfeite. The experiments carried out in the scope of the CORATAM and HANCAD projects are detailed in Chapter 5. A video of our projects was produced and submitted to the prestigious AAAI video competition, where it received the "Best Robot Video" award:

- A. L. Christensen, M. Duarte, V. Costa, T. Rodrigues, J. Gomes, F. Silva, S. M. Oliveira, "**A Sea of Robots**", *30th Conference on Artificial Intelligence (AAAI-16)*, 2016, Phoenix, Arizona, USA. **Winner of the "Best Robot Video Award"**

A.2 COHiTEC

During the CORATAM and HANCAD projects, six of the BioMachines Lab members participated in the COHiTEC program as team OceanSwarm. The purpose of the program is to help academic researchers move their technology from the laboratory to the industry. The approach presented in this thesis was a key component of the technology that we used as a basis for our participation in the program. After four intensive months of market research, business models, financials, intellectual property, SWOT analysis and many other business tools and methods, we developed and pitched our business proposal in Pavilhão do Conhecimento, Lisbon, to potential investors, partners, and the media. We are currently continuing the process that we started in COHiTEC in order bring our technology to the market.

A.3 Media Coverage

The CORATAM and HANCAD projects, as well as our participation in CO-HiTEC, recently helped us gain media exposure in several outlets:

- Successo.pt, Sic Notícias, national TV segment, "Sucesso.pt da Cohitec", 21-09-2015, available at `http://goo.gl/shqD8r` (0:10 to 15:15)
- Exame Informática TV, national TV segment, Sic Notícias, "Exame Informática n.º 467", 20-09-2015, available at `http://goo.gl/mODSYi` (0:20 to 2:40)
- Telejornal Açores, RTP Açores, regional TV segment, "Mini Fórum CYTED", 16-06-2015, available at `http://goo.gl/qmJ1p7` (13:00 to 16:00)
- Telejornal Açores, RTP Açores, regional TV segment, "Utilização de drones", 20-06-2015, available at `http://goo.gl/RhtdM9` (3:30 to 5:00)
- Exame Informática, national print article, "Heróis do Mar e do Enxame", 05-09-2015, available at `http://goo.gl/chQXAy`
- Exame Informática, online article, "Cohitec: detetar um enfarte numa hora e o HIV em três dias", 15-07-2015, available at `http://goo.gl/Cg3c48`
- Observador, online article, "Inovações que detetam o HIV em três dias e drones que procuram peixe sozinhos. Em português", 14-07-2015, available at `http://goo.gl/E95prh`
- Açoreano Oriental, regional print article, "Investigadores desenvolvem drone para realizar tarefas no mar", 22-06-2015, available at `http://goo.gl/RCAL5K`
- Fórum Estudante, national print article, "Drones no ISCTE-IUL", 06-2015 , available at `http://goo.gl/296t9p`

A.4 Software Tools

To carry out the experiments reported in this thesis, several software tools had to be developed or extended. In this section, we provide an overview of the contributions in terms of the main software tools developed, which are available under the GNU GPL license and can be found online at `https://github.com/BioMachinesLab`.

A.4.1 JBotEvolver

JBotEvolver (Duarte et al., 2014c) is a simulation platform for research and education in evolutionary robotics. JBotEvolver is a Java-based open-source, cross-platform framework, and has been used in a number of previous ER studies of our research group, from offline evolution to online evolution and learning, and from single to multirobot systems (swarms of up to 1,000 robots have been simulated in real-time), and in a number of undergraduate and graduate courses at the University Institute of Lisbon. JBotEvolver has been used as the main simulator in over 30 publications since 2011 (see `https://github.com/BioMachinesLab/jbotevolver` for an updated list of publications).

JBotEvolver's main features are its ease of installation and use, and its versatility in terms of customization and extension. A fundamental design philosophy behind JBotEvolver is to provide a basis for ER experiments without the need for detailed framework-specific knowledge. Following this philosophy, JBotEvolver enables the configuration of experiments programmatically or via a plaintext file that specifies which features will be included in the simulation. The corresponding classes are then seamlessly loaded in execution time via Java's Reflection API. In this way, JBotEvolver can also make use of external, user-defined classes that extend the base implementation. Additionally, JBotEvolver is self-contained, but can also be used as an external library in other applications.

During the course of the work conducted for this thesis, JBotEvolver was significantly improved and extended. Major contributions include an integration with the distributed computing system Conillon (see the following section), a general-purpose GUI, lowering the barrier to entry for new users, the implementation of reflection-based dynamic class loading, and the integration of the NEAT (NeuroEvolution of Augmenting Topologies) evolutionary algorithm (Stanley and Miikkulainen, 2002). We use JBotEvolver for all simulation-based experiments conducted in this thesis.

A.4.2 Conillon

Conillon is a Java-based distributed computing system developed at the University Institute of Lisbon (Silva et al., 2011). Conillon enables arbitrary tasks to be computed in a distributed fashion, allowing for the parallelization of computationally-intensive processes. It is based based in a Client-Server-Worker model, where a centralized server distributes tasks, submitted by clients, to worker nodes. These nodes can be added to the network in an ad-hoc manner either through: (i) a standalone application, (ii) a Java applet running in a browser, or (iii) as a screensaver on PCs. Conillon's dynamic request of Java classes allows tasks with different codebases to be submitted simultaneously. As with JBotEvolver, Conillon was improved and extended during the work conducted for this thesis. Major contributions include the development of an administrative interface, performance improvements, and several bug fixes.

A.4.3 Evolution Automator

Evolution Automator is a tool that automates certain aspects of the evolutionary process, using JBotEvolver as a library. In the simplest use case, the tool allows the user to setup multiple experiments with different parameters, and execute those experiments at a controlled pace (only a few at a time). This is particularly relevant when several users are running experiments using Conillon in order to distribute the computing capacity. The user can define a single configuration file that sets up all different experiments, including the option to conduct post-evaluations after all evolutionary runs of a particular experiment have terminated.

While Evolution Automator can be used to conduct regular experiments and has been adopted for that use in our research lab, its original purpose was to automate the evolution of hierarchical controllers. In this scenario, the user defines a configuration file with the various sub-controllers, including information on how they are connected and the evolutionary setup for each node (environment, controller, evolutionary algorithm, and evaluation function). The simulator then synthesizes and composes the hierarchical controller by picking the best controller for every hierarchical node, based on the post-evaluation results from multiple runs. The user can then test each node on the real robot and, if a controller proves to be difficult to transfer, the user can change the configuration for that particular node and restart the evolutionary process from that node and up. By reducing the manual labor and eliminating the sequential workflow necessary to synthesize control hierarchically, the time necessary to synthesize hierarchical controllers was reduced by an order of magnitude.

