Distributed Multi-Robot Learning using Particle Swarm Optimization

THÈSE N^{O} 6707 (2015)

PRÉSENTÉE LE 21 AOÛT 2015

À LA FACULTÉ DE L'ENVIRONNEMENT NATUREL, ARCHITECTURAL ET CONSTRUIT
LABORATOIRE DE SYSTÈMES ET ALGORITHMES INTELLIGENTS DISTRIBUÉS
PROGRAMME DOCTORAL EN GÉNIE ÉLECTRIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Ezequiel Leonardo DI MARIO

acceptée sur proposition du jury:

Prof. C. N. Jones, président du jury
Prof. A. Martinoli, directeur de thèse
Prof. L. Gambardella, rapporteur
Dr R. Gross, rapporteur
Prof. A. Ijspeert, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2015

Acknowledgements

This work would not have been possible without the help of many people. I would like to start by thanking my advisor Alcherio Martinoli for his guidance and support, always reminding me of the big picture while still paying attention to detail.
I would like to thank Iñaki Navarro, who joined the project in 2013, and contributed to many aspects of this research with his experience in flocking and real robot experiments.
I would also like to thank Zeynab Talebpour for sharing her expertise in Reinforcement Learning, Steven Roelofsen for his help with the french abstract, and students Guillaume Jornod, Ayberk Ozgur and Alexandre Cherpillod, who I had the pleasure to supervise.
Finally, I would like to thank all the members of the DISAL lab for their support and feedback. In particular, I am very grateful to José Nuno Pereira, Chris Evans, Maria Boberg and Adrian Arfire, who accompanied me for most of the ride; Sven Gowal, Amanda Prorok, Grégory Mermoud and Massimo Mastrangeli, who guided me through the first years; and Corinne Farquharson, Denis Rochat, and Emmanuel Droz who helped me with all the logistics and administrative aspects.

Lausanne, 25 Mai 2015 Ezequiel Di Mario

Abstract

This thesis studies the automatic design and optimization of high-performing robust controllers for mobile robots using exclusively on-board resources. Due to the often large parameter space and noisy performance metrics, this constitutes an expensive optimization problem.

Population-based learning techniques have been proven to be effective in dealing with noise and are thus promising tools to approach this problem. We focus this research on the Particle Swarm Optimization (PSO) algorithm, which, in addition to dealing with noise, allows a distributed implementation, speeding up the optimization process and adding robustness to failure of individual agents.

In this thesis, we systematically analyze the different variables that affect the learning process for a multi-robot obstacle avoidance benchmark. These variables include algorithmic parameters, controller architecture, and learning and testing environments. The analysis is performed on experimental setups of increasing evaluation time and complexity: numerical benchmark functions, high-fidelity simulations, and experiments with real robots.

Based on this analysis, we apply the distributed PSO framework to learn a more complex, collaborative task: flocking. This attempt to learn a collaborative task in a distributed manner on a large parameter space is, to our knowledge, the first of such kind.

In addition, we address the problem of noisy performance evaluations encountered in these robotic tasks and present a distributed PSO algorithm for dealing with noise suitable for resource-constrained mobile robots due to its low requirements in terms of memory and limited local communication.

Keywords: Distributed Learning, Multi-Robot Systems, Particle Swarm Optimization

Résumé

Cette thèse étudie la conception automatique et l'optimisation de contrôleurs à la fois robustes et de haute performance pour robots mobiles en utilisant exclusivement des ressources disponibles à bord. À cause de la dimensionnalité de l'espace des paramètres communément grande et des métriques de performance bruitées, il s'agit d'un problème d'optimisation couteux.

Les techniques d'apprentissage basées sur des populations de solutions se sont révélées efficaces pour traiter avec ces problèmes bruités et sont donc des outils prometteurs pour les approcher. Nous focalisons cette recherché sur l'Optimisation par Essaims Particuliers (PSO pour Particle Swarm Optimization en anglais), qui en plus de traiter avec le bruit, permet une implémentation distribuée, accélérant le processus d'optimisation et ajoutant de la robustesse vis à vis l'échec des agents individuels.

Dans cette thèse, nous analysons systématiquement les différentes variables qui affectent le processus d'apprentissage pour notre tâche de référence qu'est l'évitement d'obstacles multi-robot. Ces variables comprennent les paramètres algorithmiques, l'architecture de contrôle, et les environnements d'apprentissage et d'évaluation. L'analyse est effectuée avec des configurations d'expérimentation dont le temps d'évaluation et la complexité vont croissants : des fonctions de référence numériques, des simulations de haute fidélité, et des expériences avec de vrais robots.

Sur la base de cette analyse, nous appliquons la méthode de PSO distribuée à l'apprentissage d'une tâche plus complexe et coopérative : le mouvement coordonné en banc. Cette tentative d'apprendre une tâche coopérative de manière distribuée sur un espace de paramètres de grande dimension est, à notre connaissance, la première de ce genre.

En outre, nous abordons le problème des métriques de performance bruitées rencontré dans ces tâches robotiques et présentons un algorithme PSO distribué adapté à la fois au bruit induit par le problème d'apprentissage et aux ressources limitées du robot grâce à ses besoins réduits en termes de mémoire et de communication locale limitée.

Mots clés : Apprentissage Automatique Distribué, Systèmes Multi-Robots, Optimisation par Essaims Particulaires

Contents

Introduction Part I

1 Learning Robotic Controllers: An Expensive Optimization Problem

Human design of high-performing robotic controllers is not a trivial task for a number of reasons. In the first place, even the simplest of modern robots have a large number of sensors and actuators, which implies a large number of control parameters to choose. Secondly, real systems often present discontinuities and nonlinearities, making it difficult to apply well-understood linear control techniques. Finally, when porting the designed controller to real robots there might be an unexpected drop in performance due to a number of factors such as imperfections in fabrication, changes in the environment, or modeling inaccuracies.

Engineers have developed a set of strategies for dealing with these challenges. For instance, in the framework of model-based approaches, the use of models reduces the number of parameters in the system; if hierarchical models are considered, nonlinearities and discontinuities can be addressed in lower levels facilitating the task of the high-level controllers; and the performance of the deployed system can be improved by refining the models in an iterative fashion until the desired criteria are met.

Machine-learning techniques are an alternative, promising tool to aid human designers in addressing the previously mentioned challenges. In particular, population-based, evaluative, metaheuristic methods are effective in dealing with high-dimensional search spaces with discontinuities and nonlinearities, as seen for instance in black-box optimization competitions [1].

Furthermore, the learning process can be implemented fully on-board [2], [3], which enables automatic adaptation to the underlying hardware and environment, and has the potential to find innovative solutions not foreseen by human designers.

However, the main drawback of working in an on-line, evaluative framework is the large amount of time needed to characterize the performance of candidate controller solutions, which is substantially larger than that required to generate them. Moreover, due to several sources of uncertainty, such as sensor noise, manufacturing tolerances, or lack of strict coordination in multi-robot settings, it may be necessary to re-evaluate some solutions to build

sufficient statistics for meaningful adaptation. Because of these two reasons, large evaluation time and uncertainties in the performance measurement, the adaptation process is considered an expensive optimization problem.

Implementing the adaptation process in a distributed fashion brings two distinct advantages. Firstly, it reduces the required total evaluation time through parallel evaluations. Secondly, it increases robustness by avoiding a critical point of failure, which is of particular interest in real multi-robot implementations.

The aim of this thesis is therefore to explore fully on-board distributed strategies for the automatic synthesis of robotic controllers. We focus this research on the Particle Swarm Optimization (PSO) algorithm [4], as it has been shown to scale well to high-dimensional problems [5], it is well suited for distributed multi-robot implementations due to its limited local communication and low computational requirements [6], and it has been shown to outperform Genetic Algorithms for limited-time learning of obstacle avoidance under the presence of noise [7].

Our approach consists in systematically analyzing the different factors that affect the learning process. These include algorithmic parameters, controller architecture, level of collaboration required in the task, and learning and testing environments. The analysis is performed on experimental setups of increasing evaluation time and complexity: numerical benchmark functions, physics-based simulations, and experiments with real robots.

In order to organize our contributions, we have divided the thesis into four parts. Part I contains the present introduction, related work, and the materials and methods that are common throughout the thesis.

Part II presents an analysis of the factors involved in distributed multi-robot learning using multi-robot obstacle avoidance as a benchmark task. Each chapter in this part deals with a particular factor of interest for the learning problem, so that it can be easily referenced in the future. The analyzed factors include algorithmic parameters (Chapter 5), learning and testing environments (Chapter 6), and controller architecture (Chapter 7). We also perform a comparison of the PSO approach with Q-learning, another evaluative algorithm used for robotic learning in Chapter 8.

Part III groups two contributions that go beyond the obstacle avoidance task in multi-robot systems: learning collaborative behaviors and enhancing noise-resistance. Chapter 9 shows how, based on the analysis performed in Part II, we apply the distributed PSO framework to learn a more complex, collaborative task: flocking. This attempt to learn a collaborative task in a distributed manner on a large parameter space is, to our knowledge, the first of such kind.

In Chapter 10 we go deeper into the challenges arising from noisy performance evaluations that we have identified in robotic learning, and how they affect PSO. Then, based on this analysis, in Chapter 11 we present an improved distributed PSO algorithm for dealing with

noise suitable for resource-constrained mobile robots due to its low requirements in terms of memory and limited local communication. Chapter 12 describes more in detail the role of the parameters specific to this algorithm.

Finally, Part IV concludes this thesis with a discussion that integrates the results from previous chapters and an outlook for future work.

2 Related Work

Learning algorithms can be divided into two categories depending on the number of candidate solutions that they consider at the same time: population-based and single-point search [8]. Single-point search algorithms, sometimes also called trajectory-based or hill-climbing, refine a single candidate solution iteratively. The Reinforcement Learning (RL) family of algorithms [9] constitutes an example of single-point search learning which is popular in the robotics domain [10].

Population-based algorithms maintain a set of candidate solutions at any given time that are used to derive new solutions. As examples in this category we can cite Genetic Algorithms (GA) [11], Evolution Strategies (ES) [12], Ant Colony Systems (ACO) [13], and Particle Swarm Optimization (PSO) [4].

2.1 PSO

Particle Swarm Optimization (PSO) is a relatively new metaheuristic originally introduced by Kennedy and Eberhart [4]. PSO is inspired by the movement of flocks of birds, and represents a set of candidate solutions as a population or swarm of particles moving in a multi-dimensional space.

PSO has been shown to be stable when the dimensionality of the search space is increased, i.e., when increasing the number of dimensions with fixed settings for PSO parameters, the algorithm behaves in a similar manner [5].

Because of its simplicity and versatility, PSO has been used in a wide range of applications such as antenna design, communication networks, finance, power systems, and scheduling [14]. Within the robotics domain, popular topics are robotic search [15]–[17], path planning [18]–[21], odor source localization [22]–[24], and gait optimization [25].

2.2 Distributed PSO

PSO is well suited for distributed implementations due to its distinct individual and social components and the use of the neighborhood concept. Most of the work on distributed implementations has been focused on benchmark functions running on computational clusters [26]–[28], as opposed to implementations on real robots as used in this work.

Moreover, implementations with mobile robots are mostly applied to odor source localization [22], [23], and robotic search [16]. In these applications, the particles' position is usually directly matched to the robots' position in the arena. Thus, the search is conducted in two dimensions and with few or even only one local extrema. For these reasons, even though robotic search is a challenging practical task, it does not represent a complex optimization problem. On the other hand, Pugh et al. used PSO to optimize a set of 24 control parameters for a multi-robot obstacle avoidance task [6], which constitutes a high-dimensional optimization problem. The algorithm was implemented on-board on real robots in a distributed fashion.

2.3 Noise

Most of the research on optimization in noisy environments has focused on evolutionary algorithms [29]. Fitzpatrick and Grefenstette studied the trade-off between population size and sample size for Genetic Algorithms in noisy environments, showing that for a constant total evaluation time more efficient search may result from increasing the population size while decreasing the number of samples for re-evaluations, testing this hypothesis on one of De Jong's test functions and in an image registration problem [30]. Stagge proposed a variation in which only the best solutions are re-evaluated instead of the whole population [31]. Arnold and Beyer studied the effect of different noise distributions on the performance of evolution strategies, concluding that significant differences occur when the noise distribution assumed to be Gaussian is skewed, biased, or presents unbounded variance [32].

The performance of PSO under noise has not been studied so extensively. Parsopoulos and Vrahatis showed that standard PSO was able to cope with noisy and continuously changing environments, and even suggested that noise may help to avoid local minima [33]. Pan et al. proposed a hybrid PSO-Optimal Computing Budget Allocation (OCBA) technique for function optimization in noisy environments [34]. It should be noted that these two studies were conducted on benchmark functions with an additive Gaussian noise model, and one of the goals of our research is to extend this analysis to the noisy performance evaluations found in multi-robot learning. Pugh et al. [7] introduced a noise-resistant variation of PSO that consists in re-evaluating personal best positions and averaging them with the previous evaluations, which was inspired by the Genetic Algorithm variation proposed by Stagge [31], and applied to the learning of multi-robot obstacle avoidance. This noise-resistant PSO will be the basis for our study of noise in Chapter 10.

2.4 Comparison with Other Algorithms

Since its introduction, PSO has often been compared with Genetic Algorithms and other metaheuristics. In their original article, Eberhart and Kennedy mention that PSO required similar number of evaluations to reach the same performance as GA on the Schaffer f_6 function [4], [35]. Kennedy and Spears analyzed the performance of three versions of Genetic Algorithms and a binary PSO variation on a factorial time-series experiment, and found that PSO was the only algorithm that could locate the global optimum on every trial [36]. Fourie and Groenwold showed that PSO was superior to GA on a set of benchmark structural optimization problems [37]. Hansen et al. compared PSO with Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES), concluding that while PSO outperforms CMA-ES in separable, ill-conditioned functions, CMA-ES performs orders of magnitude better than PSO in rotated and non-separable, ill-conditioned functions [38]. Pugh et al. showed that PSO could outperform Genetic Algorithms on benchmark functions with added noise and for limited-time learning of multi-robot obstacle avoidance [6], [7]. Given enough evaluation time all algorithms performed similarly, but when the total evaluation time was reduced by a factor of 100, the noise-resistant PSO variation consistently outperformed the other algorithms. Pugh also showed that PSO can achieve satisfactory performances with low population sizes; this result is of particular interest for our work because it allows a smaller number of robots to be used while still leaving the optimization process robust to connectivity issues between the robots.

When comparing optimization algorithms, it should be noted that for any algorithm, any elevated performance over one class of problems is offset by poorer performance over another class, a result which is known as the “no free lunch” theorem for optimization [39]. Therefore, we cannot consider an algorithm to be universally better. However, we can summarize the main characteristics that make PSO well-suited for the particular problem of distributed multi-robot learning, based on the related work we have referenced in this Chapter:

- good performance in a wide range of problems
- good performance under noise
- good performance in high-dimensional search spaces
- good performance with low population sizes
- distributed implementation
- limited local communication within a neighborhood
- limited computational requirements

2.5 Multi-Robot Learning

Learning in multi-robot scenarios requires addressing the specific problem of credit assignment, i.e., determining how much an agent's individual actions affect the performance of the whole team. A simplifying approach is to use homogeneous controllers with a group reward signal [40]–[42], which may be seen as effectively turning the problem into single agent learning [43]. Group performance can also be estimated locally, which introduces an uncertainty that depends on the agents' perception and communication range [44]. An alternative is to assign rewards based only on individual performance, which may lead to faster learning due to greedy behaviors but may not necessarily converge to optimal team performance [45]. Individual rewards may also increase the homogeneity within the team, hindering the appearance of specialization [43], [46].

Previous works have shown that it is feasible to use learning to generate collaborative behaviors either by centralized approaches, or by distributed approaches on low-dimensional search spaces. For instance, [47] and [48] addressed the topic of learning in multi-robot teams using a small number of parameters per robot. In the case of [49], [50] and [51] learning has been done in a centralized manner, using homogeneous controllers and a global performance metric. To our knowledge, our attempt to distribute the learning of a collaborative task on a large parameter space is the first of such kind, and will be described in detail in Chapter 9.

3 Materials and Methods

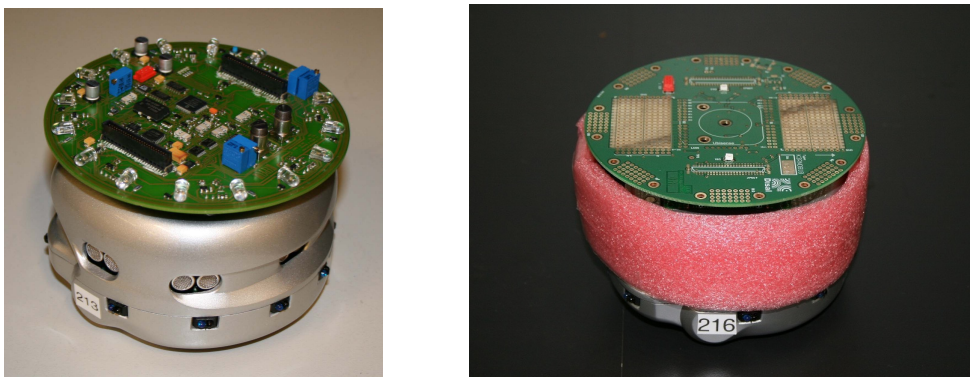
This chapter describes the tools and methods that are common to all experiments conducted in this thesis. We will begin with the hardware and software tools used, and then proceed to the algorithms and benchmark tasks on which they will be tested. When presenting results in subsequent chapters, we will detail the parameters and configurations that are specific to each experiment.

3.1 Khepera III Robot

The experimental platform used in this work is the Khepera III mobile robot, a differential wheeled vehicle with a diameter of 12 cm. The Khepera III is equipped with nine infra-red sensors as well as five ultrasound sensors for short and medium range obstacle detection. The position of the wheels can be measured using two encoders.