A.4.4 Common Interface, Raspberry Controller, and Robot Control Console

In the context of the CORATAM and HANCAD projects, we developed the control software for our aquatic robotic platform. The control software has been developed in order to allow any robot that can run the Java runtime environment to be controlled, and has been successfully tested with our aquatic robots and the Thymio II (Riedo et al., 2013). The software to control the robots is has three components:

Common Interface We developed the Common Interface in order to minimize the differences between the implementation of the robots' control systems in simulation and in reality, allowing us to run the same codebase both in JBotEvolver and onboard the real robots. The Common Interface implements communication, sensing and actuating components only once, therefore maintaining coherence between the simulated and real environment.

Raspberry Controller The Raspberry Controller is used in order to access the hardware layers of the real robots. The Common Interface can access data from the robot's hardware through the Raspberry Controller, such as retrieving data from a GPS, a compass, or a temperature sensor, sending data packets through, for instance, the Wi-Fi module (to other robots or a base station), or actuating the robot's motors.

Robot Control Console The Robot Control Console is a GUI that allows a user to: configure geo-entities, such as waypoints and geofences; deploy controllers and entities to groups of robots; access telemetry data from the robots, such as speed, location, orientation and other diagnostics information; see the robots' locations in real-time during field tests; and remotely update and restart a robot's onboard control software.

Appendix B

Aquatic Robotic Platform

During the CORATAM and HANCAD projects, we produced 10 operational aquatic robots, including mechanical, electronic, and software components (see Figure B.1). Prior to producing the final prototype, we did nine iterations of the design until a final model was reached. The robot developed for our experiments is a differential drive monohull boat. Each robot is relatively small (65 cm) and inexpensive (300 EUR). All our components are available as open-source software, and schematics, 3D models, and source code are available at `http://biomachineslab.com`. In this section, we describe the different components of the final robot design. A paper detailing the design and development process of our platform was accepted for publication at the international conference OCEANS:

- V. Costa, M. Duarte, T. Rodrigues, S. M. Oliveira and A. L. Christensen, "**Design and Development of an Inexpensive Aquatic Swarm Robotics System**", *in Proceedings of the MTS-IEEE OCEANS*, 2016, in press.

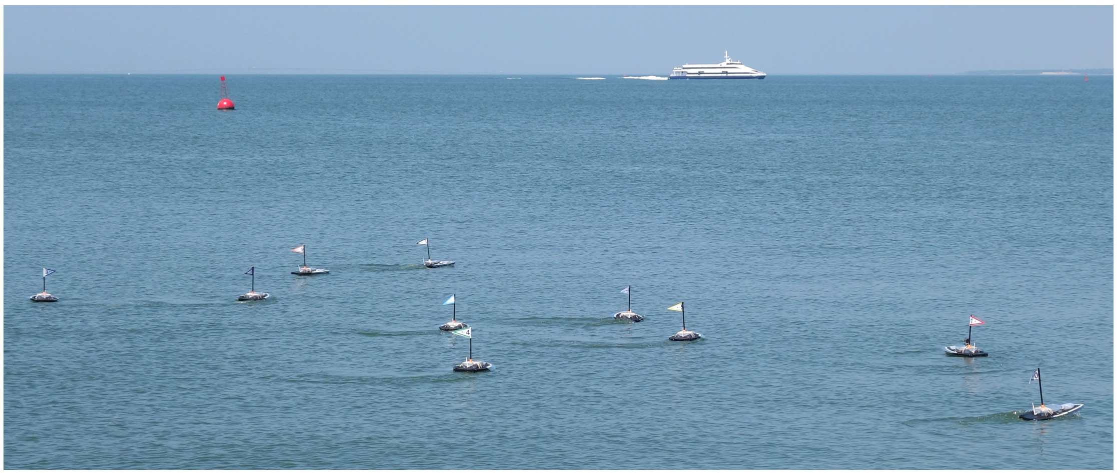
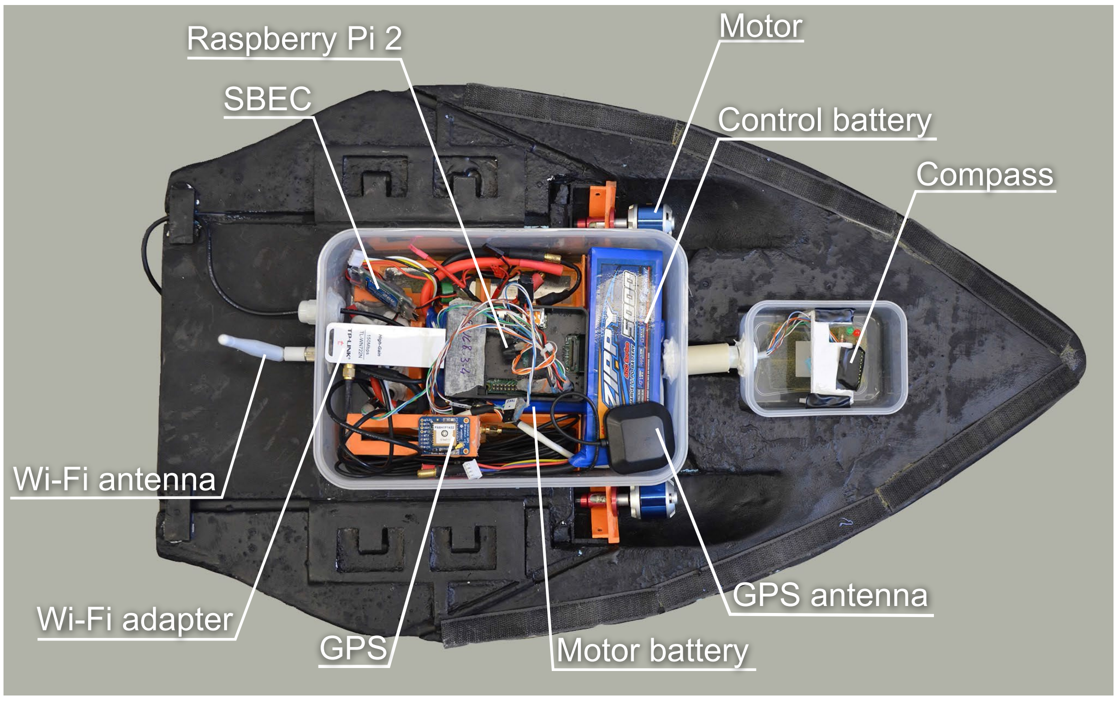


Figure B.1: A swarm of 10 robots performing a homing task during REX'15 at the Lisbon Naval Base, Alfeite.

B.1 Hardware Design and Specifications

In order to design the hull and the majority of the support components we used the Rhinoceros 5 CAD software. The hulls were then milled in an *Ouplan 3020* Computerized Numeric Cut (CNC) machine, and several support components where 3D printed with a *BQ Prusa i3 Hephestos*. The use of digital manufacturing and rapid prototyping techniques allowed us to quickly optimize the designs. For the hull production we used Extruded Polystyrene (XPS) material since it is buoyant, easily machinable, and inexpensive. In total, we produced 19 differentrobots hulls, nine of them prototypes. The final batch of operationalrobots were coated in Epoxy resin and fiber glass in order to waterproof the hull and make it resistant to impacts. Eachrobot has 12 different 3D printed components, which were produced in Polylactic Acid (PLA), which is an inexpensive biodegradable thermoplastic.

We used widely available hardware, and off-the-shelf sensors and motors in order to keep costs low, see Table B.1 and Figure B.2. The physical and movement properties of the robot are presented in Table B.2. The *Raspberry Pi 2* general-purpose computing platform was used for the control unit of each robot, and communication is achieved using Wi-Fi. A Kalman filter was applied to the GPS and compass readings of the real robots before they are used to compute sensory readings for the controller.



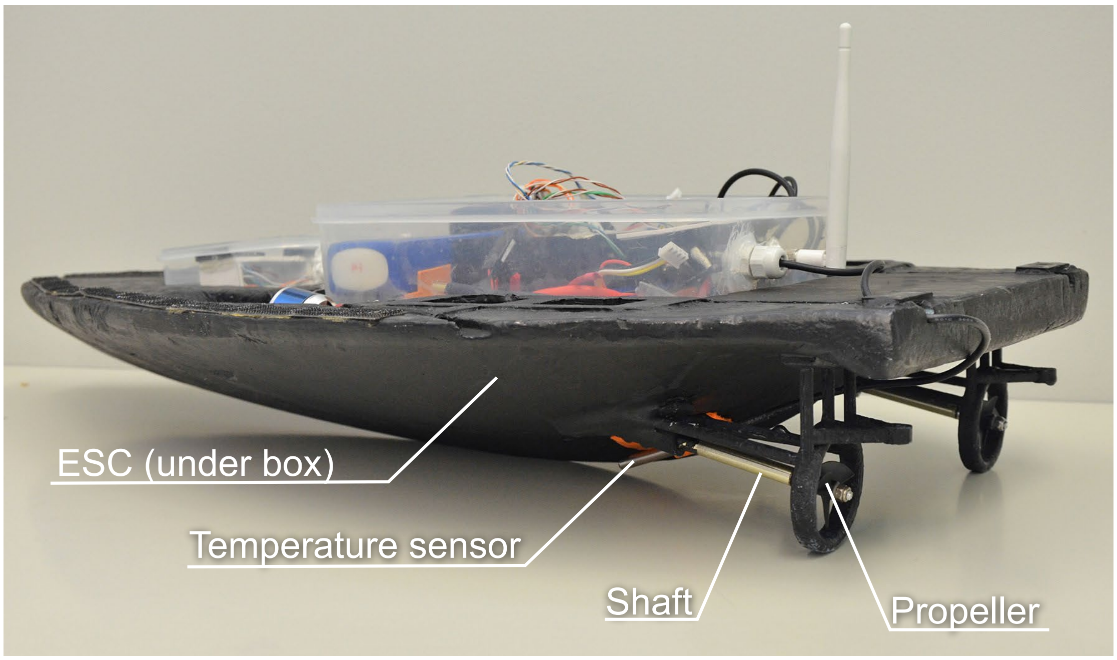


FIGURE B.2: Top and side view of the final robot prototype with a description of the components.

Robots can communicate with neighboring robots and with a base station using Wi-Fi. In order to assess the range of the chosen Wi-Fi adapter, we conducted empirical tests with the robots floating on the water surface, and achieved communication up to 40 m. When the swarm of robots is deployed, inter-robot communication is achieved by broadcasting messages. Each robot transmits a short status message, indicating its identification, position, and orientation. The status message is broadcast every second allowing neighboring robots to sense one another.

Table B.1: Components list.

Component	Make & Model
Motors (A)	NTM Prop Drive Series 28-30 A 750 kv / 140w
Motors (B)	Emax 2215/25 950 kv 2-3S
Shaft	4 mm drive shaft
Shaft sleeve	255 mm boat shaft sleeve
Propellers	3-blade 28 mm
ESC	HobbyKing 50 A Boat ESC
Control battery	Zippy 40C Series 5000 mA 3 S LiPo
Motor battery	Zippy 30C Series 8000 mA 3 S LiPo
GPS	Adafruit Ultimate GPS Breakout
Compass	STMicroelectronics LSM303D
Water temperature sensor	DS18B20
Onboard computer	Raspberry Pi 2
Wi-Fi adapter	TP-Link TL-WN722N
Hull material	Extruded Polystyrene (XPS), fiberglass, and epoxy
Structural components	3D printed Polylactic Acid (PLA)
Electronics enclosure	2.5 L watertight plastic box
Compass enclosure	0.4 L watertight plastic box

Table B.2: Measured movement dynamics and physical properties.

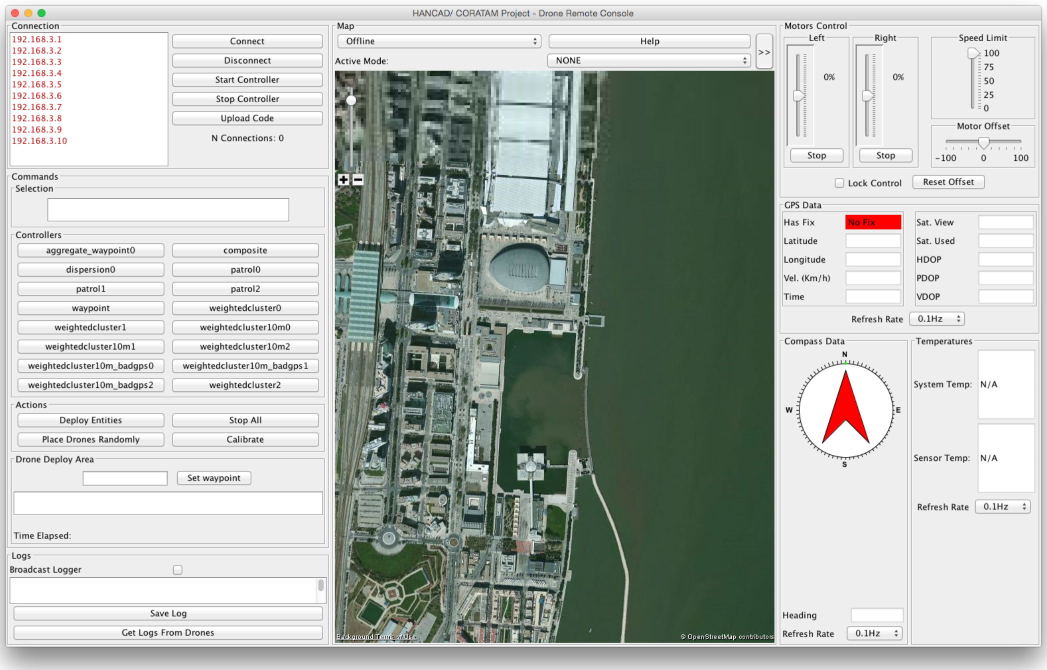
Parameter	Value	Parameter	Value
Size	65 × 40 × 15 cm	Weight	3 Kg
Minimum speed	0.3 m/s	Maximum speed	1.7 m/s
Maximum turning radius	90 °/s	Maximum acceleration	1.7 m/s^2
Time from full speed to stop	5 s	Autonomy	2-3 h

B.2 Onboard Software

The on-board Raspberry Pi runs the *Raspbian* Linux operating system, which is based on the Linux Debian distribution for ARM hard-float architecture. In terms of software, we use several different software components, some of them existing open-source components and others developed by us. To interact with the motor controllers, we use the *Servoblaster* module, which generates the control signal through the Raspberry Pi's GPIO. To interact with sensors and with the general input-outputs from the hardware, we use the *WiringPi* C library and the *Pi4J* library, which enables access to low-level functions. Our on-board *Raspberry Controller* software was developed in Java and interacts with the aforementioned libraries and modules. The software has the task of handling all aspects of therobot, ranging from low- and high-level control, sensor interfacing, communication and logging.