Communication between robots is done using a IEEE 802.11 wireless interface and the UDP protocol. A relative positioning system mounted on an extension board, depicted in Figure 3.1a, provides range and bearing to nearby robots based on the strength of an infrared signal [52]. The relative positioning system also communicates the ID of the robot, allowing us to estimate the relative heading of neighboring robots by exchanging bearings between a pair of robots through the Wi-Fi interface. Another extension board with two color LEDs is added for tracking purposes. For experiments in the obstacle avoidance task, we also add a soft plastic cover to protect the robots during collisions, which can occur during the early phases of the learning process (Figure 3.1b).

Since the response of the Khepera III infrared proximity sensors is not a linear function of the distance to the obstacles, the proximity values are inverted and normalized using measurements of the real robot sensor's response as a function of distance. This inversion and normalization results in a proximity value of 1 when touching an obstacle, and a value of 0 when the distance to the obstacle is equal to or larger than 10 cm.



(a) (b)

Figure 3.1: (a) A Khepera III robot with a range and bearing module attached. (b) Khepera III robot with protective cover and tracking board.

3.2 Arena and Tracking Setup

Experiments with real robots take place in an arena with rectangular walls, whose size can be adjusted from 70 cm to 3 m per side. The arena is equipped with an overhead camera connected to a computer running SwisTrack [53], an open source tracking software that enables us to log the position and orientation of robots in the arena. The setup is shown in Figure 3.2a.

3.3 Webots Simulator

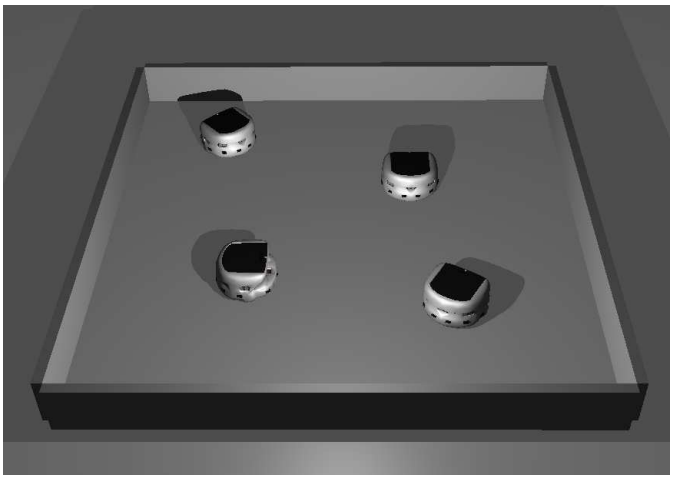
Simulations are performed in Webots [54], a high-fidelity physics-based simulator that models dynamical effects such as friction and inertia. Webots can also be considered a submicroscopic simulator as it models intra-robot modules individually (e.g., sensors, actuators, and body parts), which results in a higher level of detail than standard microscopic models that consider robots as a whole entity. In particular, the simulator has a built-in relative positioning system that gives information about the distance and direction to neighboring robots within line-of-sight, mimicking the relative positioning extension board used in the real robots. Figure 3.2b shows a screenshot of the simulator.

Large scale simulations are performed on a computational cluster of 28 nodes. All nodes have multicore Intel Xeon CPUs (4 nodes with 12 cores and 24 nodes with 8 cores, resulting in 240 cores in total), and run Ubuntu 14.04 as operating system.

3.4 Numerical Benchmark Functions

In his PhD thesis, De Jong introduced a set of functions that became a standard in the analysis of the performance of optimization algorithms [55]. Each function was designed to represent a

(a)



(b)

Figure 3.2: (a) Experimental setup with overhead camera and tracking software. (b) Simulation of 4 Khepera III robots navigating in a square arena.

specific difficulty in optimization problems. For instance, the sphere, defined in Equation 3.1 and shown in Figure 3.3a, is smooth and unimodal, making it easy to find its minimum. On the other hand, in Rosenbrock's function (Equation 3.2, Figure 3.3b), the minimum is located in a narrow valley which makes gradient-based approaches zigzag and take more time to converge. Other more complex test functions that became standard are the Rastrigin function (Equation 3.3, Figure 3.3c) and the Griewank function (Equation 3.4, Figure 3.3d), which display several local optima.

$$f_1(\mathbf{x}) = \sum_{i=1}^{D} x_i^2 \tag{3.1}$$

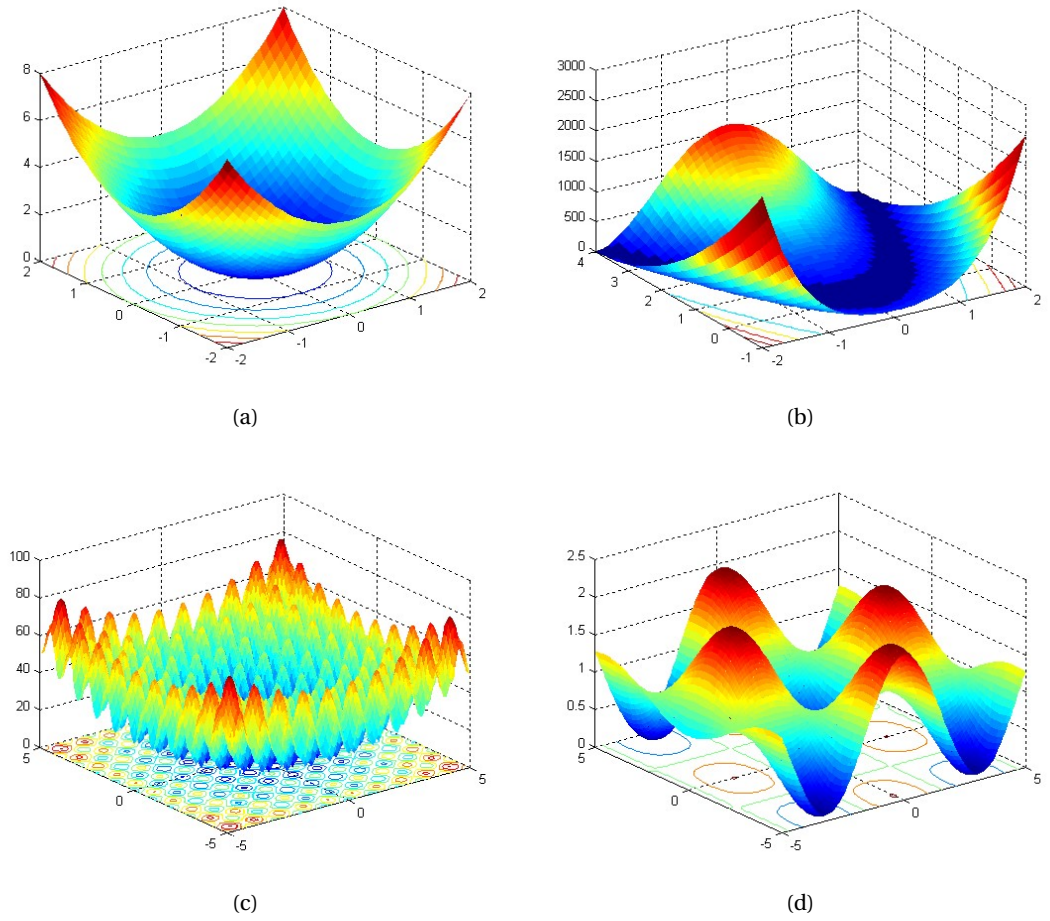
$$f_2(\mathbf{x}) = \sum_{i=1}^{D-1} [(1-x_i^2) + 100(x_{i+1} - x_i^2)^2] \tag{3.2}$$

$$f_3(\mathbf{x}) = 10D + \sum_{i=1}^{D} [x_i^2 - 10\cos(2\pi x_i)] \tag{3.3}$$

$$f_4(\mathbf{x}) = 1 + \frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos(\frac{x_i}{\sqrt{i}}) \tag{3.4}$$

We use these four functions to benchmark our learning algorithms before moving to robotic experiments where the function evaluations become significantly more time consuming.

Within the framework of this research, we published an open source visualization tool that



(a) (b)

(c) (d)

Figure 3.3: Benchmark functions on two dimensions with contour plots shown on the x-y plane. (a) Sphere. (b) Rosenbrock. (c) Rastrigin. (d) Griewank.

allows users to visualize the result of PSO on canonical benchmark functions such as the ones mentioned above. The software, named *Swarmviz*, was developed mainly by Guillaume Jornod, with Iñaki Navarro's and my co-supervision. *Swarmviz* is described thoroughly in an article published in the IEEE Congress on Evolutionary Computation [56] and made freely available on https://github.com/epfl-disal/SwarmViz.git.

3.5 Robotic Benchmarks

In an analogous fashion to De Jong's benchmark functions [55], we intend to use a suite of robotic optimization problems to enable quantitative performance comparisons of the algorithms proposed. While there are several well established benchmarks in numerical optimization (for instance, see [1]), to our knowledge there is no agreed set of robotic benchmark tasks.

In this thesis we will study two tasks of different complexity. The first task is multi-robot obstacle avoidance. Obstacle avoidance is a basic behavior in robotics that was used in one of the earliest works of evaluative adaptation with Genetic Algorithms applied to real robots [2]. This task has also been employed to test other learning algorithms such as Particle Swarm Optimization [6] and Reinforcement Learning [57].

We chose obstacle avoidance as a benchmark task because, in addition to its popularity in the literature, it can be implemented with different number of robots in order to analyze the differences between single and multi-robot systems, requires basic sensors and actuators that are available in most mobile robots, and the chosen performance metric can be fully evaluated with on-board resources. Thus, it can serve as a benchmark for testing distributed learning algorithms with real robots in the same way standard benchmark functions are used in numerical optimization.

The second robotic benchmark task that we will analyze in this thesis is coordinated motion, in which the goal is to have several robots moving together as a group. This task can be seen as an example of loose formation or flocking [58]–[64], as the robots are not required to adopt any particular configuration

Compared to obstacle avoidance, this task has the added complexity that it requires coordination between robots. For the obstacle avoidance task the process of distributing the PSO algorithm is rather straightforward, since each robot evaluates its own performance locally, and while the presence of other robots in the arena negatively affects the performance of the controller being optimized, the small decrease in performance may be considered worthwhile to achieve a learning speed-up factor proportional to the number of robots used [6]. However, for the coordinated motion task the fitness evaluation for each single robot will now be highly dependent on the behaviors of the others. Also, the coordinated motion task requires additional sensing to distinguish between robots and other obstacles.

In this section we will describe the common methods used when approaching these two tasks. Specific details for obstacle avoidance and coordinated motion will be given in Chapters 4 and 9 respectively.

3.5.1 Performance Metrics

A performance metric for a given task gives a quantitative measure of the goodness of a candidate solution. The goal of the learning process is to find a controller that maximizes this metric for a particular task.

Performance metrics are usually designed by combining factors for more specific, simpler behaviors [65]. It is common for factors to conflict with one another, requiring some kind of trade-off. For instance, in obstacle avoidance, turning to avoid obstacles conflicts with maximizing speed, and in flocking, a higher compactness of the group of robots results in more collisions between flockmates.

The most general approach to deal with this conflict is to use a multi-objective optimization algorithm [66] to obtain the full Pareto frontier of solutions. However, this is very computationally expensive, and requires additional time to explore the set of Pareto-efficient solutions to select the most appropriate one.

Scott and Antonsson give an alternative approach to aggregate these factors into a single performance metric based on quasi-linear weighted means [67]. The weight parameters on these aggregation functions can be adjusted to fine tune the behaviors, and if the full Pareto frontier is needed, it can be obtained by systematically exploring the entire range of aggregation weights [68].

In this work, for the obstacle avoidance task in Part II we will use a pre-existing metric from the literature for comparison purposes. On the other hand, for the flocking task we will design metrics to learn this task in a distributed manner by employing the generalized aggregation function approach. The design of this metrics is part of the contributions of this thesis and will be described in detail in Chapter 9.

3.5.2 Control Architecture

In order to perform the learning, we need to parametrize the function that determines the outputs (actuator values) as a function of the inputs (sensor readings) and possibly some internal state (memory). We have selected Artificial Neural Networks as the control architecture because they can model functions of arbitrary number of inputs and outputs, and they have been shown to be able uniformly approximate arbitrary functions [69], [70].

An artificial neural network is a set of connected artificial neurons or units. Each unit calculates its output y by appling a nonlinear function $s(\cdot)$ to the sum of inputs x_i weighted by parameters w_i. Usually, a constant bias is added as an additional input, so that if there are k input variables, the total number of weights is $k+1$. This relation is given in Equation 3.5 and schematized in Figure 3.4.

$$y = s(w_0 + \sum_{i=1}^{k} x_i \cdot w_i) \tag{3.5}$$

The activation function, which is the non-linear function $s(\cdot)$ applied after summing all the inputs, is the sigmoid defined by Equation 3.6.

$$s(x) = \frac{1}{1+e^{-x}} \tag{3.6}$$

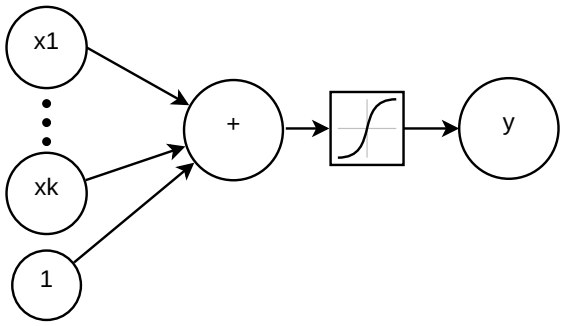


Figure 3.4: Schematic of a single artificial neuron.

The number of inputs, outputs, and weight parameters of the artificial neural network will vary for the different tasks. The baseline architecture for the obstacle avoidance task will be given in Chapter 4, and several variations of it will be explored in Chapter 7. The specific architecture for the coordinated motion task will be given in Chapter 9.

3.5.3 Optimization Problem

Given a general controller parametrized by weights $\{w_i\}$ (e.g., an Artificial Neural Network as described in the previous subsection), the optimization problem to be solved by the learning algorithm is to choose the set of weights $\{w_i\}$ such that the performance metric for the task at hand is maximized.

It should be noted that there is no explicit mathematical expression of the performance metric as a function of the weight parameters of the controller. This mapping depends on the direct interactions between the robot and the environment, which are not known in advance, and could result in a more or less nonlinear, discontinuous, and noisy landscape. This precludes the use of gradient-based approaches and justifies the use of a black-box, gradient-free optimization metaheuristic such as PSO.

In addition, the control optimization tasks analyzed in this thesis present up to five distinct sources of uncertainties in the performance evaluations: the proximity sensors' noise, the robots' wheel slip, the initial robots' pose for each evaluation, the position of obstacles if any, and the behavior of other robots in the same arena. This last factor, the behavior of other robots, can be considered a source of uncertainty because there is no communication used to explicitly coordinate the motion of the robots. In the flocking task in particular, communication is used to estimate relative headings to neighbors, i.e., it is part of the robots' sensing and not an explicit coordination mechanism.

```
1: Initialize particles
2: for N_i iterations do
3:    for N_p particles do
4:       Evaluate new particle position
5:       Share personal best within neighborhood
6:       Update particle position (Eqs. 3.7 and 3.8)
7:    end for
8: end for
```

Figure 3.5: Pseudocode for the standard PSO algorithm.

3.6 PSO Algorithms

Two optimization algorithms are used a starting point in the analysis on benchmark functions and robotic tasks. The first one is a standard PSO version based on the work of Kennedy et al. [4], while the second one is a noise-resistant PSO variation introduced by Pugh et al. [7]. Both algorithms have low requirements in terms of computation and communication and can be easily executed completely on-board the real robots.

3.6.1 Standard PSO

The pseudocode for standard PSO is shown in Fig. 3.5, where N_i is the number of iterations for PSO and N_p is the total number of particles.

Each particle i has a position x_i in the search space which is a vector that represents a candidate solution. For benchmark functions, a candidate solution simply means a position in the Euclidean space, while for robotic experiments, it means a set of parameter values for a candidate controller.

Particles' positions and velocities are initialized randomly: each component of the position and the velocity is assigned a random number drawn from a uniform distribution in a closed interval $[-X_{init}, X_{init}]$ (the size of the interval is determined by the parameter X_{init}). Next, the whole population of candidate solutions, which in PSO is often referred to as swarm, is evaluated.

Particle evaluations in benchmark functions are simply done by calculating the function value according to the corresponding equation. On robotic experiments, on the other hand, each particle evaluation consists of a robot moving in the arena for a fixed time t_e running the controller with the parameters given by that particle's position. The fitness corresponding to the particle is equivalent to the performance of the robot measured with the metric selected for that task. When the algorithm is implemented in a distributed fashion, each robot evaluates in parallel a different candidate solution corresponding to a particle's position.

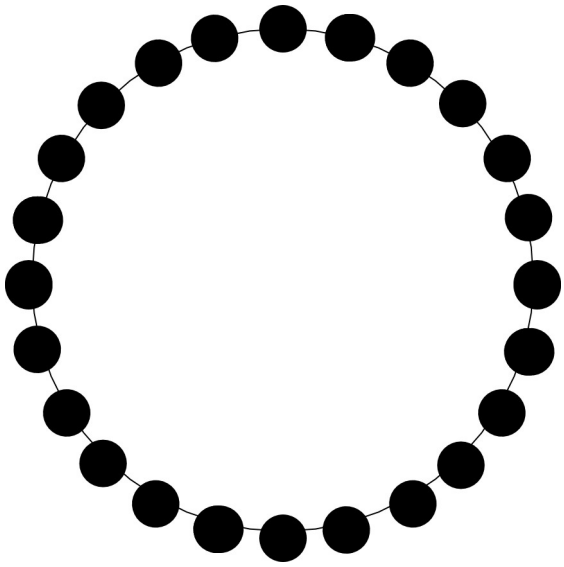


Figure 3.6: Ring neighborhood topology used for PSO.