B.3 Remote Monitoring

In order to monitor the swarm, we developed the Robot Control Console application (see Figure B.3). This application monitored the messages that were sent by the robots in the swarm and displayed the locations in a map. To increase the range at which the inter-robot communication could be eavesdropped, an *Ubiquiti*



192.168.3.1
192.168.3.2
192.168.3.3
192.168.3.4
192.168.3.5
192.168.3.6
192.168.3.7
192.168.3.8
192.168.3.9
192.168.3.10

N Connections: 0

Map: Offline
Active Mode: NONE

Left: 0%
Right: 0%

Has Fix: No Fix
Refresh Rate: 0.1Hz

System Temp: N/A
Sensor Temp: N/A
Refresh Rate: 0.1Hz

Refresh Rate: 0.1Hz

FIGURE B.3: Robot Control Console

BULLET-M2-HP was used at the base station, effectively increasing the monitoring range up to 300 m. The application has several features to facilitate simple and flexible swarm control:

- Configuration of entities (waypoints, geofences and obstacles)
- Deployment of controllers and entities for groups ofrobots
- Automatic logging of sent commands and received broadcast communication messages
- Telemetry information (speed, location, orientation, diagnostics)
- Offline maps for field tests
- Synchronization between multiple instances of therobot Control Console running on different computers
- Remote update of arobot's onboard software

Appendix C

Experimental Parameters for Aquatic Tasks

Table C.1 lists the parameters used in our experiments, both in the simulation environment and in the real experiments. The noise that was applied in simulation during evolution is described below. All random numbers were drawn from uniform distributions. Regarding the movement model of the simulated robot, the values were taken by systematically performing tests with the real robot at different speeds and headings. The simulated dynamics were implemented taking into account the measurements obtained from these tests, and match the physical properties described in Table B.2.

GPS noise: upper limit for noise added to the robots' GPS unit at every simulation timestep. The value was taken from the technical specification of the GPS used in our robots.

Compass noise: upper limit for noise added to the robot's compass at every simulation timestep, chosen from empirical tests with the LSM303D compass.

Motor delay: fixed delay between executing a motor speed command and observing a reaction in the movement of the robot.

Heading offset: upper limit for noise added to the heading of the robot. The value is set individually for each robot at the beginning of a sample.

Speed offset: upper limit for noise added to the speed of the robot. The value is set individually for each robot at the beginning of a sample.

Motor output noise: upper limit for noise added to the output of the controllers at every simulation step.

Drift speed: upper limit for the translation component added to all robots. The value is chosen at the beginning of each sample, and applied at every simulation timestep.

TABLE C.1: Parameters used in the experiments.

Parameter	Value	Parameter	Value
NEAT			
Population size	150	Target species count	5
Recurrency allowed	true	Mutation prob.	25%
Prob. add node	3%	Prob. mutate bias	30%
Prob. add link	5%	Crossover prob.	20%
Simulation noise			
GPS noise	1.8 m	Compass noise	10°
Motor delay	500 ms	Heading offset	5%
Speed offset	10%	Motor output noise	5%
Drift speed	[0,0.1] m/s		
Homing task			
Trial length (real)	240 s	Trial length (evolution)	100 s
Generations	100	Waypoint distance (real)	40 m
Waypoint distance (evolution)	[0,50] m	Robot sensor range	20 m
Waypoint sensor range	10 m		
Dispersion task			
Trial length (real)	90 s	Trial length (evolution)	100 s
Generations	100	Target distance	20 m
Deploy area, 8 robots (real)	28 m × 28 m	Deploy area, 6 robots (real)	24 m × 24 m
Deploy area, 4 robots (real)	20 m × 20 m	Robot sensor range	40 m
Clustering task			
Trial length (real)	180 s	Trial length (evolution)	200 s
Generations	400	Robot starting distance	[20,40] m
Clustering threshold	7 m	Robot sensor range	40 m
Deploy area (real)	100 m × 100 m		
Sequential area monitoring task			
Trial length (real)	300 s	Trial length (evolution)	200 s
Generations	100	Robot sensor range	40 m
Monitoring area (real)	1 ha	Monitoring area (evolution)	[0.5,1.7] ha
Geo-fence sensor range	40 m		
Hierarchical intruder detection task			
Trial length (real)	500 s	Monitoring area (real)	100 m × 100 m
Generations (intruder task)	100	Robot sensor range	40 m
Intruder sensor range	20 m	Geo-fence sensor range	40 m
Hierarchical intruder detection task (Lampedusa scalability)			
Trial length (real)	24 h	Communication range	100 m
Intruder sensor range	50 m	Robot sensor range	100 m
Geo-fence sensor range	100 m		

Bibliography

A. Abreu and L. Correia. Fuzzy behaviors and behavior arbitration in autonomous vehicles. In *Proceedings of the Portuguese Conference on Artificial Intelligence (EPIA)*, pages 237–251. Springer, Berlin, Germany, 1999.

A. Abreu and L. Correia. A multi-layer behavior-based architecture for decision control in autonomous robots. In *Proceedings of the 4th European Workshop on Advanced Mobile Robots (EUROBOT)*. IEEE Computer Press, 2001.

C. Ampatzis, E. Tuci, V. Trianni, and M. Dorigo. Evolution of signaling in a multi-robot system: Categorization and communication. *Adaptive Behavior*, 16 (1):5–26, 2008.

E. Bahgeçi et al. Evolving aggregation behaviors for swarm robotic systems: A systematic case study. In *Proceedings of the 2005 IEEE Swarm Intelligence Symposium*, pages 333–340. IEEE Press, Piscataway, NJ, 2005.

G. Baldassarre, S. Nolfi, and D. Parisi. Evolving mobile robots able to display collective behaviors. *Artificial Life*, 9(3):255–267, 2003.

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

M. Basiri, F. Schill, D. Floreano, and P. U. Lima. Audio-based localization for swarms of micro air vehicles. In *Proceedings of the 2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4729–4734. IEEE Press, Piscataway, NJ, 2014.