The performance of the current position is compared to that of the personal best, which is the position with the best performance ever encountered by a specific particle. If the performance of the current position is better, the personal best is updated to be the current position. Then, the personal best is shared within the particle's neighborhood, and the neighborhood best, which is the position with the best performance ever encountered in that neighborhood, is updated with the information received from neighbors.

The neighborhood used for the particles' communication in this thesis has a ring topology with one neighbor on each side, shown in Figure 3.6. This neighborhood space depends only on the particles' indexes, and is different from the search space where the particles move and also from the physical space in which the robots move. In Chapter 12 we will discuss the effect of neighborhood size on distributed PSO implementations for this topology. Other neighborhood topologies were analyzed in [6], [71] and are therefore not studied in this work.

At each iteration, the position of particle i in dimension j, denoted by $x_{i,j}$, is updated by adding a velocity $v_{i,j}$ according to Equation 3.7.

$$x_{i,j} := x_{i,j} + v_{i,j} \tag{3.7}$$

$$v_{i,j} := w_I \cdot v_{i,j} + w_p \cdot rand() \cdot (x^*_{i,j} - x_{i,j}) + w_n \cdot rand() \cdot (x^*_{i',j} - x_{i,j}) \tag{3.8}$$

This velocity depends on three components: the velocity at the previous iteration weighted by an inertia coefficient w_I, a randomized attraction to its personal best $x^*_{i,j}$ weighted by w_p,

```
1: Initialize particles
2: for N_i iterations do
3:    for N_p particles do
4:       Evaluate particle new particle position
5:       Re-evaluate personal best
6:       Aggregate personal best with previous best
7:       Share personal best within neighborhood
8:       Update particle position (Eqs. 3.7 and 3.8)
9:    end for
10: end for
```

Figure 3.7: Pseudocode for the *PSO pbest* algorithm.

and a randomized attraction to the neighborhood's best $x^*_{i',j}$ weighted by w_n (Equation 3.8). $rand()$ is a random number drawn from a uniform distribution between 0 and 1. If the new velocity falls outside the interval $[-X_{init}, X_{init}]$, then its value is limited to the closest bound on that interval. Note that while the particles' velocity is bounded, their positions are not, meaning that they can potentially reach any point in the search space.

3.6.2 PSO pbest

The noise-resistant variation, which we will refer to as *PSO pbest*, operates by re-evaluating personal best positions and calculating an average of all personal best evaluations using the stored previous average and the number of evaluations. This re-evaluation results in a more accurate estimate of the performance metric in the presence of noise, but requires twice as many evaluations at each iteration than standard PSO.

The pseudocode for *PSO pbest* is shown in Fig. 3.7. The difference with the standard PSO pseudocode is the addition of Lines 5 and 6.

Part II
Learning Obstacle Avoidance in Multi-Robot Systems

4 Methods for Obstacle Avoidance

This chapter details the specifics of the methods described in Chapter 3 when applied to the multi-robot obstacle avoidance task. It is divided into four sections, concerning the performance metric, the obstacles used, the control architecture, and the statement of the optimization problem to be solved by PSO. A final fifth section provides a short chapter summary.

4.1 Performance Metric

The goal in the obstacle avoidance task as defined in this thesis is for robots to navigate autonomously in a bounded arena avoiding collisions with the arena walls, other robots, and other static obstacles placed in the arena. Since we are using the obstacle avoidance task as a benchmark, we selected an existing metric of performance introduced by Floreano and Mondada in [2], which was also used in [6], [72].

The total fitness, defined in Equation 4.1, is the product of three factors, which reward robots that move quickly (high speed factor f_v), turn as little as possible (low turn factor f_t), and stay away from obstacles (low proximity factor f_i).

$$f = f_v \cdot (1 - \sqrt{f_t}) \cdot (1 - f_i) \tag{4.1}$$

The calculation of each factor is detailed Equations 4.2 to 4.4:

$$f_v = \frac{1}{N_e} \sum_{k=1}^{N_e} \frac{|v_{l,k} + v_{r,k}|}{2} \tag{4.2}$$

$$f_t = \frac{1}{N_e} \sum_{k=1}^{N_e} \frac{|v_{l,k} - v_{r,k}|}{2} \tag{4.3}$$

$$f_i = \frac{1}{N_e} \sum_{k=1}^{N_e} i_{max,k} \tag{4.4}$$

where $\{v_{l,k}, v_{r,k}\}$ are the normalized speeds of the left and right wheels at time step k, $i_{max,k}$ is the normalized proximity sensor activation value of the most active sensor at time step k, and N_e is the number of time steps in the evaluation period (i.e., the evaluation time t_e divided by the time step of the controller, which is in our case 32 ms).

All factors are normalized to the interval [0, 1]. Each factor is calculated at each time step and then the product is averaged for the total number of time steps in the evaluation period. The averaging effect reduces the impact of the initial conditions, but it also filters abrupt variations, unless they last appreciably in time, e.g., the robot getting stuck is much more heavily penalized than a single instantaneous collision.

This fitness would be equal to one only if the robot was moving in a straight line at full speed with no obstacles within its sensing range. In order to avoid obstacles, the robot must first detect them with its proximity sensors, lowering its proximity factor ($f_i < 1$), and then change its velocity, lowering either the speed (f_v) or turn (f_t) factor. These two aspects result in a total fitness that is lower than one ($f < 1$). Therefore, the maximum possible fitness in a given environment depends on the obstacle density and configuration, with more cluttered environments having lower maximum fitness.

4.2 Obstacles

We use cylindrical obstacles of different sizes, as well as rectangular walls that can be rearranged to form corridors or empty squares. Other robots in the same arena also constitute obstacles, therefore the difficulty of the task increases with the density of robots in the arena. The type, shape, and size of obstacles for each experiment will be detailed in subsequent chapters. In particular, Chapter 6 will focus specifically on the role that the obstacle configuration has on the learned behaviors.

Between evaluations, the obstacles' positions and the robots' poses (positions and orientations) are randomized in order to reduce the influence of the previous evaluation on the current one. In simulation, both obstacles' positions and robots' poses are set randomly with a uniform probability distribution before each fitness evaluation, verifying that each object does not overlap with walls or other objects. In real robot experiments, the obstacles are randomly positioned the first time, and then kept in this fixed position for the rest of the experiments,

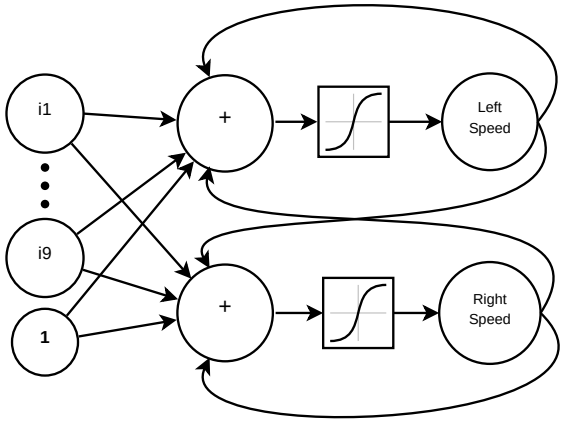


Figure 4.1: Artificial Neural Network used as controller.

while the robots are manually repositioned in random poses between evaluations.

4.3 Control Architecture

The control architecture used in most experiments on obstacle avoidance is a recurrent artificial neural network of two units, shown in Figure 4.1. By recurrent we mean that the outputs of the network from the previous time step are stored in memory and used as inputs for the next time step. We will refer to this controller as *ann24*, since it has 24 parameters.

Each neuron has 12 input connections: the 9 normalized infrared sensors values $\{i_1, \cdots, i_9\}$, a connection to a constant bias speed, a recurrent connection from its own output, and a lateral connection from the other neuron's output, resulting in the 24 following weight parameters: $\{w_0, \cdots, w_{23}\}$. The outputs of the units determine the speeds of the left and right wheels $\{v_{l,t}, v_{r,t}\}$, as shown in Equation 4.5.

$$\begin{aligned} v_{l,t} &= s(w_0 + \sum_{k=1}^{9} i_k \cdot w_k + w_{10} \cdot v_{l,t-1} + w_{11} \cdot v_{r,t-1}) \\ v_{r,t} &= s(w_{12} + \sum_{k=1}^{9} i_k \cdot w_{k+12} + w_{22} \cdot v_{l,t-1} + w_{23} \cdot v_{r,t-1}) \end{aligned} \tag{4.5}$$

The activation function, which is the non-linear function $s(\cdot)$ applied after summing all the inputs, is the sigmoid defined in Equation 3.6.

Chapter 7 will explore in detail variations of this controller for the same task and the effect

that they have on the resulting performance.

4.4 Optimization Problem

Given the Artificial Neural Network Controller *ann24* described in the previous section and parametrized by 24 weights $\{w_i\}$, the optimization problem to be solved by the learning algorithm is to choose the set of weights $\{w_i\}$ such that the fitness function f as defined in Equation 4.1 is maximized.

4.5 Chapter Summary

This chapter describes the methods that will be used in experiments on obstacle avoidance in the subsequent chapters of Part II.

The performance metric is the product of three factors, which reward robots that move quickly, turn as little as possible, and stay away from obstacles. It can be calculated locally by each robot using on-board resources.

Experiments take place in a rectangular arena with obstacles of different shapes and sizes.

The control architecture is a recurrent artificial neural network with 24 weight parameters. It determines the robot's wheel speed as a function of the proximity sensors' readings.

The optimization problem for PSO is to select the weights of the neural network that maximize the performance metric.

5 PSO Parameters

In this chapter, we will analyze how different algorithmic parameters in a distributed PSO implementation affect the total evaluation time and resulting fitness. Our goal is to reduce the total evaluation time so that it is feasible to implement the adaptation process within the limits of the robots' battery life.

A motivating example of the need to reduce evaluation time is the work of Floreano and Mondada [2], where a tethered mobile robot learned to navigate a path and avoid obstacles using Genetic Algorithms. This experiment required 67 hours of total evaluation time, and it would require the same time to recreate it nowadays since the limit was not imposed by computational capabilities but rather by the time needed to gather enough information on the quality of the candidate solutions.

The selection of PSO parameter values to reduce evaluation time is an instance of the more general problem of configuring a metaheuristic [73]. This problem has been addressed through a number of meta-learning approaches that evaluate several candidate configurations in order to choose the best one [73], [74]. For instance, Birattari et al. proposed a racing algorithm that empirically evaluates a finite set of candidate configurations and discards poor-performing ones based on the Friedman statistical test [73]. Gagliolo and Schmidhuber represent algorithm selection as a bandit problem and propose an online method to allocate computation time to a given set of algorithms of unknown performance [74].

However, these meta-learning approaches are not appropriate for our particular problem because of three reasons. In the first place, it is hard to generate a set of candidate configurations of a relatively small size, especially due to the fact that most PSO parameters are real numbers that can take a wide range of values. In the second place, these meta-learning algorithms do not help in gaining a deeper understanding of the role of the different algorithmic parameters, which is crucial if the goal is to derive guidelines and extend the results to different robotic tasks. Finally, it seems it would be quite challenging to implement these algorithms on-board on a distributed system with limited resources such as our robotic platform.

Therefore, our approach will be to first explore in simulation those combinations of PSO parameters which allow for a reduction in the evaluation time. Based on the result of such systematic experiments, we will propose a set of empirical guidelines for choosing these parameter values, and then we will apply them to a real robot distributed implementation of PSO.

We will organize the description of the experiments and the presentation of results in the following manner. Firstly, in Section 5.1, we will introduce the four parameters that affect the total evaluation time in PSO. Secondly, in Section 5.2, we will give a baseline set of values for these parameters that represent full-time adaptation and that will be used as a qualitative and quantitative reference for when we later reduce the evaluation time. In Section 5.3, we will describe the qualitative behavior of the best solution strategies obtained in simulation for the baseline set of parameters. Section 5.4 will analyze in simulation the impact of reducing each of the four previously mentioned algorithmic parameters and propose guidelines for setting their values. In Section 5.5 we will apply the proposed guidelines to reduce the total evaluation time in simulation and compare the results with the full-time adaptation.

Then we will move from experiments in simulation to experiments with real robots. Section 5.6 will discuss how controllers optimized in simulation perform when transferred to real robots, focusing on the different noise distributions in fitness evaluations in both simulation and reality. In Section 5.7 we will implement the proposed guidelines with real robots and compare the optimization performed exclusively on real robots with a hybrid simulate-and-transfer approach which reduces the required real robot experimental time. Finally, Section 5.8 will conclude the chapter with a summary of our findings.

5.1 Total Evaluation Time for PSO

For standard PSO, the total evaluation time depends on three parameters: the population size (N_p), the individual candidate evaluation time (t_e), and the number of iterations of the algorithm (N_i). In the case of *PSO pbest*, the noise-resistant variation by Pugh et al. [7], a fourth parameter needs to be considered: the number of re-evaluations of the personal best position associated with each candidate solution within the same iteration (N_{re}). By setting $N_{re} = 0$ we obtain standard PSO, while $N_{re} = 1$ results in *PSO pbest*.

The total evaluation time as a function of these four parameters is shown in Equation 5.1. If we assume that there is a fixed upper limit for the total evaluation time, an increase in any of the parameters would result in a proportional decrease in the rest.

$$t_{tot} = t_e \cdot N_p \cdot N_i \cdot (N_{re} + 1) \tag{5.1}$$

In a parallelized or distributed implementation, fitness evaluations are distributed among

N_{rob} robots, and the wall-clock time t_{wc} required to evaluate candidate solutions is reduced (Equation 5.2).

$$t_{wc} = t_e \cdot \left\lceil \frac{N_p}{N_{rob}} \right\rceil \cdot N_i \cdot (N_{re} + 1) \tag{5.2}$$

It is worth noting that the number of robots is not necessarily the same as the population size. In fact, the choice of the population size depends on the dimension of the search space and the complexity of the task, while the choice of number of robots is based on more experimental considerations (e.g., availability of robots, targeted number of robots needed for a specific mission in a given environment). Thus, a robot may have several particles to evaluate within the same candidate solution pool as opposed to only one.

A full optimization of the algorithmic parameters to minimize the total evaluation time would be a very computationally expensive problem, due to the large number of candidate configurations, the combination of continuous and discrete parameters, the large variation between runs, and the possible existence of local optima. Thus, our approach is to analyze each parameter individually, taking into account its impact on the final performance as compared to a full-time adaptation baseline.

5.2 Baseline Parameter Values

Ideally, the full-time baseline would be selected so that any further increase in the evaluation time does not result in a better final performance. Our baseline set of parameters is based on the work of Pugh et al. [6], and is shown in Table 5.1. This set of parameters amounts to a total evaluation time of approximately 417 hours if carried out on a single robot, what we refer to as full-time adaptation as opposed to the limited-time adaptation obtained after reducing the parameters.

Experiments on the obstacle avoidance task will follow the methodology described in Chapter 4. We will use a square arena of 1x1 m without static obstacles and the number of robots will be varied between 1 and 8. At the end of each optimization run, the best performing solution will be tested with 40 evaluations of 20 s, and the final performance will be the average of these final evaluations.

To complement our robotic case study and add more generality, we will also perform runs on the four numerical benchmark functions without noise described in Section 3.4: the sphere, Rosenbrock's, Rastrigin's, and Griewank's. The baseline parameters for the algorithm when run on benchmark functions are also shown in Table 5.1.

Table 5.1: Algorithmic parameter values

Parameter	Obstacle Avoidance	Benchmark functions
Population size N_p	100	100
Iterations N_i	50	500
Evaluation span t_e	150 s	-
Re-evaluations N_{re}	1	0
Personal weight w_p	2.0	2.0
Neighborhood weight w_n	2.0	2.0
Dimension D	24	24
Inertia w_I	0.8	0.6
Initial range X_{init}	20	5.12

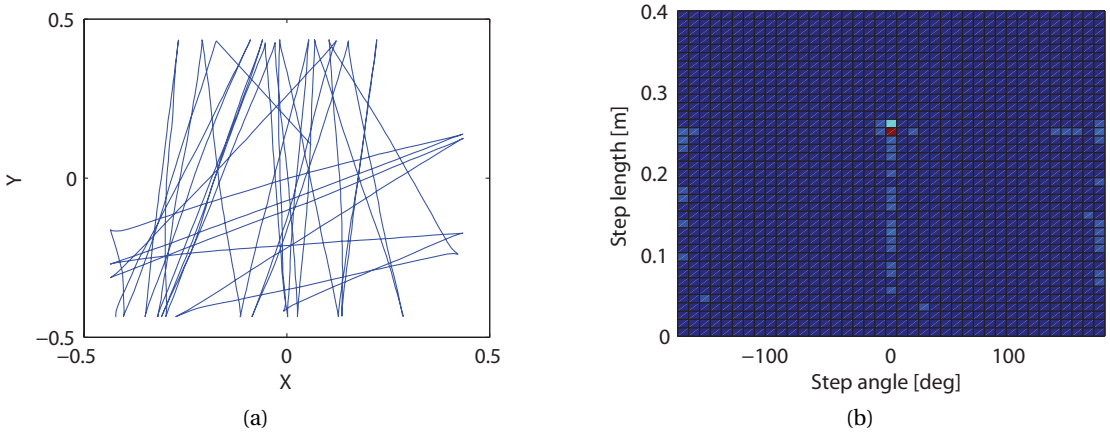


Figure 5.1: (a) A sample trajectory described by the best solution strategy for single-robot learning. (b) Its step length and angle distribution.

5.3 Best Solution Strategies for Obstacle Avoidance

Before looking at the quantitative performance of the robots on the obstacle avoidance task, we would like to analyze the qualitative behaviors that result in a successful avoidance strategy when there is sufficient total evaluation time. In later sections, when the evaluation time is reduced, we will verify if the same qualitative behavior is maintained.

Consider a single robot in an empty arena. A priori, we can think of two different obstacle avoidance strategies: going back and forth between walls, and circling around the center of the arena. Conjecturing behaviors in a multi-robot setting is less straightforward due to the interactions between robots.

In simulation, we analyzed the best solution strategies by examining the trajectories described by the best performing controllers, focusing on the step length and angle distributions as described in [75].

The best solution strategy observed both in the single and multi-robot case consisted in going back and forth between walls in a straight line, reversing the direction of motion every time an obstacle was detected (Figure 5.1a.) This strategy is not surprising when we consider that the fitness function does not penalize going backwards but penalizes turning. Therefore, by exploiting the recurrence of the artificial neural network architecture the robot can perform a quick reversal of direction with little penalty.

Figure 5.1b shows the bidimensional step length and angle distribution of the best trajectory. Most steps consists of the robot moving straight at maximum speed (step length=25cm, step angle=0°). The remaining steps represent the robot decelerating when reversing direction (0<step length<25cm, step angle=0°), and sharp 180° turns.

We looked for circular solutions by searching for high absolute values of the median step angle. We chose this criterion as opposed to the mean step angle as it is less sensitive to the sharp turns that occur when encountering obstacles. A negative median step angle implies clockwise turning, a positive one counter clockwise turning, and a value of zero moving in straight lines.

We wanted to determine if the circular behavior could arise automatically under different environmental conditions. Therefore, we varied the length of the square arena from 0.4 m to 20 m and the number of robots from 1 to 8, but the best solution found for all cases was moving in straight lines. Circular behaviors became the best strategy only when going backwards was explicitly penalized in the fitness function, as we will see in Chapter 6.

5.4 Analysis of PSO Parameters in Simulation

As a first step in the analysis of the algorithmic parameters, we started from the total evaluation time baseline of 417 h and reduced N_p, N_i, and t_e individually to 5, 5, and 5 s respectively, while keeping the other two parameters at their baseline values, plotting the three curves in the same graph for better comparisons (for details on the ranges used for each parameter see subsections 5.4.1 to 5.4.4).

We performed 100 independent runs for each set of parameter values and with 1, 2, 4, and 8 robots. When multiple robots were considered, all of them were learning in parallel. Results are shown in Figure 5.2. The fitness presented in Figure 5.2 and in all fitness figures in Sections 5.4 to 5.5 are the final performances (mean performance of the best solution over the 40 final evaluations) averaged for 100 optimization runs. Error bars represent the standard deviation among the 100 optimization runs.

In all cases, reducing the evaluation span t_e has the least impact on the resulting fitness, followed by N_p and N_i. When comparing the same total evaluation time across different numbers of robots, it can be noted that as the number of robots increases, the arena becomes more crowded and obstacle avoidance becomes harder, thus causing the average fitness to decrease. Also, performances are noisier (see larger error bars in lower right corner) and

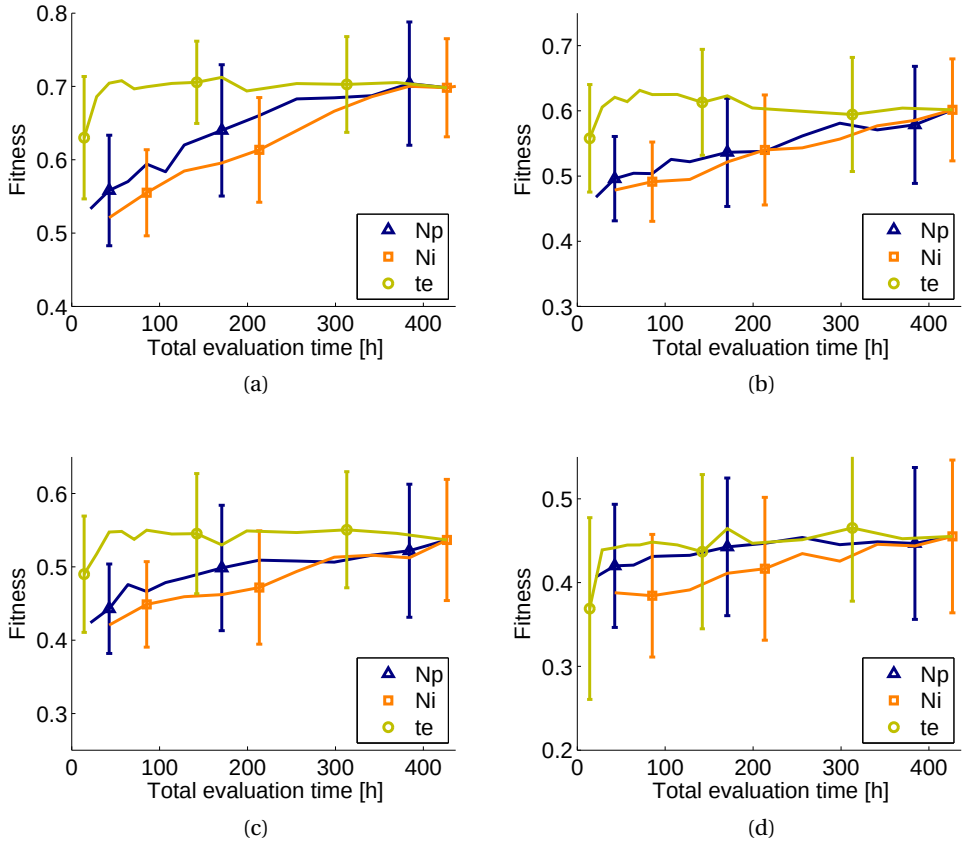


Figure 5.2: Fitness as a function of total evaluation time for 1, 2, 4, and 8 robots corresponding to labels (a) to (d) respectively. Each curve was generated by varying one algorithmic parameters (population size, number of iterations, and evaluation span) with the others held constant.

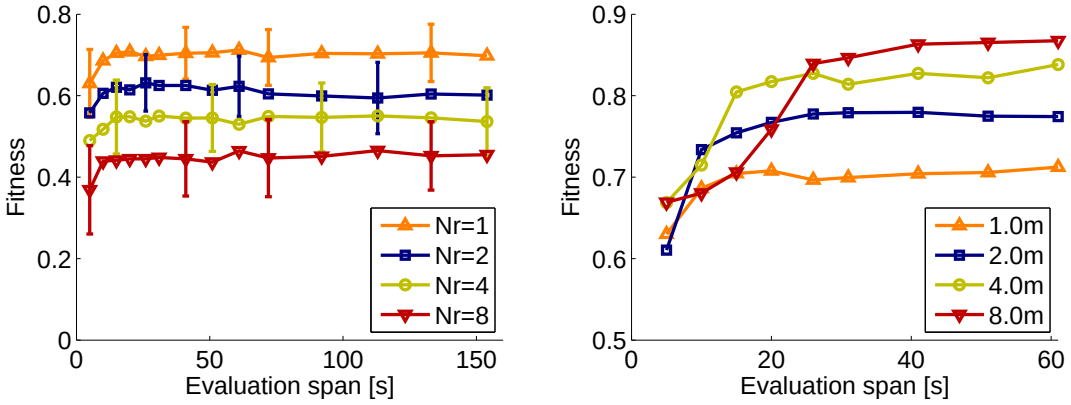


Figure 5.3: Mean fitness as a function of the evaluation span. (a) 1, 2, 4, and 8 robots in a 1x1 m arena. (b) Single robot in arenas of side lengths 1, 2, 4, and 8 meters (y axis zoom in the 0.5 to 0.9 range).

therefore there is less impact of a decreased N_p or N_i (flatter profile than with 1-2 robots).

The following subsections present a more detailed analysis of each individual parameter.

5.4.1 Evaluation Span

To analyze the effect of the evaluation span, we reduced t_e from 150 s to 5 s, with 20 s steps in the higher range and 5 s steps in the low range. We did 100 runs for each value, and plotted the mean and standard deviation in Figure 5.3a.

The mean fitness remains fairly constant for 30 s $< t_e <$ 150 s for all number of robots. In fact, the difference in fitness between 150 s and 30 s is not statistically significant in all cases (Mann-Whitney U test, 5% significance level). For $t_e <$ 30 s the fitness starts to decrease, although at different rates for different numbers of robots. In particular, for 8 robots t_e can be reduced to 10 s without a major change in fitness, which suggests that a crowded arena may allow for shorter evaluation spans due to more frequent collisions, and thus more opportunities to learn to avoid them. It is interesting to note that reducing the evaluation span does not seem to increase the fitness variation between runs.

The evaluation span parameter depends on the task and the environment. With the goal of trying to explain the lower limits for our task, we varied the evaluation span for several arena sizes using one robot (Figure 5.3b). In this case, the point where performance starts to drop occurs at longer evaluation spans for larger arena sizes (15 and 25 seconds for 4 and 8 meters respectively). We suspect this point is related to the minimum time it takes to have at least one collision. In fact, if the robot moves in a straight line at a maximum speed of 0.25 m/s, it takes 16 and 32 seconds to cross one side of an arena of 4 and 8 meters respectively.

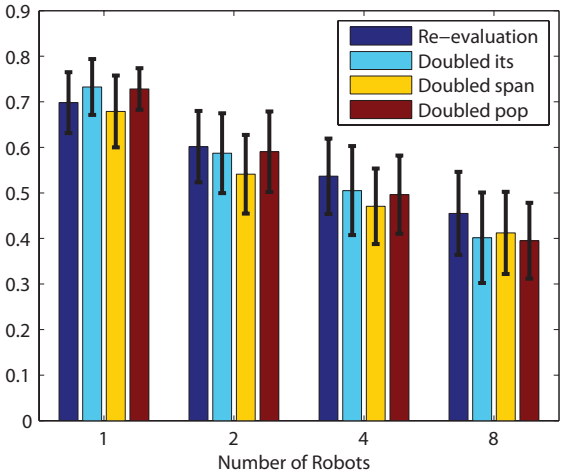


Figure 5.4: Average fitness and standard deviation for *PSO pbest*, standard PSO with doubled number of iterations, standard PSO with doubled evaluation span, and standard PSO with doubled population size.

Therefore, the robot speed and environment size can be used as guidelines to choose an evaluation span. We suggest that, in general, the minimum evaluation span should guarantee at least one interaction of the robot with other components of the environment relevant to the task at hand.

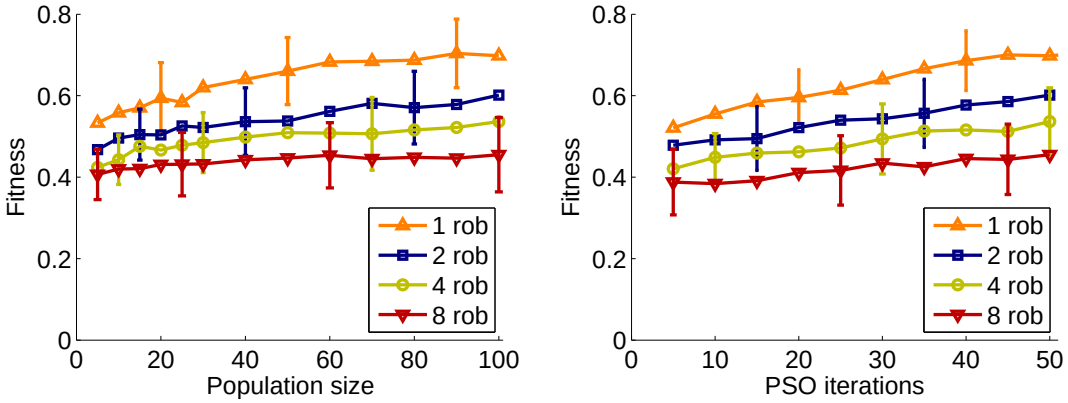
5.4.2 Re-evaluations

Next, we compared the performance of *PSO pbest* (number of re-evaluations $N_{re} = 1$) with standard PSO (number of re-evaluations $N_{re} = 0$) to determine if re-evaluations improve performance in limited time scenarios.

For any given set of parameters, *PSO pbest* takes twice as much evaluation time as standard PSO due to the personal best re-evaluations. In order to perform a fair comparison, if we remove re-evaluations we need to double one of the other parameters to keep total evaluation time constant. We thus compared four alternatives: re-evaluations, doubled iterations, doubled evaluation span, and doubled population size.

We performed 100 runs for each algorithmic variant and plotted the final fitness in Figure 5.4.

In the single robot case, *PSO pbest* performs significantly worse than standard PSO with doubled iterations and doubled population size (Mann-Whitney U test, 5% significance level). However, as the number of robots is increased, the relative performance of *PSO pbest* improves: for 2 robots there is no significant difference, and for 4 and 8 robots *PSO pbest* significantly outperforms standard PSO with doubled iterations and doubled population size. It is worth noting that there is no significant difference between doubling population size and number of iterations for all number of robots, and that doubling the evaluation span performs



(a) (b)

Figure 5.5: (a) Mean fitness for different population size values. (b) Mean fitness for different number of iterations.

significantly worse in all cases except for 8 robots.

These results suggest that the decision to invest time in re-evaluations depends on the amount of uncertainty in fitness evaluations. In fact, as the arena becomes more crowded, there is more uncertainty in fitness evaluations, which depend both on the performance of other robots (avoiding other robots is easier if other robots are also trying to avoid you) and on initial conditions such as the position in the arena (a robot is more likely to be trapped against a corner in a crowded arena).

Re-evaluations may also reduce the effect of heterogeneities on solution sharing in multi-robot evaluations, as shared solutions are re-evaluated at each iteration and thus can be dropped if they do not perform as well as they had done on other robots. The final advantage of re-evaluations can be seen in the case of dynamic environments, where a previously found good solution may no longer be valid after a certain amount of time. Thus, re-evaluations seem to be a good recommendation in general for multi-robot learning scenarios.

5.4.3 Population Size

Both for our robotic case study and the benchmark functions, the population size was reduced from 100 to 30 in steps of 10, and from 30 to 5 in steps of 5, in order to obtain more data points in what we expected to be an interesting region, while keeping all the other parameters the same as in the baseline.

We did 100 independent runs for each value; results are shown in Figure 5.5a for obstacle avoidance and Figure 5.6 for benchmark functions.

It is clear from Figures 5.2 and 5.6 that, at least with our baseline parameters, reducing the

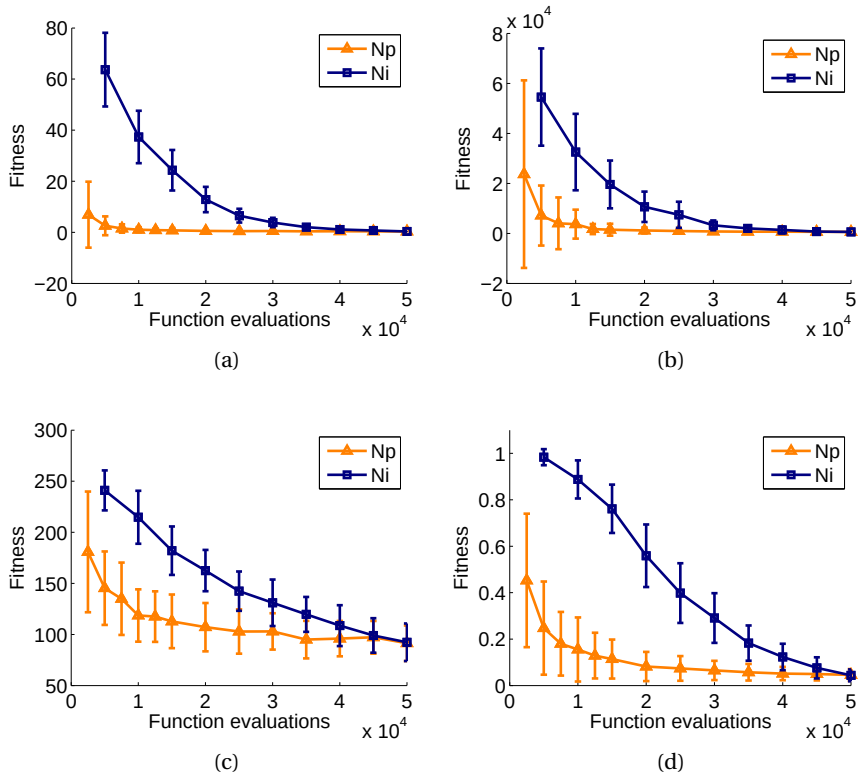


Figure 5.6: Fitness as a function of number of evaluations for benchmark functions f1 (a), f2 (b), f3 (c), and f4 (d). Lower fitness is better. Each curve was generated by varying one algorithmic parameter (population size and number of iterations) with the other held constant.

population size is better in terms of mean fitness than reducing the number of iterations, both for obstacle avoidance and for all benchmark functions (note that in benchmark functions lower fitness values mean better performance). Now the question that arises is how low should we set the population size? While there is no clear consensus in PSO literature[76], there are a few guidelines based on the dimension D of the search space such as $N_p = D$ or $N_p = 10+2\sqrt{D}$ (this last formula is used in Standard PSO 2006, an effort to define a PSO standard published online in Particle Swarm Central[1]).

Another approach is to start with a fixed value such as $N_p = 40$ and restart the algorithm with a larger N_p if early convergence is noticed. However, it is hard to determine if a restart is needed, especially when the maximum feasible fitness is not clear beforehand, which is often the case when learning robotic behaviors.

In Figure 5.5a, we note a slight change in the fitness slope at around $N_p = 25$, which is very close to the value of the dimension of the search space $D = 24$. This effect is much more clear in the case of the benchmark functions f_2, f_3, and f_4 (in Figure 5.6, $N_p = 25$ corresponds to 12500 function evaluations on the x axis, based on the default parameters mentioned in Table 5.1).

Also, when the population size becomes small, more outliers appear due to runs that fail to converge to a satisfactory solution. This can be noted in the higher standard deviation seen in reduced N_p as compared to reduced N_i with the same total evaluation time (see Figure 5.2).

Thus, because of higher fitness, lower variance, the possibility to distribute particles among robots, and the impracticality of the restart process, we prefer to err on the side of larger population sizes, and we suggest the following guideline:

$$N_p = \max(D, N_{rob}) \tag{5.3}$$

According to Equation 5.3, in the case where the number of robots is greater than the dimension, the population size will be set equal to the number of robots in order to take advantage of parallel evaluations and increased robustness.