M. A. Batalin and G. S. Sukhatme. Spreading out: A local approach to multi-robot coverage. In *Proceedings of the 5th International Symposium on Distributed Autonomous Robotic Systems (DARS)*, pages 373–382. Springer, Japan, 2002.

L. Bayındır. A review of swarm robotics tasks. *Neurocomputing*, 172(8):292–321, 2016.

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

J. A. Becerra, F. Bellas, J. S. Reyes, and R. J. Duro. Complex behaviours through modulation in autonomous robot control. In *Proceedings of the International Work-Conference on Artificial Neural Networks (IWANN)*, pages 717–724. Springer, Berlin, Germany, 2005.

S. A. Bedini. The role of automata in the history of technology. *Technology and Culture*, 5(1):24–42, 1964.

R. D. Beer and J. C. Gallagher. Evolving dynamical neural networks for adaptive behavior. *Adaptive Behavior*, 1(1):91–122, 1992.

G. Bekey and J. Yuh. The status of robotics. *IEEE Robotics & Automation Magazine*, 15(1):80–86, 2008.

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

J. Blynel and D. Floreano. Exploring the T-Maze: Evolving learning-like robot behaviors using CTRNNs. In *Applications of Evolutionary Computing*, pages 593–604. Springer, Berlin, Germany, 2003.

E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence: from natural to artificial systems*. Number 1. Oxford University Press, 1999.

G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 25(11), 2000.

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

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

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

S. Camazine, J.-L. Deneubourg, N. R. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau. *Self-organization in biological systems.* Princeton University Press, Princeton, NJ, 2003.

M. Castillo-Cagigal, A. Brutschy, A. Gutiérrez, and M. Birattari. Temporal task allocation in periodic environments. In *Swarm Intelligence*, volume 8667 of *Lecture Notes in Computer Science*, pages 182–193. Springer, 2014.

S. Celis, G. S. Hornby, and J. Bongard. Avoiding local optima with user demonstrations and low-level control. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 3403–3410. IEEE Press, Piscataway, NJ, 2013.

A. L. Christensen and M. Dorigo. Evolving an integrated phototaxis and hole avoidance behavior for a swarm-bot. In *Proceedings of the International Conference on the Simulation & Synthesis of Living Systems (ALIFE)*, pages 248–254. MIT Press, Cambridge, MA, 2006a.

A. L. Christensen and M. Dorigo. Incremental evolution of robot controllers for a highly integrated task. In *Proceedings of the 9th International Conference on Simulation of Adaptive Behavior (SAB)*, pages 473–484. Springer, Berlin, Germany, 2006b.

A. L. Christensen, R. O'Grady, M. Birattari, and M. Dorigo. Fault detection in autonomous robots based on fault injection and learning. *Autonomous Robots*, 24(1):49–67, 2008.

A. L. Christensen, R. O. Grady, and M. Dorigo. From fireflies to fault-tolerant swarms of robots. *IEEE Transactions on Evolutionary Computation*, 13(4): 754–766, 2009.

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

M. Clerc. *Particle swarm optimization*, volume 93. John Wiley & Sons, 2010.

J. Clune, K. Stanley, R. Pennock, and C. Ofria. On the performance of indirect encoding across the continuum of regularity. *IEEE Transactions on Evolutionary Computation*, 15(3):346–367, Jun 2011.

J. Coppens. The Lampedusa disaster: How to prevent further loss of life at sea? *TransNav, the International Journal on Marine Navigation and Safety of Sea Transportation*, 7(4):589–598, 2013.

L. Correia. Towards engineered evolutionary robotics, 1998.

V. Costa, M. Duarte, T. Rodrigues, S. M. Oliveira, and A. L. Christensen. Design and development of an inexpensive aquatic swarm robotics system. In *Proceedings of IEEE/MTS OCEANS*. IEEE Press, Piscataway, NJ, 2016. in press.

E. Şahin. Swarm robotics: From sources of inspiration to domains of application. In *Swarm Robotics*, volume 3342 of *Lecture Notes in Computer Science*, pages 10–20. Springer, Berlin, Germany, 2005.

A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret. Robots that can adapt like animals. *Nature*, 521(7553):503–507, 2015.

C. Darwin. *On the origin of species.* New York. Appleton and Co., 1859.

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

M. Dorigo, V. Trianni, E. Şahin, R. Groß, T. H. Labella, G. Baldassarre, S. Nolfi, J.-L. Deneubourg, F. Mondada, D. Floreano, et al. Evolving self-organizing behaviors for a swarm-bot. *Autonomous Robots*, 17(2-3):223–245, 2004.

M. Dorigo, E. Tuci, R. Groß, V. Trianni, T. H. Labella, S. Nouyan, C. Ampatzis, J.-L. Deneubourg, G. Baldassarre, S. Nolfi, et al. The swarm-bots project. In *Swarm Robotics*, volume 3342 of *Lecture Notes in Computer Science*, pages 31–44. Springer, 2005.

M. Dorigo, D. Floreano, L. M. Gambardella, F. Mondada, S. Nolfi, T. Baaboura, M. Birattari, M. Bonani, M. Brambilla, A. Brutschy, et al. Swarmanoid: a novel concept for the study of heterogeneous robotic swarms. *IEEE Robotics & Automation Magazine*, 20(4):60–71, 2013.

M. Duarte. Hierarchical evolution of robotic controllers for complex tasks. Master's thesis, University Institute of Lisbon (ISCTE-IUL), 2012.

M. Duarte, S. Oliveira, and A. L. Christensen. Towards artificial evolution of complex behavior observed in insect colonies. In *Proceedings of the Portuguese Conference on Artificial Intelligence (EPIA)*, pages 153–167. Springer, Berlin, Germany, 2011.

M. Duarte, S. Oliveira, and A. L. Christensen. Hierarchical evolution of robotic controllers for complex tasks. In *IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL EpiRob)*, pages 1–6. IEEE Press, Piscataway, NJ, 2012a.

M. Duarte, S. Oliveira, and A. L. Christensen. Automatic synthesis of controllers for real robots based on preprogrammed behaviors. In *Proceedings of the International Conference on Adaptive Behaviour (SAB)*, pages 249–258. Springer, Berlin, Germany, 2012b.

M. Duarte, S. M. Oliveira, and A. L. Christensen. Structured composition of evolved robotic controllers. In N. Siebel, editor, *Proceedings of the 5th International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems*, pages 19–25, 2012c.

M. Duarte, S. M. Oliveira, and A. L. Christensen. Hybrid control for large swarms of aquatic drones. In *Proceedings of the 14th International Conference on the Synthesis & Simulation of Living Systems (ALIFE)*, pages 785–792. MIT Press, Cambridge, MA, 2014a.

M. Duarte, S. M. Oliveira, and A. L. Christensen. Evolution of hierarchical controllers for multirobot systems. In *Proceedings of the 14th International Conference on the Synthesis & Simulation of Living Systems (ALIFE)*, pages 657–664. MIT Press, Cambridge, MA, 2014b.