5.4.4 PSO Iterations

For our robotic case study, the number of iterations was reduced from 50 to 5 in steps of 5, again keeping all the other parameters the same as in the baseline. We did 100 independent runs for each parameter value, results are shown in Figure 5.5b. The chosen benchmark functions traditionally use larger values of N_i, so we chose 500 as a baseline and reduced it to 50 in steps of 50 (see Figure 5.6).

[1] `http://www.particleswarm.info`

Table 5.2: Limited-time adaptation parameter values

Parameter	Value
Population size N_p	24
Iterations N_i	30
Evaluation span t_e	20 s
Re-evaluations N_{re}	1

We observed a nearly linear performance drop for obstacle avoidance and on benchmark functions f_3 and f_4. For f_1 and f_2, the behavior of N_i was similar to that of N_p, but with a worse fitness overall.

Given that N_i is the easiest parameter to adjust on the fly, this parameter seems suitable for trade-offs between performance and available learning time. That is, for a fixed available time t_{wc}, we suggest using the previous guidelines to determine the 3 other parameters and allocate all remaining time to N_i using Equation 5.4, which is obtained by isolating N_i from Equation 5.2.

$$N_i = \frac{t_{wc}}{t_e \cdot \left\lceil \frac{N_p}{N_{rob}} \right\rceil \cdot (N_{re}+1)} \tag{5.4}$$

5.5 Limited Time Adaptation in Simulation

Our goal is to perform a full optimization run with 4 robots within 2 hours, the approximate battery autonomy of the Khepera III with tracking board and wireless card.

For this purpose, we followed the guidelines proposed in the previous section to determine the four algorithmic parameters, shown in Table 5.2. We then ran the learning process for this limited-time adaptation in simulation for $N_{rob} = \{1, 2, 4, 8\}$. The final fitness (mean performance of the best solution over the 40 final evaluations) averaged for 100 optimization runs is shown in Figure 5.7.

The mean difference in fitness between full-time and limited time adaptation is 17%, 17%, 14%, and 9% for 1, 2, 4, and 8 robots respectively. These relative performance drops are low when we take into account the fact that the evaluation time was reduced more than 52 times.

More importantly, both limited and full-time adaptation converged to the same obstacle avoidance strategy, regardless of the number of robots: going back and forth between walls in a straight line, reversing the direction of motion every time an obstacle was detected. We verified the sameness of the solution strategies by analyzing the trajectories described by the robots, focusing on the step length and angle distributions as described in Section 5.3.

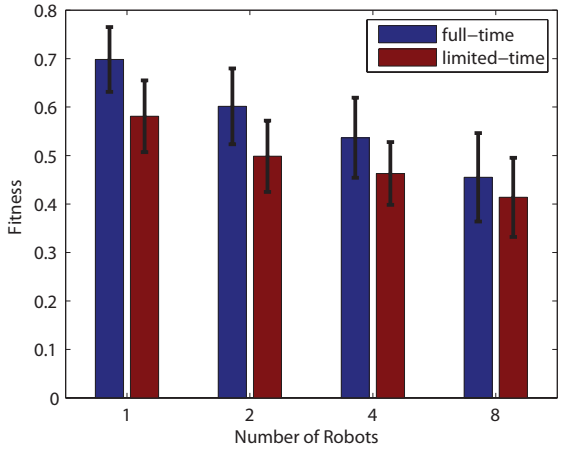


Figure 5.7: Average fitness and standard deviation for full-time and limited-time adaptation.

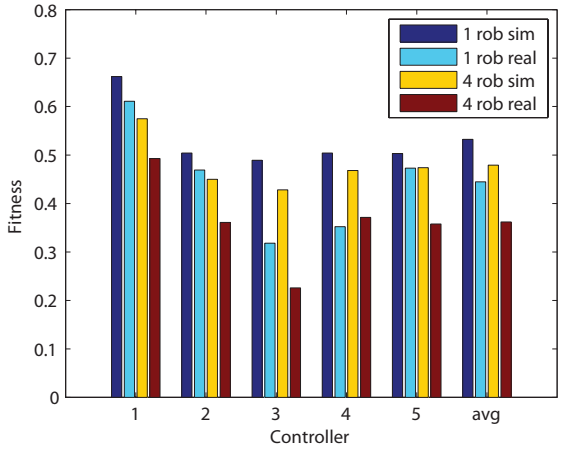


Figure 5.8: Mean fitness for five controllers optimized in simulation and evaluated in simulation and in reality with one and four robots.

5.6 Evaluation Runs with Real Robots

Results presented in previous sections of this chapter were obtained through simulation. Computer simulations may run several times faster than real-time, offering the advantage of reduced evaluation time. However, great care must be taken to ensure that the performance of a candidate solution is eventually maintained in the real implementation. Therefore, in this section we will analyze how well controllers learned in simulation transfer to real robots.

In order to achieve this goal, the performance of five different set of weights resulting from five independent optimization runs in simulation was tested both in simulation and reality. Figure 5.8 shows the performance of the five controllers under four evaluation conditions: 1 robot in simulation, 1 robot in reality, 4 robots in simulation, and 4 robots in reality. For each condition, 40 evaluation runs were performed.

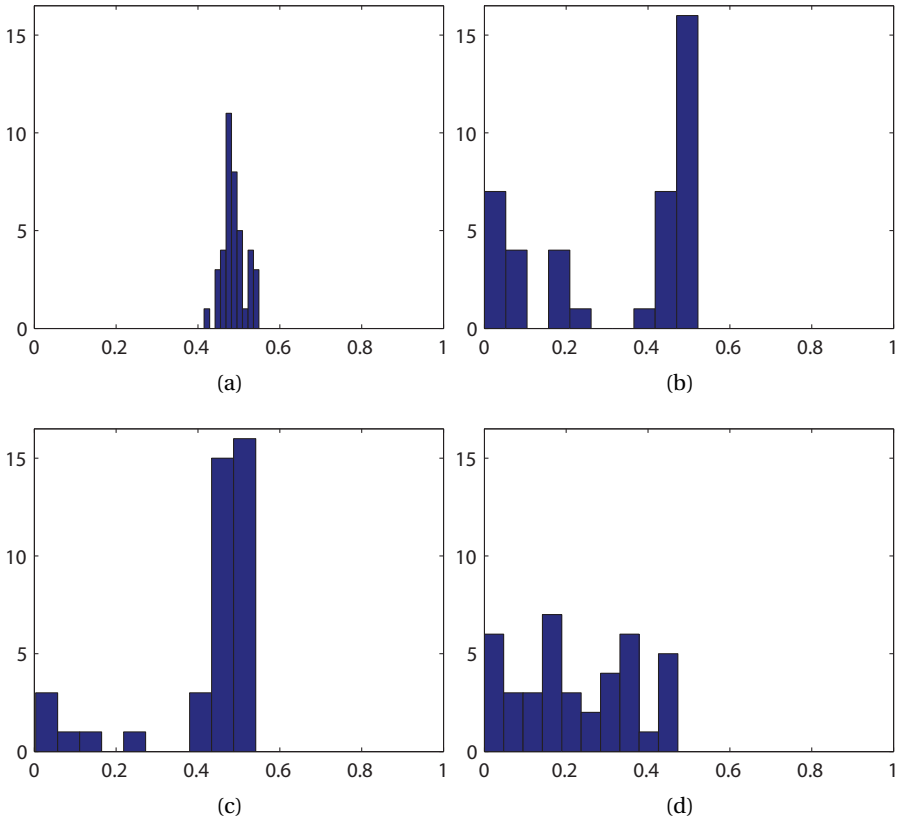


Figure 5.9: Fitness distributions for controller 3 under different evaluation conditions. The horizontal axis represents the fitness, the vertical axis the number of evaluations. The cumulative value of all bins is 40, the total number of evaluations. (a) Evaluated in simulation with 1 robot. (b) Evaluated in reality with 1 robot. (c) Evaluated in simulation with 4 robots. (d) Evaluated in reality with 4 robots.

In general, it can be noted that single robot evaluations perform better than multi-robot ones. This is to be expected, since as the number of robots increases, the arena becomes more crowded and obstacle avoidance becomes harder, especially when there is no explicit coordination scheme among robots.

Also, the performance in simulation is higher than with real robots. In fact, it is interesting to note that the difference in performance between simulation and reality is significantly larger with 4 robots than with 1 robot in all cases except for Controller 4 (average performance difference: 24% for 4 robots and 16% for 1 robot).

Further insight on the difference between simulation and reality can be gained by analyzing the fitness evaluation distributions. Figure 5.9 shows the distributions for Controller 3, the one with highest difference between simulation and reality.

For a single robot simulation, the fitness distribution is bell-shaped and has a relatively

low variance (Figure 5.9a). There is no statistically significant difference with a Gaussian distribution (one-sample Kolmogorov-Smirnov test, $p = 0.531$). This distribution is the result of the added effects of the sensor noise, the actuator noise, and the random initial pose in the arena. These are the only sources of uncertainty in the single robot simulation, and they all present Gaussian or uniform distributions.

In the real single robot case (Figure 5.9b), several evaluations have performances similar to the ones in simulation (16 evaluations in the 0.5 fitness bin), but there are a few cases in which the robot gets stuck, resulting in a very poor performance. This effect is probably due to two physical details that were not modelled in the simulation: the plastic cover added to the robots to protect them against collisions, which has a higher friction against walls and therefore facilitates getting stuck, and the wireless card, which protrudes from the body of the robot causing the back proximity sensor to be less exposed. The difference between the simulated and real distributions is statistically significant (two-sample Kolmogorov-Smirnov test, $p = 4.7e-5$).

In the multi-robot simulation (Figure 5.9c), most evaluations have fitness values that are slightly lower than the single robot simulation, but we can notice failed runs that were not seen in the single robot simulation, which decrease the average performance considerably. These failed runs occur when a robot gets blocked in a corner or against a wall by other robots, a new source of uncertainty that does not seem to follow a Gaussian distribution (one-sample Kolmogorov-Smirnov test, $p = 1.2e-4$).

Finally, the multi-robot real evaluations have the lowest performance of all cases (Figure 5.9d), due to the combination of unmodelled effects and the interference between robots. The difference between the simulated and real distributions for 4 robots is statistically significant (two-sample Kolmogorov-Smirnov test, $p = 3.7e-10$) and larger than in the case of 1 robot (Kolmogorov-Smirnov distance of 0.725 and 0.5 respectively).

5.7 Optimization Runs with Real Robots

In Section 5.6 we have shown a noticeable performance drop when evaluating with real robots controllers optimized in simulation. The goal of this section is to determine whether simulations can still be used to reduce the required real robot evaluation time when optimizing controllers for real robots through a hybrid approach.

Simulation time is negligible when compared to real-robot evaluations (the speed-up factor is approximately 40x), but simulated solutions do not always transfer well to real robots. The hybrid PSO solution solves this problem by running initially PSO in simulation in order to learn best solution strategies, transferring afterwards the candidate solution pool to real robots, and finally optimizing on real robots to obtain refined solutions.

Using the parameters from Table 5.2 and $N_{rob} = 4$, we did 30 optimization iterations in

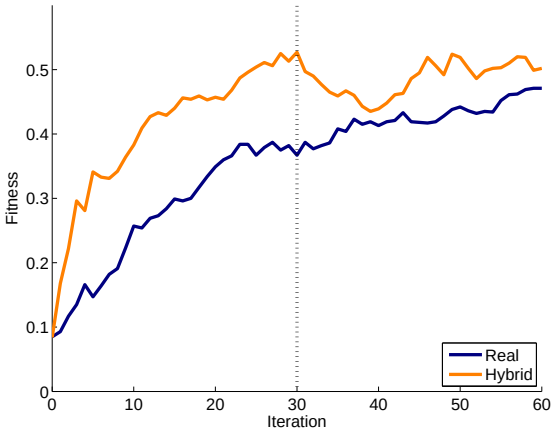


Figure 5.10: Average population fitness per iteration for a single run of PSO with four real robots and a single run of hybrid PSO. For hybrid PSO the pool of candidate solutions was transferred from simulation to real robots at iteration 30 (marked with a vertical dashed line).

simulation, transferred the pool of candidate solutions to real robots, and did another 30 iterations. We compared this hybrid approach to 60 iterations ran exclusively on real robots. Result for a single run of both approaches are shown in Figure 5.10.

In the hybrid approach, after the switch from simulation to reality at iteration 30, the average population fitness decreases for approximately 10 iterations as previous bests are re-evaluated and lower real performances are averaged with the previous simulated ones. After this initial decrease, the performance starts increasing again as new solutions that perform better in reality are found and shared through the population.

In the optimization run using real robots exclusively, the average population fitness increases constantly but at a much lower rate. The hybrid approach reaches a slightly higher final performance while requiring half the real robot evaluation time (2h18min vs 4h36min).

Furthermore, both approaches converge to the same straight line strategy discussed in Section 5.3. Therefore, even though simulated controllers do not perform as expected when transferred to real robots, a hybrid simulate-and-transfer approach coupled with a noise-resistant algorithm that re-evaluates performances at each iteration may help to reduce the required experimental time as well as the wear on the equipment.

5.8 Chapter Summary

The total evaluation time for PSO depends on four factors: population size, individual candidate evaluation time (evaluation span), number of iterations of the algorithm, and number of re-evaluations of the personal best of each candidate solution. In a distributed implementation, fitness evaluations are distributed among multiple robots, thus the required wall-clock time is reduced.

In this chapter, we explored the algorithmic parameters that determine total evaluation time with the goal of implementing the adaptation process with a limited amount of time determined by the robots' battery autonomy. Each parameter was varied independently, and based on the resulting fitness we proposed guidelines to choose these parameters when a fixed limited total evaluation time is given. To add more generality to our guidelines, we ran analogous tests on numerical benchmark functions having the same dimension of the optimization problem as that characterizing the robotic controllers.

For the population size parameter, we suggested to use at least the dimension of the search space D, and to use the number of robots if it is greater than D to take advantage of parallel evaluations and increased robustness. We proposed using the robot speed and environment size as guidelines to choose an evaluation span that guarantees at least one interaction of the robot with other components of the environment relevant to the task at hand. Due to the inherent uncertainty in controller evaluations when using more than one robot, we showed that re-evaluations seem to be a good recommendation in multi-robot learning scenarios. The last parameter, the number of iterations, can be adjusted to fit the total evaluation time available.

By applying our guidelines, we were able to optimize 24 control parameters for the obstacle avoidance task within 2 hours (the robots' energy autonomy). This resulted in a maximum quantitative performance drop of 17% but with no observable difference in the qualitative behaviors of the solutions.

We then compared the performance of simulated and real robots. Using trajectory analysis, we studied the best solution strategies and showed that both simulation and real-robot optimization converged to the same strategy of going back and forth between walls in both the single and multi-robot case. We also found that the differences in performance between simulation and reality are more pronounced in a multi-robot setting than in a single-robot one. In addition, except for the simpler single robot simulation, the fitness distributions were non-Gaussian and multi-modal.

Finally, we tested a hybrid approach in which the pool of candidate solutions is transferred to real robots for final refinement after the initial simulation stages. We compared a single run of the hybrid approach with a single run of pure real-robot optimization, where the hybrid approach achieved a slightly higher performance while requiring half the real-robot evaluation time. These results suggest that simulation could still be employed to further reduce the required real-robot experimental time in spite of the differences in performance between simulation and reality, even though more runs are required to make these results statistically significant.

Results from this chapter were presented at the International Symposium on Distributed Autonomous Robotic Systems [77] and in the journal Robotica [78].

6 Learning Environment

The ability to move in complex environments is a fundamental requirement for robots to be a part of our daily lives. In simple environments, it is usually straightforward for human designers to anticipate the different conditions a robot will be exposed to. However, for more complex environments, the human design of high-performing controllers becomes a challenging task. This is especially true if the on-board resources of the robot are limited, as humans may not be aware of how to exploit limited sensing capabilities. Evaluative, on-board machine-learning techniques have the potential to develop specific behaviors adapted to the underlying hardware and to the environment where the robots are deployed.

Several researchers have shown that the environment has a direct influence on the outcome of robotic learning. Nolfi proposed that the behavior of a robot (and of any other agent) depends on the interaction between its controller, its body, and the external environment [79]. He observed that evolved controllers were able to produce new unexpected behaviors by testing them in different conditions. In [80], Nolfi and Parisi evolved neural network controllers for robotic exploration, switching between two different environments during the evolution process. They evolved two different neural networks: with and without the capability to learn how to behave in the environment where the robot is placed. Different behaviors resulted from evolution depending on whether learning was allowed and on the environment where the robots were tested.

Auerbach and Bongard [81] studied the relationship between environmental and morphological complexity in evolved robots, showing that many complex environments lead to the evolution of more complex body forms than those of robots evolved in simple environments.

Monirul Islam and Murase [82] evolved a robotic controller for obstacle avoidance and showed that tools from chaos theory such as the fractal dimension can be used to measure the complexity of the resulting behaviors in the learning environment and other testing environments.

Berlanga et al. [83] proposed a co-evolutive method for robot navigation where the initial positions of the robots used for evolving the controllers are also evolved. This method produced

solutions that generalized better than using a traditional evolutionary strategy method.

Nelson et al. [84] evolved robotic controllers while increasing the complexity of the environments during evolution, and showed that the evolved controllers were competitive with a hand-coded knowledge-based controller.

Inspired by the related work, our purpose in this chapter is two-fold. First, to verify whether different behaviors arise as a function of the learning environment in the chosen benchmark task of multi-robot obstacle avoidance. Secondly, to test how the learned behaviors perform in environments not encountered during learning, that is, to evaluate how general are the solutions found through the learning process.