M. Duarte, F. Silva, T. Rodrigues, S. M. Oliveira, and A. L. Christensen. JBotEvolver: A versatile simulation platform for evolutionary robotics. In *Proceedings of the 14th International Conference on the Synthesis & Simulation of Living Systems (ALIFE)*, pages 210–211. MIT Press, Cambridge, MA, 2014c.

M. Duarte, S. M. Oliveira, and A. L. Christensen. Evolution of hybrid robotic controllers for complex tasks. *Journal of Intelligent and Robotic Systems*, 78 (3–4):463–484, 2015.

M. Duarte, V. Costa, J. Gomes, T. Rodrigues, F. Silva, S. M. Oliveira, and A. L. Christensen. Evolution of collective behaviors for a real swarm of aquatic surface robots. *PLoS ONE*, 11(3):e0151834, 2016a. doi: 10.1371/journal.pone.0151834.

M. Duarte, J. Gomes, V. Costa, S. M. Oliveira, and A. L. Christensen. Hybrid control for a real swarm robotic system in an intruder detection task. In

Proceedings of the 18th European Conference on the Applications of Evolutionary Computation (EvoStar), pages 213–230. Springer International Publishing, 2016b.

M. Duarte, J. Gomes, V. Costa, T. Rodrigues, F. Silva, V. Lobo, M. Marques, S. M. Oliveira, and A. L. Christensen. Application of swarm robotic systems to marine environmental monitoring. In *Proceedings of IEEE/MTS OCEANS*. IEEE Press, Piscataway, NJ, 2016c. in press.

M. J. Er, B. H. Kee, and C. C. Tan. Design and development of an intelligent controller for a pole-balancing robot. *Microprocessors and Microsystems*, 26 (9-10):433–448, 2002.

D. Floreano. Evolutionary robotics in behavior engineering and artificial life. In *Evolutionary robotics. From intelligent robots to artificial life*. AAI Books, Ontario, Canada, 1998.

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

D. Floreano and F. Mondada. Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. In *Proceedings of the 3rd International Conference on Simulation of Adaptive Behavior (SAB)*, pages 421–430. MIT Press, Cambridge, MA, 1994.

D. Floreano and F. Mondada. Evolution of homing navigation in a real mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics*, 26(3):396–407, 1996.

D. Floreano, J.-C. Zufferey, and J.-D. Nicoud. From wheels to wings with evolutionary spiking circuits. *Artificial Life*, 11(1-2):121–138, Jan 2005.

G. Francesca, M. Brambilla, A. Brutschy, L. Garattoni, R. Miletitch, G. Podevijn, A. Reina, T. Soleymani, M. Salvaro, C. Pinciroli, V. Trianni, and M. Birattari. An experiment in automatic design of robot swarms. In *Proceedings of the 9th*

International Conference on Swarm Intelligence, pages 25–37. Springer, Berlin, Germany, 2014.

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

J. Gomes, P. Urbano, and A. L. Christensen. PMCNS: Using a Progressively Stricter Fitness Criterion to Guide Novelty Search. *International Journal of Natural Computing Research*, 4:1–19, 2014.

F. Gomez and R. Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5(3-4):317–342, 1997.

R. Groß and M. Dorigo. Evolution of solitary and group transport behaviors for autonomous robots capable of self-assembling. *Adaptive Behavior*, 16(5):285–305, 2008.

R. Groß and M. Dorigo. Towards group transport by swarms of robots. *International Journal of Bio-Inspired Computing*, 1(1–2):1–13, 2009.

R. Groß, M. Bonani, F. Mondada, and M. Dorigo. Autonomous self-assembly in swarm-bots. *IEEE Transactions on Robotics*, 22(6):1115–1130, 2006.

A. Gutiérrez, A. Campo, M. Dorigo, D. Amor, L. Magdalena, and F. Monasterio-Huelin. An open localization and local communication embodied sensor. *Sensors*, 8(11):7545–7563, 2008.

E. Haasdijk, A. Eiben, and G. Karafotias. On-line evolution of robot controllers by an encapsulated evolution strategy. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 1–7. IEEE Press, Piscataway, NJ, 2010.

E. Haasdijk, A. Atta-ul Qayyum, and A. E. Eiben. Racing to improve on-line, on-board evolutionary robotics. In *Proceedings of the 13th Genetic and Evolutionary Computation Conference (GECCO)*, pages 187–194. ACM Press, New York, NY, 2011.

J. Halloy, F. Mondada, S. Kernbach, and T. Schmickl. Towards bio-hybrid systems made of social animals and robots. In *Proceedings of the 2^{nd} International Conference on Biomimetic and Biohybrid Systems (LM)*, pages 384–386. Springer, Berlin, Germany, 2013.

H. Hamann, T. Schmickl, and K. Crailsheim. A hormone-based controller for evaluation-minimal evolution in decentrally controlled systems. *Artificial Life*, 18(2):165–198, 2012.

L. K. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.

I. Harvey, P. Husbands, and D. Cliff. Seeing the light: artificial evolution, real vision. In *Proceedings of the International Conference on Simulation of Adaptive Behavior (SAB)*, pages 392–401. MIT Press, Cambridge, MA, 1994.

I. Harvey, P. Husbands, D. Cliff, A. Thompson, and N. Jakobi. Evolutionary robotics: the sussex approach. *Robotics and Autonomous Systems*, 20(2–4): 205–224, 1997.

S. Hauert, J.-C. Zufferey, and D. Floreano. Evolved swarming without positioning information: an application in aerial communication relay. *Autonomous Robots*, 26(1):21–32, 2009.

M. Hehn and R. D'Andrea. A flying inverted pendulum. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 763–770. IEEE Press, Piscataway, NJ, 2011.

J. Hiller and H. Lipson. Automatic design and manufacture of soft robots. *IEEE Transactions on Robotics*, 28(2):457–466, 2012.

G. S. Hornby, S. Takamura, T. Yamamoto, and M. Fujita. Autonomous evolution of dynamic gaits with two quadruped robots. *IEEE Transactions on Robotics*, 21(3):402–410, 2005.

P. Husbands. Evolving robot behaviours with diffusing gas networks. In *Proceedigs of the European Workshop Evolutionary Robotics (EvoRobot)*, pages 71–86. Springer, Berlin, Germany, 1998.

A. J. Ijspeert, A. Crespi, D. Ryczko, and J.-M. Cabelguen. From swimming to walking with a salamander robot driven by a spinal cord model. *Science*, 315 (5817):1416–1420, 2007.

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

C. Jones and M. Mataríc. Behavior-based coordination in multi-robot systems. In S. Ge and F. Lewis, editors, *Autonomous Mobile Robots: Sensing, Control, Decision Making and Applications*, pages 549–569. CRC Press, Boca Raton, FL, 2006.