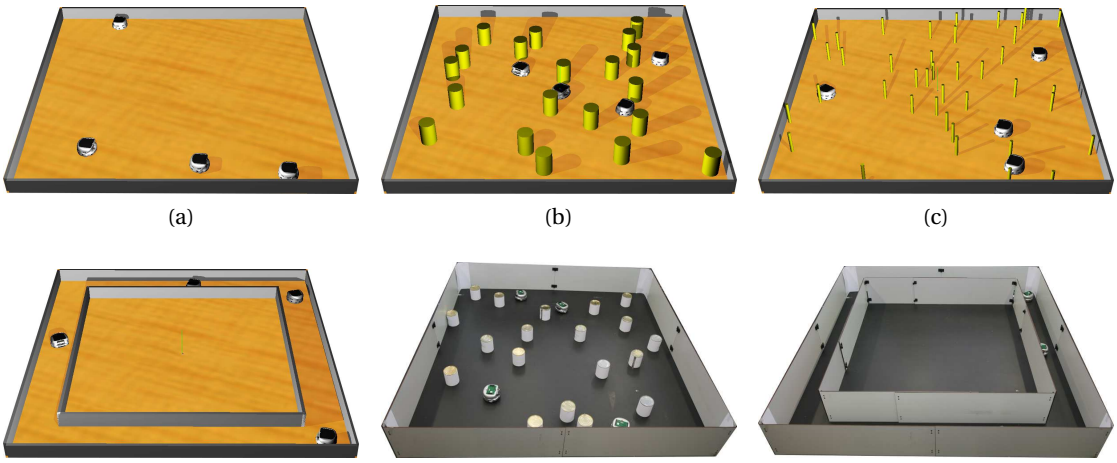
The remainder of this chapter is organized as follows. Firstly, we describe the different environments and other experimental details particular to this chapter in Section 6.1. Secondly, we perform the learning in simulation in the different environments and present the corresponding results in Section 6.2. Thirdly, the best controller from each learning environment is tested in every other environment in simulation in Section 6.3. Then, the best controller from each learning environment is also tested with real robots in Section 6.4. Finally, in Section 6.5, we conclude the chapter with a summary of our findings.

6.1 Experimental Methodology

We conduct experiments in four different environments, shown in Figure 6.1. The first one is an empty square arena of 2 m x 2 m, where the walls and the other robots are the only obstacles. The second and third environments are based on the same bounded arena, where cylindrical obstacles of two sizes are added in different numbers. The second environment has 20 medium-sized obstacles (diameter 10 cm), while the third has 40 small-sized obstacles (diameter 2 cm). The fourth environment is the same size as the empty arena with an inner wall of 1.5 m creating a continuous corridor of 25 cm width.

In simulation, the cylindrical obstacles are randomly repositioned before each fitness evaluation, meaning that the second and third environments are dynamic. In real-robot experiments, the obstacles are kept in fixed positions, the variation between runs is provided by the randomized initial pose of the robots. The third environment was not tested with real robots given the difficulty of keeping such thin cylinders vertical during collisions, but it should be noted that this kind of obstacles can occur in real environments, for example in the case of a chair or table with very thin legs.

The performance metric used in this chapter slightly differs from the one presented in Section 4.1. In order to have a wider variety of behaviors than the back and forth strategy discussed in Section 5.3, we explicitly penalize going backwards in the speed factor f_v as shown in Equa-



(a) (b) (c)

(d) (e) (f)

Figure 6.1: Different environments used in the adaptation and evaluation of the controllers. (a) Empty arena in simulation. (b) Medium-sized obstacles arena in simulation. (c) Small-sized obstacles arena in simulation. (d) Corridor arena in simulation. (e) Real medium-sized obstacles arena. (f) Real corridor arena. Real empty arena not shown.

tion 6.1.

$$f_v = \frac{1}{N_e} \sum_{k=1}^{N_e} \frac{\max\{v_{l,k} + v_{r,k}, 0\}}{2} \tag{6.1}$$

where $\{v_{l,k}, v_{r,k}\}$ are the normalized speeds of the left and right wheels at time step k and N_e is the number of time steps in the evaluation period (i.e., the evaluation time t_e divided by the time step of the controller, which is in our case 32 ms). The other two factors remain as described in Section 4.1.

All experiments are conducted with 4 robots. The method for initializing the robots' pose for each fitness evaluation is different between simulation and experiments with real robots. In simulation, the initial positions are set randomly with a uniform probability distribution, verifying that they do not overlap with obstacles or other robots. For the experiments with real robots, in the empty arena a random speed is applied to each wheel for three seconds to randomize the robots' pose. In the two arenas with obstacles and in the corridor one, the robots are manually repositioned to avoid disturbing the location of the obstacles, and then the robots turn in place with a random speed for two seconds to randomize their orientation.

The PSO algorithmic parameters are set following the guidelines for limited-time adaptation we presented in the previous Chapter and are shown in Table 6.1.

Table 6.1: PSO parameter values

Parameter	Value
Number of robots N_{rob}	4
Population size N_p	24
Iterations N_i	200
Evaluation span t_e	40 s
Re-evaluations N_{re}	1
Personal weight w_p	2.0
Neighborhood weight w_n	2.0
Dimension D	24
Inertia w_I	0.8
Initial range X_{init}	20

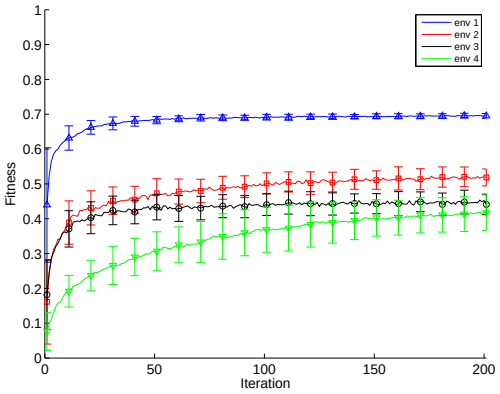
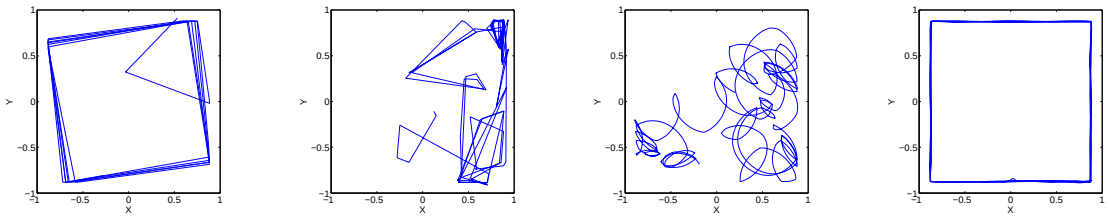


Figure 6.2: Best fitness found at each iteration for 100 PSO optimization runs. Bars represent the standard deviation across runs. Fitness in empty arena in blue (env 1). Fitness in arena with 20 medium cylindrical obstacles in red (env 2). Fitness in arena with 40 small cylindrical obstacles in black (env 3). Fitness in corridor arena in green (env 4).

6.2 Learning in Simulation

Since PSO is a stochastic optimization method and the performance measurements are noisy, each PSO optimization run may converge to a different solution. Therefore, for statistical significance, we performed in simulation 100 PSO learning runs for each learning environment. Figure 6.2 shows the progress of the PSO learning at each iteration for the four environments. Vertical bars show the standard deviation among the 100 PSO runs.

The highest performance corresponds to the empty arena since it is the easiest environment with just the bounding walls and the other robots acting as obstacles. The fitness in both environments with cylindrical obstacles is very similar for the whole learning process. The slowest learning rate occurs for the narrow corridor, indicating that this environment is more challenging for the learning algorithm. By the end of the adaptation process the performance is slightly lower than in the arenas with cylindrical obstacles.



(a) (b) (c) (d)

Figure 6.3: Trajectories of one of the four robots during a single experiment in simulation for the controllers learned in the four environments under study. (a) Empty arena. (b) Medium sized obstacles arena. (c) Small sized obstacles arena. (d) Corridor arena.

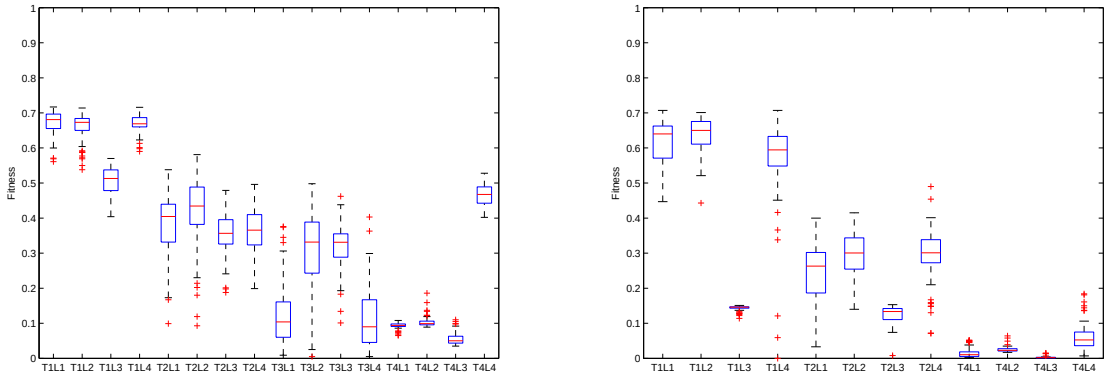
It should be noted that the learning environment has a significant impact in the variation between runs, as the standard deviation is lowest in the empty arena and it increases markedly for the more complex environments.

Trajectories can be a useful tool to identify the behavior of the robots, as we have seen in [75]. Figure 6.3 shows the resulting trajectories of the best learned behaviors in simulation for each environment where adaptation took place. As opposed to the results from Section 5.3, the back and forth strategy strategy did not arise due to the explicit penalization for going backwards enforced in the modified metric. Instead, it can be seen how in the empty arena and in the medium-sized obstacles arena the robot trajectories are straight until they find an obstacle (wall, cylindrical obstacle, or other robot), performing then a sharp turn and continuing straight afterwards.

The trajectory learned in the arena with small-sized obstacles is curvilinear when there are no obstacles within range. When the robot detects an obstacle, it makes a sharp turn to later continue its curvilinear movement. The small obstacles are thinner than the distance between two contiguous infra-red sensors, so sometimes the robots are not able to detect them. Curvilinear movements may help in avoiding getting stuck in front of the small obstacles, and thus the behavior learned with PSO does not involve moving in straight lines as in the other cases.

In the corridor arena, the robot moves along the corridor, turning 90 degrees to head into the following sub-corridor, and thus exploring the whole arena.

As expected, the different environments cause the robots to learn different behaviors. In the next section we will show how the learned controllers behave in the other environments that were not encountered during learning.



(a) (b)

Figure 6.4: Boxplot showing the fitness of the four learned controllers (L1-L4). (a) Evaluated in the four testing environments (T1-T4) in simulation. (b) Evaluated in three testing environments (T1, T2 and T4) with real robots. The box represents the upper and lower quartiles, the line across the middle marks the median, and the crosses show outliers.

6.3 Testing in Simulation

In the previous section, we obtained four different controllers corresponding to each environment where learning took place. In this section, we test the controllers in all environments to see how they perform in situations not encountered while learning, i.e., to see how general and robust are the obtained behaviors.

Figure 6.4a shows the boxplot of the fitness of 20 evaluation runs performed in simulation for each controller and testing environment. Since all experiments are conducted with 4 robots, this results in 80 fitness measurements per controller and environment. For the sake of brevity, we use T to denote testing environment, L for learning environment, and we number the environments from one to four in the following order: empty arena, arena with 20 medium cylindrical obstacles, arena with 40 small cylindrical obstacles and corridor arena. Thus, $T1L4$ for instance should be read as: test performed in the empty environment with the controller learned in the corridor environment.

As expected, for each environment, the controller learned in the testing environment has the highest performance. However, for the simplest environment ($T1$), there is no significant difference between the performance of controllers $L1$, $L2$, and $L4$. Regarding the generality of the learned behaviors, controller $L4$ seems to be the most robust, as it significantly outperforms all other controllers in the corridor and still performs almost as good as $L1$ in $T1$ and reasonable well in $T2$, although it performs poorly in $T3$.

Further insight on the performances can be obtained by analyzing the trajectories described by the robots in the different environments. Out of the 16 evaluation conditions, we show the ones we consider most interesting in Figure 6.5.

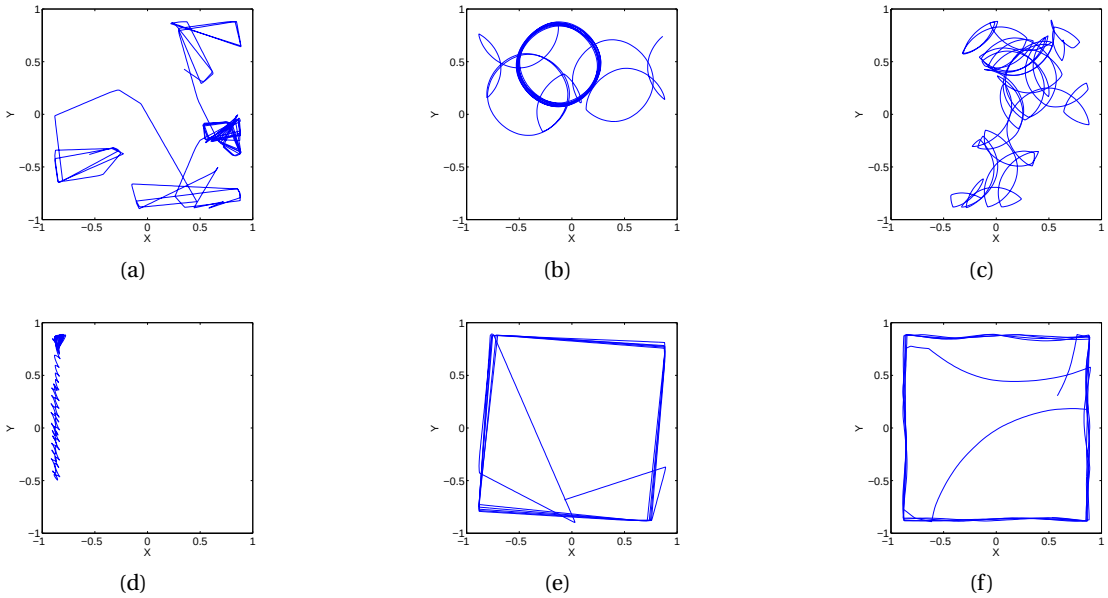


Figure 6.5: Trajectories of one of the four robots during a single experiment in simulation for different learned controllers (LX) and testing environments (TX). (a) T2L1. (b) T1L3. (c) T2L3. (d) T4L3. (e) T1L4. (f) T1L4*.

The behavior of controller $L1$ is similar to that of controller $L2$ in all testing environments since they employ similar avoidance strategies: moving in straight lines and making sharp turns near obstacles (for example, compare the trajectories from Figure 6.5a and Figure 6.3b). This result becomes evident when considering that the medium-sized cylindrical obstacles in Environment 2 are very similar in shape and size to the Khepera III robot. However, maybe due to the higher obstacle density of Environment 2, controller $L2$ is more robust in the sense that it performs better in Environments 3 and 4.

The curvilinear behavior of controller $L3$, which enables it to avoid very thin obstacles, is also observed with the larger obstacles of Environment 2 (Figure 6.5c), and results in fully circular trajectories in the empty environment (Figure 6.5b). However, this controller as well as controllers $L1$ and $L2$ were not able to move along the corridor, doing instead short straight movements alternated with sharp turns (Figure 6.5d).

Controller $L4$ was the only one able to move smoothly along the corridor in Environment 4, and it performed well in all environments except $T3$. The behavior learned can be observed when tested in the empty environment ($T1L4$) in Figure 6.5e. The robot moved straight performing a 90 degree sharp turn when finding an obstacle. This exact 90 degree turn was learned in the corridor environment to perform the transition from one sub-corridor to another.

As mentioned previously, we run 100 PSO runs for each environment, and controller $L4$ is the best-performing one from the 100 runs in the corridor environment, but we noticed that not all the resulting controllers have the same behavior. A different controller resulting from the

corridor environment is shown for the empty arena ($T1L4*$) in Figure 6.5f. The robot learned a wall-following behavior, performing a curvilinear movement in the absence of obstacles.

However, when testing this controller in the corridor ($T4L4*$) the trajectory looks exactly the same as the one from $T4L4$. Thus, it is interesting to notice that this behavior could only be observed when testing in other environments than the learning one, which shows the importance of using varied environments to observe the whole range of behaviors of a given controller.

6.4 Testing with Real Robots

In order to validate the results obtained in simulation, we tested the same controllers from the previous section with real robots in Environments 1, 2, and 4. We did 20 evaluation runs with 4 robots, leading to 80 fitness measurements per case. The resulting fitness is shown in Figure 6.4b.

As in simulation, the performance of controllers $L1$ and $L2$ was similar. Again, controller $L4$ seemed to be the most robust, outperforming all other controllers in the corridor and performing similarly to the best controllers in the other two environments.

Controller $L3$ suffered a noticeable performance drop when going from simulation to reality due to an unmodeled effect: the Khepera III motors' were not able to work smoothly at low speeds, and thus the inner wheel in the circular movements in open spaces was practically stopped, resulting in circles with a very small radius.

Finally, controller $L4$ was also able to move along the corridor as in simulation, although the behavior was not as smooth and turns midway through the corridor were more frequent than in simulation (probably due to inaccuracies in the sensor model and the increased noise in real environments). Thus, the real-world performance was lower.

6.5 Chapter Summary

In this chapter, we studied the effect of the environment on the multi-robot learning of an obstacle avoidance behavior. We showed that the same controller architecture, fitness function, and learning algorithm implemented in different environments lead to different avoidance behaviors, such as moving in straight lines with sharp turns, curvilinear movements, and wall-following around obstacles.

We then tested the learned controllers in environments not encountered during learning, both in simulation and with real robots, which allowed us to see the full range of behaviors of each controller. Finally, we saw that no single learning environment was able to generate a behavior general enough to succeed in the four testing environments. Using multiple environments during learning could be a possible way to address this issue.