S. Kernbach, E. Meister, F. Schlachter, K. Jebens, M. Szymanski, J. Liedke, D. Laneri, L. Winkler, T. Schmickl, R. Thenius, et al. Symbiotic robot organisms: Replicator and symbrion projects. In *Proceedings of the 8th workshop on performance metrics for intelligent systems*, pages 62–69. ACM, 2008.

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

T. Larsen and S. T. Hansen. Evolving composite robot behaviour - a modular architecture. In *Proceedings of the International Workshop on Robot Motion and Control (RoMoCo)*, pages 271–276. IEEE Press, Piscataway, NJ, 2005.

W.-P. Lee. Evolving complex robot behaviors. *Information Sciences*, 121(1-2): 1–25, 1999.

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

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

Q. Lindsey, D. Mellinger, and V. Kumar. Construction with quadrotor teams. *Autonomous Robots*, 33(3):323–336, 2012.

H. Lipson and J. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406(6799):974–978, 2000.

D. Lutterbeck. Policing migration in the mediterranean. *Mediterranean Politics*, 11(1):59–82, 2006.

L. Marques, U. Nunes, and A. T. de Almeida. Particle swarm-based olfactory guided search. *Autonomous Robots*, 20(3):277–287, 2006.

J.-A. Meyer, P. Husbands, and I. Harvey. Evolutionary robotics: A survey of applications and problems. In *Proceedings of the European Workshop on Evolutionary Robotics (EvoRobot)*, pages 1–21. Springer, Berlin, Germany, 1998.

J.-A. Meyer, S. Doncieux, D. Filliat, and A. Guillot. Evolutionary approaches to neural control of rolling, walking, swimming and flying animats or robots. In *Biologically Inspired Robot Behavior Engineering*, volume 109 of *Studies in Fuzziness and Soft Computing*, pages 1–43. Springer, Berlin, Germany, 2003.

O. Miglino, H. H. Lund, and S. Nolfi. Evolving mobile robots in simulated and real environments. *Artificial Life*, 2(4):417–434, 1996.

R. Moioli, P. Vargas, F. Von Zuben, and P. Husbands. Towards the evolution of an artificial homeostatic system. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC)*, pages 4023–4030. IEEE Press, Piscataway, NJ, 2008.

F. Mondada, E. Franzi, and A. Guignard. The development of khepera. In *Experiments with the Mini-Robot Khepera, Proceedings of the First International Khepera Workshop*, pages 7–14, 1999.

F. Mondada, G. C. Pettinaro, A. Guignard, I. W. Kwee, D. Floreano, J.-L. Deneubourg, S. Nolfi, L. M. Gambardella, and M. Dorigo. Swarm-bot: A new distributed robotic concept. *Autonomous Robots*, 17(2-3):193–221, 2004.

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

J. M. Moore, A. J. Clark, and P. K. McKinley. Evolution of station keeping as a response to flows in an aquatic robot. In *Proceedings of the 15th Genetic and Evolutionary Computation Conference (GECCO)*, pages 239–246. ACM Press, New York, NY, 2013.

J.-B. Mouret and S. Doncieux. Incremental evolution of animats' behaviors as a multi-objective optimization. In *Proceedings of the 10th International Conference on Simulation of Adaptive Behaviour (SAB)*, pages 210–219. Springer, Berlin, Germany, 2008.

H. Nakamura, A. Ishiguro, and Y. Uchilkawa. Evolutionary construction of behavior arbitration mechanisms based on dynamically-rearranging neural networks. In *Proceedings of Congress on Evolutionary Computation (CEC)*, pages 158–165. IEEE Press, Piscataway, NJ, 2000.

A. L. Nelson, G. J. Barlow, and L. Doitsidis. Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems*, 57(4): 345–370, 2009.

S. Nolfi. Evolutionary robotics: Exploiting the full power of self-organization. *Connection Science*, 10(3–4):167–184, 1998.

S. Nolfi and D. Floreano. *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines.* MIT Press, Cambridge, MA, 2000.

S. Nolfi and D. Parisi. Evolving non-trivial behaviors on real robots: an autonomous robot that picks up objects. In *Proceedings of the Congress of the Italian Association for Artificial Intelligence (AI*IA)*, pages 187–198. Springer, Berlin, Germany, 1995.

N. Noskov, E. Haasdijk, B. Weel, and A. E. Eiben. MONEE: Using parental investment to combine open-ended and task-driven evolution. In *Proceedings of the 16th European Conference on the Applications of Evolutionary Computation (EvoStar)*, pages 569–578. Springer, Berlin, Germany, 2013.

R. O'Grady, A. L. Christensen, and M. Dorigo. SWARMORPH: multirobot morphogenesis using directional self-assembly. *IEEE Transactions on Robotics*, 25 (3):738–743, 2009.

A. M. Okamura, N. Smaby, and M. R. Cutkosky. An overview of dexterous manipulation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 255–262. IEEE Press, Piscataway, NJ, 2000.

R. O'Grady, R. Groß, F. Mondada, M. Bonani, and M. Dorigo. Self-assembly on demand in a group of physical autonomous mobile robots navigating rough terrain. In *Advances in Artificial Life*, pages 272–281. Springer, 2005.

G. Pini and E. Tuci. On the design of neuro-controllers for individual and social learning behaviour in autonomous robots: an evolutionary approach. *Connection Science*, 20(2-3):211–230, 2008.

T. Praczyk. Using augmenting modular neural networks to evolve neuro-controllers for a team of underwater vehicles. *Soft Computing*, 18(12):2445–2460, 2014.

J. H. Reif and H. Wang. Social potential fields: A distributed behavioral control for autonomous robots. *Robotics and Autonomous Systems*, 27(3):171–194, 1999.

C. W. Reynolds. Evolution of corridor following behavior in a noisy world. In *Proceedings of the International Conference on Simulation of Adaptive Behavior (SAB)*, pages 402–410. MIT Press, Cambridge, MA, 1994.

F. Riedo, M. Chevalier, S. Magnenat, and F. Mondada. Thymio ii, a robot that grows wiser with children. In *IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO)*, pages 187–193. IEEE Press, Piscataway, NJ, 2013.

T. Rodrigues, M. Duarte, S. M. Oliveira, and A. L. Christensen. What you choose to see is what you get: an experiment with learnt sensory modulation in a robotic foraging task. In *Proceedings of the 16th European Conference on the Applications of Evolutionary Computation (EvoStar)*, pages 789–801. Springer, Berlin, Germany, 2014.

T. Rodrigues, M. Duarte, M. Figueiró, V. Costa, S. M. Oliveira, and A. L. Christensen. Overcoming limited onboard sensing in swarm robotics through local communication. In *Transactions on Computational Collective Intelligence XX*, volume 9420 of *Lecture Notes in Computer Science*. 2015a. In press.

T. Rodrigues, M. Duarte, S. M. Oliveira, and A. L. Christensen. Beyond onboard sensors in robotic swarms: Local collective sensing through situated communication. In *Proceedings of the International Conference on Agents and Artificial Intelligence (ICAART)*, pages 111–118. SCITEPRESS, Lisbon, Portugal, 2015b.