Results from this chapter were presented at the European Conference on Artificial Life [85].

7 Analysis of Control Architecture

In the previous chapter, we saw how more complex environments result in a harder obstacle avoidance task. Increasing the complexity of the controller is one possible way to obtain a higher performance in these challenging scenarios. In particular, for artificial neural networks this can be achieved by increasing the number of neurons and sensory inputs, adding memory in the form of recurrence, or adding non-linear operators. In fact, it has been shown that feedforward neural networks with a single internal, hidden layer and sigmoidal nonlinearities are universal approximators, that is, they can uniformly approximate arbitrary functions given enough hidden units [69], [70].

However, this increased complexity of the controller may result in a harder problem for the learning algorithm. A harder problem may mean a search space of higher dimension, noisier performance measurements, narrower optimal parameter regions, or slower convergence.

Thus, the goal of this chapter is to quantify the trade-offs between the complexity of computer-synthesized controllers and their performance in different environments. For this purpose, we will use the multi-robot obstacle avoidance benchmark task.

The remainder of this chapter is organized as follows. First, in Section 7.1, we describe the experimental methodology, comprising controllers, environments, and algorithmic parameters. In Section 7.2 we present and discuss the results from the different experiments. Finally, Section 7.3 concludes the chapter with a summary.

7.1 Experimental Methodology

In order to analyze the trade-offs between controller complexity and performance in different environments, we perform a set of twelve experiments, involving six controller architectures (described in Subsection 7.1.1) and two environments (Subsection 7.1.2). Algorithmic parameters common to all experiments are described in Subsection 7.1.3. The learning is conducted in simulation, and the best solutions are later tested both in simulation and with real robots.

Table 7.1: Summary of Controller Architectures

Controller	# Parameters	# Sensors	Linear	Memory
brait2a	2	2	Yes	No
brait2b	2	2	Yes	No
brait10	10	4	Yes	No
brait20	20	9	Yes	No
ann20	20	9	No	No
ann24	24	9	No	Yes

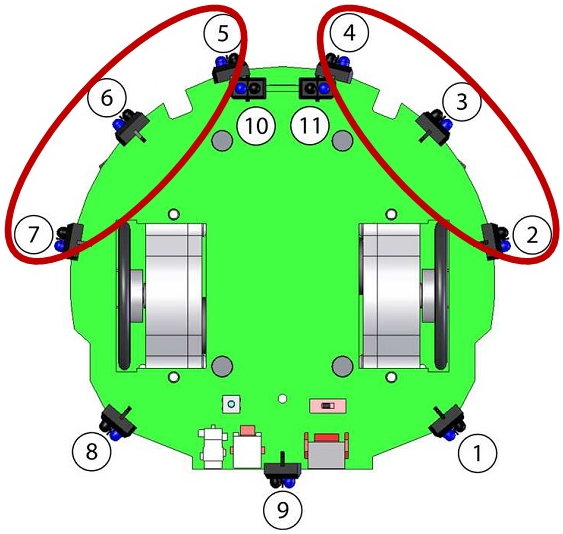


Figure 7.1: Schematic bottom view of the Khepera III robot showing the IR sensor grouping used in controllers *brait2a* and *brait2b*.

7.1.1 Controllers

Six controllers of incremental complexity are used to understand the effect of complexity in the adaptation process and its relationship with the environment. The incremental complexity is achieved by increasing the number of sensors used as inputs, adding non-linearities, and adding memory in the form of recurrent neural network connections. Table 7.1 presents a summary of the controller architectures.

The two simplest Braitenberg controllers use only two parameters. They both take as inputs two virtual sensors, left and right, obtained from averaging and normalizing the sensor values of the three front sensors situated at the left and right sides of the robot, disregarding the three sensors in the back part. This grouping is shown in Figure 7.1.

Controller *brait2a* uses one parameter for each virtual sensor, and a fixed bias speed set at the maximum, which results in two control parameters (see Figure 7.2 for a schematic representation).

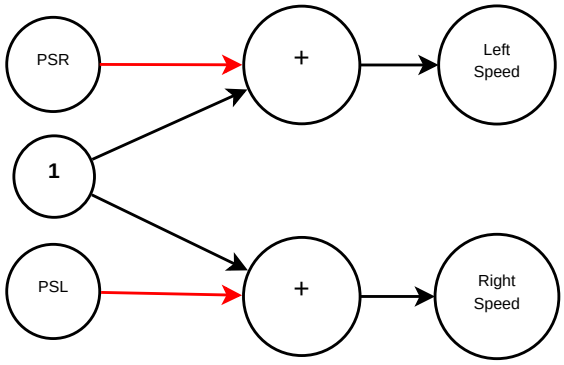


Figure 7.2: *brait2a* controller architecture. *PSL* and *PSR* represent the left and right virtual proximity sensors, and the connections shown in red have weights which are parameters of the controller.

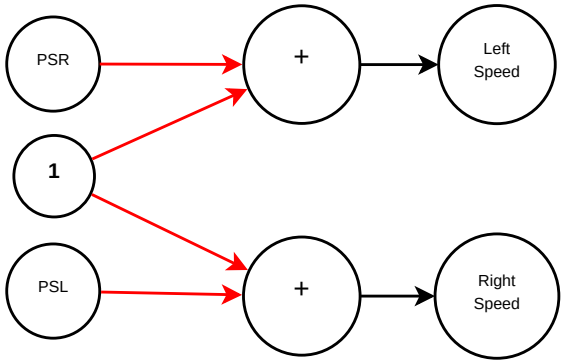


Figure 7.3: *brait2b* controller architecture. *PSL* and *PSR* represent the left and right virtual proximity sensors, and the connections shown in red have weights which are parameters of the controller. Note that the two connections to the virtual proximity sensors use the same first parameter value, and the two connections to the bias speed use the same second parameter value.

Equation 7.1 specifies the normalized wheel speeds $\{v_l, v_r\}$ for controller *brait2a*, where w_0 and w_1 are the parameters to be optimized; and i_l and i_r the virtual left and right sensors.

$$\begin{aligned} v_l &= 1 + w_0 \cdot i_r \\ v_r &= 1 + w_1 \cdot i_l \end{aligned} \tag{7.1}$$

Controller *brait2b* also has two parameters, but obtained differently: the same parameter value is used for the weights of the connections to both virtual sensors, and the bias speed is added as the second parameter (schematic representation in Figure 7.3).

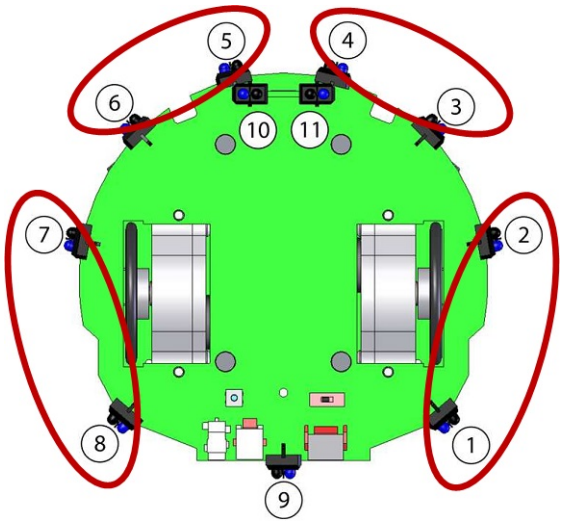


Figure 7.4: Schematic bottom view of the Khepera III robot showing the IR sensor grouping used in controller *brait10*.

Equation 7.2 defines the *brait2b* controller:

$$\begin{aligned} v_l &= w_0 + w_1 \cdot i_r \\ v_r &= w_0 + w_1 \cdot i_l \end{aligned} \tag{7.2}$$

The *brait10* controller uses ten parameters. It takes as inputs four virtual sensors (front-left, front-right, back-left and back-right) obtained from averaging and normalizing in pairs the sensor values of eight sensors of the robot and discarding the central sensor in the back part. This grouping is shown in Figure 7.4.

Equation 7.3 defines the normalized wheel speeds $\{v_l, v_r\}$ for controller *brait10*, where $\{w_0, \cdots, w_{10}\}$ are the parameters to be optimized; and $\{iv_1, \cdots, iv_4\}$ represent the four virtual sensors.

$$\begin{aligned} v_l &= w_0 + \sum_{k=1}^{4} iv_k \cdot w_k \\ v_r &= w_5 + \sum_{k=1}^{4} iv_k \cdot w_{k+5} \end{aligned} \tag{7.3}$$

The ten parameters are the eight weight connections to the virtual sensors (two connections per sensor per wheel), and the two bias speeds, as schematized in Figure 7.5.

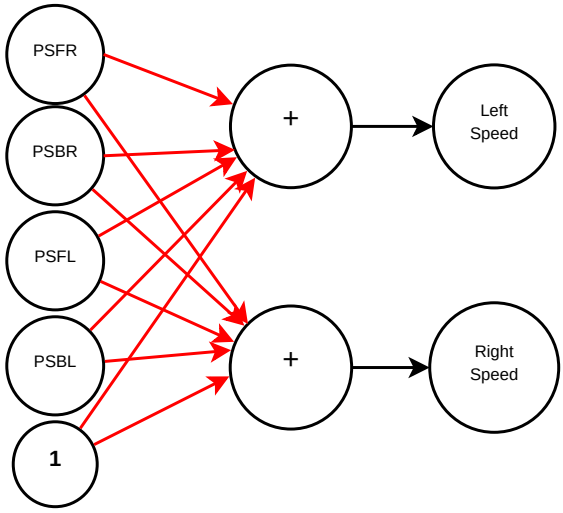


Figure 7.5: *brait10* controller architecture. *PSFL*, *PSFR*, *PSBL*, and *PSBR* represent the front-left, front-right, back-left and back-right virtual proximity sensors respectively, and the connections shown in red have weights which are parameters of the controller.

The *brait20* controller uses all individual proximity sensors as inputs, as schematized in Figure 7.6.

The wheel speeds $\{v_l, v_r\}$, given by Equation 7.4, depend on the normalized proximity sensor values $\{i_1, \cdots, i_9\}$, and the 20 weight parameters being optimized $\{w_0, \cdots, w_{19}\}$ (one weight per proximity sensor per wheel, and the two wheel speed biases).

$$\begin{aligned} v_l &= w_0 + \sum_{k=1}^{9} i_k \cdot w_k \\ v_r &= w_{10} + \sum_{k=1}^{9} i_k \cdot w_{k+10} \end{aligned} \tag{7.4}$$

The last two controllers introduce non-linear sigmoid activation functions, which is the non-linear function $s(\cdot)$ applied after summing all the inputs and given by Equation 7.5. This non-linearity is why we drop the Braitenberg denomination and switch to neural networks.

$$s(x) = \frac{1}{1 + e^{-x}} \tag{7.5}$$

The *ann20* controller is a non-recurrent artificial neural network of two units with sigmoidal activation function $s(\cdot)$. The outputs of the units define the wheel speeds $\{v_l, v_r\}$, as shown in

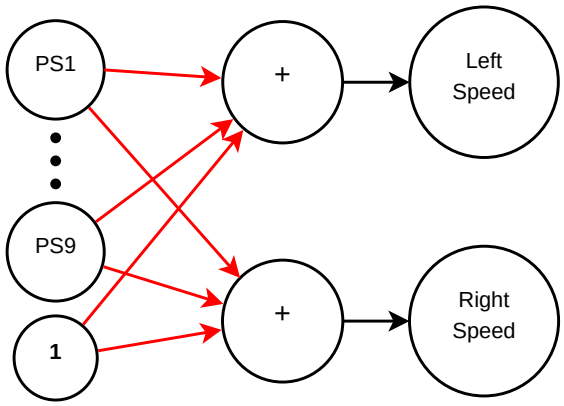


Figure 7.6: *brait20* controller architecture. *PS1* to *PS9* represent the nine lateral IR proximity sensors of the Khepera III robot, and the connections shown in red have weights which are parameters of the controller.

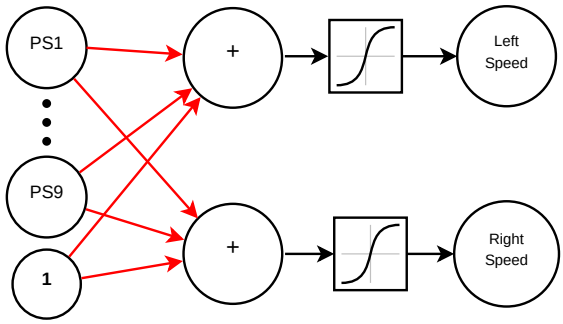


Figure 7.7: *ann20* controller architecture. *PS1* to *PS9* represent the nine lateral IR proximity sensors of the Khepera III robot, and the connections shown in red have weights which are parameters of the controller.

Equation 7.6.

$$\begin{aligned} v_l &= s(w_0 + \sum_{k=1}^{9} i_k \cdot w_k) \\ v_r &= s(w_{10} + \sum_{k=1}^{9} i_k \cdot w_{k+10}) \end{aligned} \tag{7.6}$$

Each neuron has 10 input connections: the 9 infrared sensors and a connection to a constant bias speed, as schematized in Figure 7.7. This results in 20 parameters, the same as controller *brait20*, but with a different output given by the added non-linear sigmoid activation function.

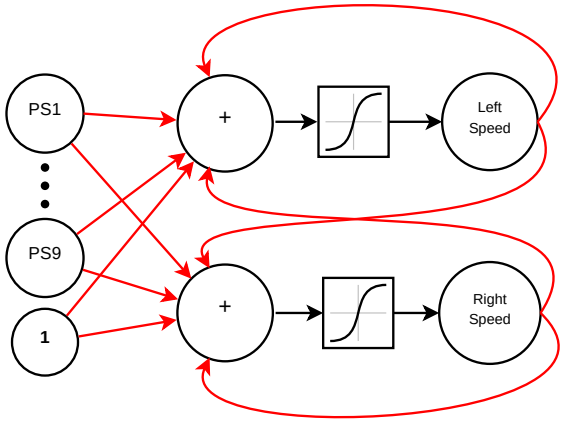


Figure 7.8: *ann24* controller architecture. *PS1* to *PS9* represent the nine lateral IR proximity sensors of the Khepera III robot, and the connections shown in red have weights which are parameters of the controller.

Finally, controller *ann24* is the default controller architecture for obstacle avoidance described in Section 4.3, and used in all the previous experiments presented in this thesis. For the sake of completeness and to highlight the contrasts with the other controllers, we will describe it again here.

The *ann24* controller is a recurrent artificial neural network of two units with sigmoidal activation functions $s(\cdot)$. By recurrent we mean that the outputs of the network from the previous time step are stored in memory and used as inputs for the next time step.

Each neuron has 12 input connections: the 9 infrared sensors, a connection to a constant bias speed, a recurrent connection from its own output, and a lateral connection from the other neuron's output, resulting in 24 weight parameters in total. A schematic representation is shown in Figure 7.8.

The outputs of the units determine the wheel speeds $\{v_{l,t}, v_{r,t}\}$, as shown in Equation 7.7.

$$\begin{aligned} v_{l,t} &= s(w_0 + \sum_{k=1}^{9} i_k \cdot w_k + w_{10} \cdot v_{l,t-1} + w_{11} \cdot v_{r,t-1}) \\ v_{r,t} &= s(w_{12} + \sum_{k=1}^{9} i_k \cdot w_{k+12} + w_{22} \cdot v_{l,t-1} + w_{23} \cdot v_{r,t-1}) \end{aligned} \tag{7.7}$$

7.1.2 Environments

We conduct experiments in two different environments. The first one is an empty square arena of 2 m x 2 m, where the walls and the other robots are the only obstacles. The second environment is the same bounded arena with cylindrical obstacles added, shown in Figure 7.9.

(a) (b)

Figure 7.9: Complex environment formed by outer walls and twenty cylindrical obstacles of different sizes. (a) Simulation arena (b) Real arena.

The obstacles have two different sizes: there are 5 large obstacles (diameter 25 cm), and 15 small obstacles (diameter 10 cm). In simulation, the obstacles are randomly repositioned before each fitness evaluation, which means that the obstacle configuration is different for each evaluation. In experiments with real robots, the obstacles are kept in fixed positions, the variation between runs is provided by the randomized initial pose of the robots.

All experiments are conducted with 4 robots. The method for initializing the robots' pose for each fitness evaluation is different between simulation and experiments with real robots. In simulation, the initial positions are set randomly with a uniform probability distribution, verifying that they do not overlap with obstacles or other robots. For the experiments with real robots, in the empty arena a random speed is applied to each wheel for three seconds to randomize the robots' pose. In the arena with obstacles, the robots are manually repositioned to avoid disturbing the location of the obstacles, and then the robots turn in place with a random speed for two seconds to randomize the orientation.

7.1.3 Algorithmic Parameters

All experiments are conducted on the multi-robot obstacle avoidance benchmark using the Khepera III robot as described in Chapter 3. For all experiments, we perform the learning using the *PSO pbest* algorithm described in Subsection 3.6.2.

The algorithm is implemented with four robots in a distributed fashion, which reduces the total evaluation time required by a factor equal to the number of robots. Each robot evaluates in parallel a different candidate solution and shares the solution with its neighbors in order to create the next pool of candidate solutions. Communication is used only to share the solutions, and the communication delay is negligible in comparison to the evaluation time of the controllers, which is 40 s. There is no explicit communication used to coordinate the motion of the robots.

Particles' positions and velocities are initialized randomly with a uniform distribution in the $[-20, 20]$ interval, and their maximum velocity is also limited to that interval.

Table 7.2: PSO parameter values

Parameter	Value
Number of robots N_{rob}	4
Population size N_p	24
Iterations N_i	200
Evaluation span t_e	40 s
Re-evaluations N_{re}	1
Personal weight w_p	2.0
Neighborhood weight w_n	2.0
Dimension D	24
Inertia w_I	0.8
Initial range X_{init}	20

The PSO algorithmic parameters are set following the guidelines for limited-time adaptation presented in Chapter 5 and are shown in Table 7.2. The parameters were chosen for the more complex controller (*ann24*) and kept the same for the simpler ones in order to keep the total learning time constant for all experiments, although simpler controllers could have been optimized with less iterations and/or less particles.