P. Romano, L. Nunes, A. L. Christensen, M. Duarte, and S. M. Oliveira. Genome variations: Effects on the robustness of neuroevolved swarm controllers. In *Proceedings of the Iberian Conference on Robotics (ROBOT)*, pages 309–319. Springer, Berlin, Germany, 2015.

C. Rossi, F. Russo, and F. Russo. *Ancient Engineers& Inventions*, volume 8 of *History of Mechanism and Machine Science*. Springer Netherlands, 2009.

T. Schmickl, R. Thenius, C. Moslinger, J. Timmis, A. Tyrrell, M. Read, J. Hilder, J. Halloy, A. Campo, C. Stefanini, L. Manfredi, S. Orofino, S. Kernbach, T. Dipper, and D. Sutantyo. CoCoRo –The Self-Aware Underwater Swarm. In *Proceedings of the 5th IEEE Conference on Self-Adaptive and Self-Organizing Systems Workshops (SASOW)*, pages 120–126. IEEE Press, Piscataway, NJ, 2011.

F. Silva, L. Correia, and A. L. Christensen. Speeding up online evolution of robotic controllers with macro-neurons. In *Proceedings of the 16th European Conference on the Applications of Evolutionary Computation (EvoStar)*, pages 765–776. Springer, Berlin, Germany, 2014a.

F. Silva, M. Duarte, S. M. Oliveira, L. Correia, and A. L. Christensen. The case for engineering the evolution of robot controllers. In *Proceedings of the International Conference on the Simulation & Synthesis of Living Systems (ALIFE)*, pages 703–710. MIT Press, Cambridge, MA, 2014b.

F. Silva, M. Duarte, L. Correia, S. M. Oliveira, and A. L. Christensen. Open issues in evolutionary robotics. *Evolutionary Computation*, 2015a. In press.

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

H. Silva, S. M. Oliveira, and A. L. Christensen. Conillon: A lightweight distributed computing platform for desktop grids. In *Proceedings of the 6th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–6. IEEE Press, Piscataway, NJ, 2011.

H. A. Simon. *The architecture of complexity.* Springer, 1991.

H. M. Smith. Synchronous flashing of fireflies. *Science*, 82(2120):151–152, 1935.

O. Soysal, E. Bahçeci, and E. Sahin. Aggregation in swarm robotic systems: Evolution and probabilistic control. *Turkish Journal of Electrical Engineering & Computer Sciences*, 15(2):199–225, 2007.

V. Sperati, V. Trianni, and S. Nolfi. Evolving coordinated group behaviours through maximisation of mean mutual information. *Swarm Intelligence*, 2(2-4):73–95, 2008.

V. Sperati, V. Trianni, and S. Nolfi. Self-organised path formation in a swarm of robots. *Swarm Intelligence*, 5(2):97–119, 2011.

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

K. Stanley and R. Miikkulainen. A taxonomy for artificial embryogeny. *Artificial Life*, 9(2):93–130, Mar 2003.

K. O. Stanley. Why evolutionary robotics will matter. In *New Horizons in Evolutionary Robotics*, volume 341 of *Studies in Computational Intelligence*, chapter 3, pages 37–41. Springer, Berlin, Germany, 2011.

K. O. Stanley and R. P. Miikkulainen. *Efficient evolution of neural networks through complexification*. Computer Science Department, University of Texas at Austin, 2004.

K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212, 2009.

M. L. Stein. *Interpolation of Spatial Data: Some Theory for Kriging*. Springer Science & Business Media, 2012.

D. Tarapore, P. U. Lima, J. Carneiro, and A. L. Christensen. To err is robotic, to tolerate immunological: fault detection in multirobot systems. *Bioinspiration & Biomimetics*, 10(1):016014, 2015.

E. C. Tolman and C. H. Honzik. Introduction and removal of reward, and maze performance in rats. *University of California Publications in Psychology*, 4(16–17):257–275, 1930.

A. B. L. Torta, M. A. Kramer, C. Thorn, D. J. Gibson, Y. Kubota, A. M. Graybiel, and N. J. Kopell. Dynamic cross-frequency couplings of local field potential oscillations in rat striatum and hippocampus during performance of a T-maze task. *Proceedings of the National Academy of Sciences*, 105(51):20517–20522, 2008.

V. Trianni and S. Nolfi. Self-Organising Sync in a Robotic Swarm. A Dynamical System View. *IEEE Transactions on Evolutionary Computation*, 14(4):722–741, 2009.

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

V. Trianni, R. Groß, T. H. Labella, E. Şahin, and M. Dorigo. Evolving aggregation behaviors in a swarm of robots. In *Proceedings of the 7th European Conference on Artificial Life (ECAL)*, pages 865–874. Springer, Berlin, Germany, 2003.

V. Trianni, S. Nolfi, and M. Dorigo. Cooperative hole avoidance in a swarm-bot. *Robotics and Autonomous Systems*, 54(2):97–103, 2006.

E. Tuci, V. Trianni, and M. Dorigo. 'Feeling' the flow of time through sensorimotor co-ordination. *Connection Science*, 16(4):301–324, 2004.

E. Tunstel. Mobile robot autonomy via hierarchical fuzzy behavior control. In *Proceedings of the International Symposium on Robotics and Manufacturing (WAC)*, pages 837–842, New York, 1996. ASME Press.

F. Velez, A. Nadziejko, A. L. Christensen, S. M. Oliveira, T. Rodrigues, V. Costa, M. Duarte, F. Silva, and J. Gomes. Wireless sensor and networking technologies for swarms of aquatic surface drones. In *Proceedings of the IEEE 82nd Vehicular Technology Conference (VTC Fall)*, pages 1–2, 2015.

I. Wagner, M. Lindenbaum, A. M. Bruckstein, et al. Distributed covering by ant-robots using evaporating traces. *IEEE Transactions on Robotics and Automation*, 15(5):918–933, 1999.

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

L. D. Whitley. Fundamental principles of deception in genetic search. In *Proceedings of the 1st Workshop on Foundations of Genetic Algorithms (FOGA)*, pages 221–241. Morgan Kaufmann, San Mateo, CA, 1991.

A. T. Winfree. *The Geometry of Biological Time*, volume 12 of *Interdisciplinary Applied Mathematics*. Springer Science & Business Media, 2001.

S. Wischmann, K. Stamm, and F. Wörgötter. Embodied evolution and learning: The neglected timing of maturation. In *Proceedings of the 9th European Conference on Artificial Life (ECAL)*, pages 284–293. Springer, Berlin, Germany, 2007.

R.-j. Yan, S. Pang, H.-b. Sun, and Y.-j. Pang. Development and missions of unmanned surface vehicle. *Journal of Marine Science and Application*, 9(4): 451–457, 2010.

J. C. Zagal and J. Ruiz-Del-Solar. Combining simulation and reality in evolutionary robotics. *Journal of Intelligent & Robotic Systems*, 50(1):19–39, 2007.