7.2 Results and Discussion

We begin by presenting the performance obtained in simulation for the twelve experimental conditions described in Section 7.1.

Figure 7.10 shows the fitness of the best set of weights found with PSO in 20 evaluation runs performed with four robots, leading to 80 fitness measurements per case. {*A, B, C, D, E, F*} represent the six controllers {*brait2a, brait2b, brait10, brait20, ann20, ann24*}, *e* stands for empty arena, and *o* stands for arena with obstacles.

As expected, the fitness in the environment with obstacles is generally lower than in the empty environment. In both environments, the greatest gain in performance is observed when adding recurrence (e.g., 27% improvement between *ann20* and *ann24* in the empty environment, 47% in the environment with obstacles).

Recurrence allows the robots to switch the direction of movement between forwards and backwards, while non-recurrent controllers move always forwards. This difference in behavior can be seen in the trajectories described by the recurrent *ann24* and non-recurrent *brait20* controllers in each environment, shown in Figure 7.11. The trajectories for the rest of the non-recurrent controllers, although not shown here, are very similar.

Interestingly, increasing the number of sensors from four to nine and adding non-linearities do not bring any significant improvement in neither environment. For example, when comparing the fitness of controller *brait10* with *ann20*, there is no statistically significant difference in

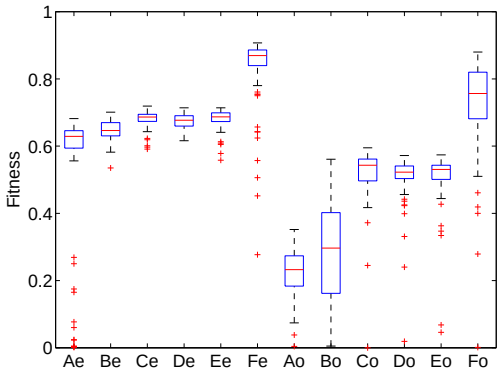


Figure 7.10: Best weights evaluated in simulation. {*A, B, C, D, E, F*} represent the six controllers {*brait2a, brait2b, brait10, brait20, ann20, ann24*}, *e* stands for empty arena, and *o* stands for arena with obstacles. The box represents the upper and lower quartiles, the line across the middle marks the median, and the crosses show outliers.

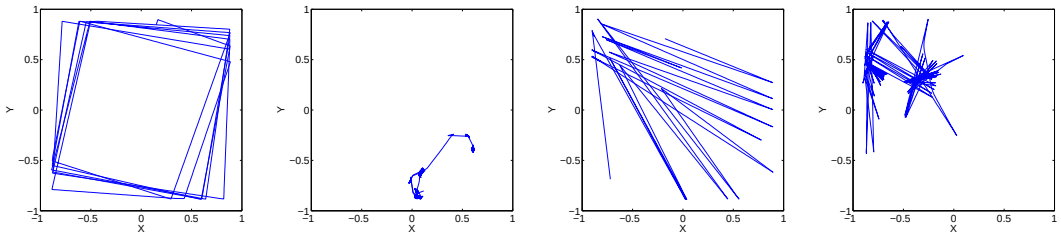


Figure 7.11: Trajectories for the best weights for recurrent and non-recurrent controllers in the two environments. (a) Controller *brait20* in empty arena (b) Controller *brait20* in obstacles arena (c) Controller *ann24* in empty arena (d) Controller *ann24* in obstacles arena.

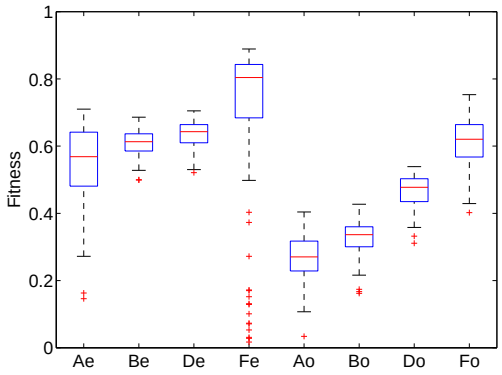


Figure 7.12: Best weights evaluated with real robots.{*A,B,D,F*} represent the 4 controllers {*brait2a, brait2b, brait20, ann24*}, *e* stands for empty arena, and *o* stands for arena with obstacles.

the empty environment (Mann-Whitney U test, $p = 0.38$), and *ann20* is slightly worse in the environment with obstacles ($p = 0.03$).

However, in the environment with obstacles, going from two to four sensors (*brait2a*/ *brait2b* to *brait10*) has a much larger impact on performance than in the empty environment. This suggests that the more complex environment requires the robot to be able to differentiate obstacles in the front and in the back, which is enabled by the two additional sensors.

In general, the performance difference between controllers is higher in the complex arena than in the empty one, meaning that fewer parameters may be used in simpler environments without a significant performance loss, but more complex environments require more complex controllers.

The best weights obtained in simulation for controllers *brait2a, brait2b, brait20, ann24* were also evaluated for 20 runs of 40 s with real robots in the two environments, and the results are shown in Figure 7.12. The performances are slightly lower, but the same trends mentioned for Figure 7.10 are observed: the controllers with more parameters perform better than the simpler ones, and this difference is larger in the environment with obstacles than in the empty one.

Since PSO is a stochastic optimization method, each PSO optimization run may converge to a different solution. Therefore, for statistical significance, we performed in simulation 100 PSO optimization runs for each experimental condition.

Figure 7.13 shows the progress of the PSO optimization for the six controllers in the two environments. Note that in this case the error bars represent the variation in the best performance found at each iteration among the 100 optimization runs, which is different from the variation in the 20 evaluation runs performed with the best solution shown in Figures 7.10 and 7.12.

The optimization takes more time to converge for the more complex controllers. For example,

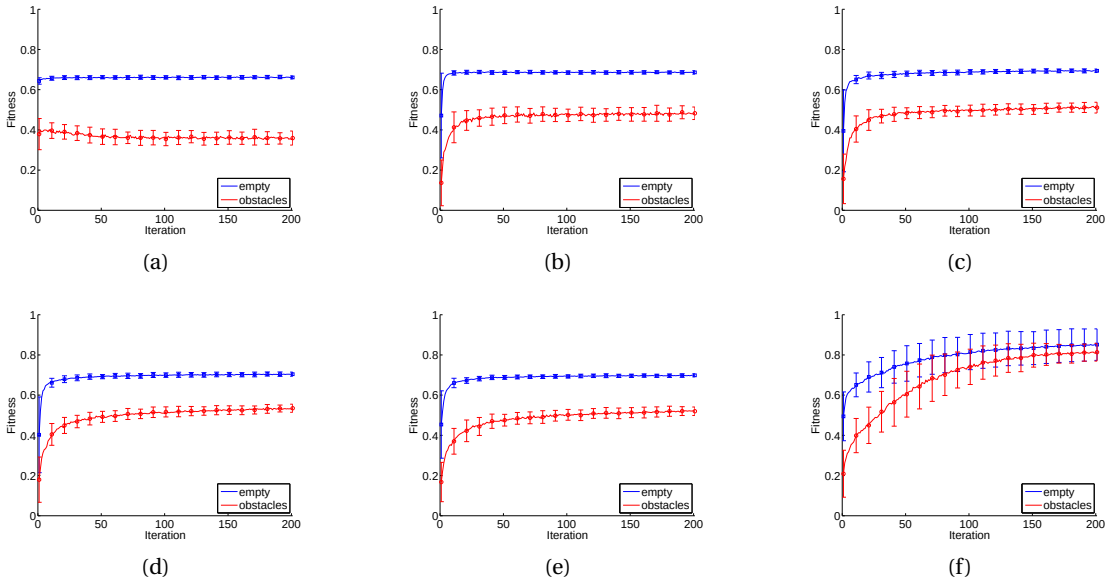


Figure 7.13: Best fitness found at each iteration for 100 PSO optimization runs. Bars represent the standard deviation between runs. (a) Controller *brait2a* (b) Controller *brait2b* (c) Controller *brait10* (d) Controller *brait20* (e) Controller *ann20* (f) Controller *ann24*

the fitness for the recurrent ANN keeps improving during the 200 iterations, while it quickly flattens before 100 iterations for the Braitenberg controllers. This is what we expected as the search for the more complex controllers takes place in a space of higher dimension.

In addition, for each controller, the optimization in the complex environment is generally noisier and takes more time to converge than in the empty environment. A notable exception is controller *brait2a*, where the average fitness does not increase with the iterations, even though the standard deviation is reduced. We suspect this is due to the fact that it is more likely to find a good solution by chance in the random initialization, as we are using 24 particles in a two-dimensional search space.

In the case of controllers *brait2a* and *brait2b* the fitness landscape can be systematically explored, which is not the case with the more complex parameters due to the high-dimensional search spaces. Figure 7.14 shows the fitness of controllers *brait2a* and *brait2b* in the two environments. Each point $(param0, param1)$ represents the average of 40 fitness measurements performed with the corresponding parameter values.

It should be noted that there is no distinct maximum in any of the settings. Instead, there are regions of high performance where a range of parameter values achieve similar fitness.

As expected, the performance of controller *brait2a* is symmetric with respect to the $w_0 = w_1$ line, given the symmetric disposition of the proximity sensors and the sensor clustering. In the arena with obstacles, solutions in the first quadrant (upper right) perform poorly, and the

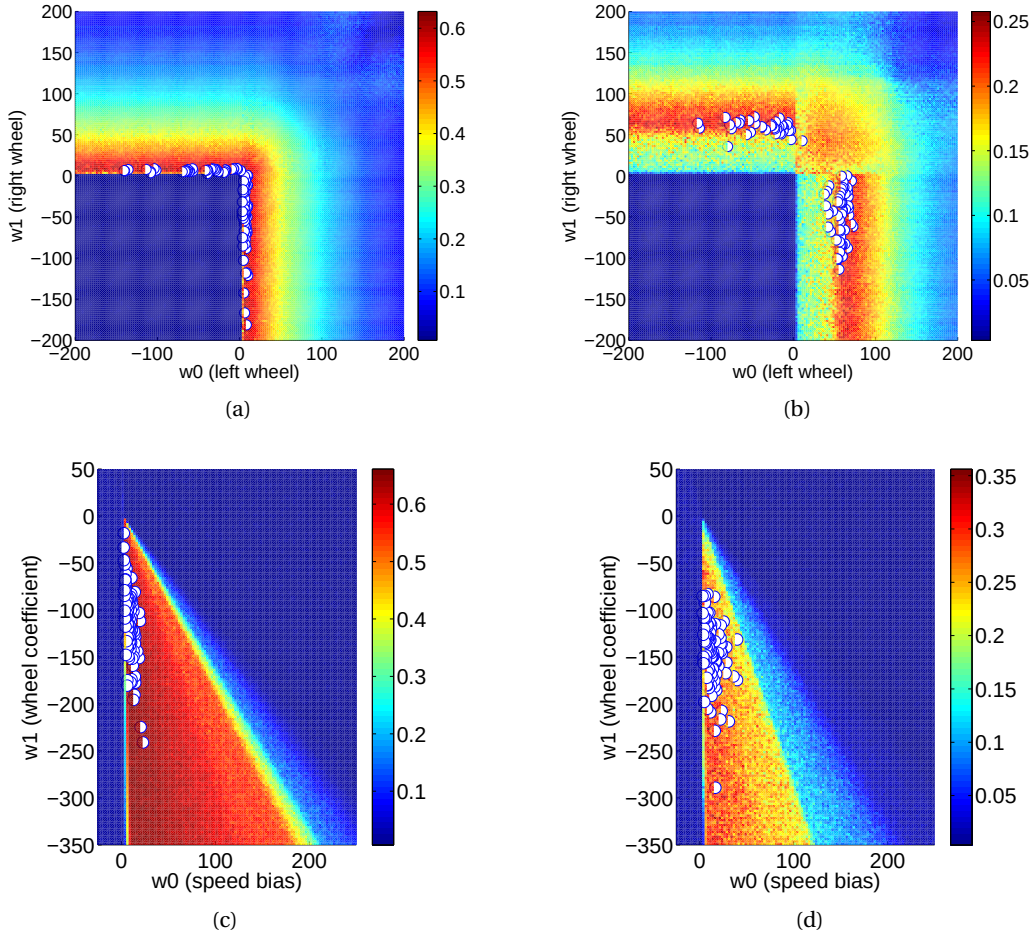


Figure 7.14: Fitness landscape for the two-parameter controllers. The solutions found by PSO are marked with white circles. (a) *brait2a* in empty arena (b) *brait2a* with obstacles (c) *brait2b* in empty arena empty (d) *brait2b* with obstacles.

best performing regions in the second and fourth quadrants (upper left and lower right) are shifted towards higher parameter values with respect to the empty arena, implying a more aggressive turning behavior.

Regarding controller *brait2b*, the best solutions are located in a triangular region in the fourth quadrant (lower right). This region seems to be unbounded, meaning that the bias speed parameter can increase as long as the sensor weight increases proportionally to make the robots turn around obstacles, and the actual robot speed will saturate when there are no obstacles around.

For both controllers, the fitness in the complex environment with obstacles is lower than in the empty environment, but also the best solution regions are smaller, and thus, the optimization process is harder.

The white circles in Figure 7.14 mark the best points found with the 100 PSO optimization runs. As there is no distinct maximum in the fitness landscape, the solutions are spread among the high fitness areas previously described.

7.3 Chapter Summary

Our goal in this chapter was to study the trade-offs between the complexity of controllers and the complexity of the environment in the distributed optimization of robotic controllers. For this purpose, we employed the multi-robot obstacle avoidance benchmark in which the complexity of controllers was varied by changing the number of sensors used as input, adding non-linear functions, and adding memory by using the output of the previous time step as an additional input. Experiments were conducted in two environments of different complexity, given by the number of obstacles in the environment. The optimization algorithm was the distributed, noise-resistant variation *PSO pbest*.

In the simple environment, linear controllers with only two parameters performed similarly to more complex non-linear controllers with up to twenty parameters, even though the latter ones required more iterations to be learned while the simpler ones could have been designed by hand or with a systematic search of the parameter space. Only the addition of memory resulted in a significant improvement in performance.

However, in the more complex environment, the difference in performance between controllers was more noticeable. The first significant performance improvement was seen when the number of sensors was increased so robots were able to differentiate between obstacles in the front and in the back, and the second improvement was due to the addition of memory by using the output of the previous time step as an additional input. These differences in performance justifies the use of more complex controllers with a larger number of parameters in more complex environments.

These results can also be interpreted from a different perspective. Given a limited computa-

tional budget for the optimization, if the parameter space is too large to be optimized with that budget and the environment is not too complex, sensor grouping can be used as a strategy to reduce the size of the search space.

Regarding the effects of the environment on the optimization process, we could extend the results from Chapter 6 by taking advantage of the controllers with two parameters to conduct a systematic exploration of the search space. We showed that in complex environments the optimization problem was harder in three aspects: the performance measurements were noisier, the optimal parameter region was smaller, and more iterations were required for the optimization process to converge.

Results from this chapter were presented at the International Conference on Robotics and Automation [86].

8 Comparison with Reinforcement Learning Techniques

In this chapter we will compare PSO, a population-based metaheuristic, with Q-learning, an evaluative, single-point search algorithm from the Reinforcement Learning (RL) family that has also been employed for robotic controller design. This work was conducted in collaboration with Zeynab Talebpour, who developed the methods to apply Q-learning to the obstacle avoidance benchmark task.

We chose to compare across families of algorithms because there are several previous studies comparing PSO with other methods within the population-based family, especially with Genetic Algorithms [4], [6], [7], [35]–[37], but also Evolution Strategies [38] (see Section 2.4 in the Background chapter for more details).

As we have seen in Chapter 5, PSO can be used to implement the adaptation process in multi-robot systems in a distributed fashion, which reduces the required evaluation time through parallelization, but requires the evaluation of multiple candidate solutions. Q-learning, on the other hand, refines a single policy iteratively, which may reduce the required evaluation time in comparison with the population-based algorithm.

In order to quantitatively compare the two evaluative learning techniques, we use the multi-robot obstacle avoidance benchmark task. By carefully defining our testing scenario, we can quantify the differences between the two algorithms in terms of performance, total evaluation time, and their resulting behaviors.

In the next section, we will give some background on RL, since previous work related to PSO has been already discussed in Chapter 2. Then, we will describe the methods used for the experiments in Section 8.2, focusing on how we applied RL to solve the multi-robot obstacle avoidance task.

Results will be divided into two sections, one for each algorithm. Section 8.3 will show the performance of PSO in terms of fitness and evaluation time. Next, Section 8.4 will present a similar analysis with Q-learning and discuss the differences between the two approaches. Finally, Section 8.5 will conclude the chapter with a summary of our findings.

8.1 Background on Reinforcement Learning

RL is a learning method which tries to maximize the expected cumulative reward for an agent during its lifetime using the interaction with the environment [9]. RL attempts to learn an optimal policy, that is as a mapping from the system's perceptual states to its actions, using the reward signal at each step. There have been numerous works on applying RL methods to the robotic domain. An extensive survey can be found in [10].

In particular, mobile robots have been the focus of study for a number of researchers in RL. Smart and Kaebling [87] present a framework for using RL on mobile robots with the ability to incorporate human knowledge about the task. In the initial phase, the RL system passively observes the states, actions and rewards encountered by the robot while it is directly controlled by a human. When the policy is good enough to control the robot, the RL system takes control and learning progresses as in the standard RL framework.

Yasuda and Ohkura [88] introduce Bayesian-discrimination-function-based Reinforcement Learning (BRL), which adaptively segments the state and action spaces through the learning process, eliminating the need for the state and action spaces to be designed by a human. They show that this method is effective at handling problems in multi-robot systems which operate in a dynamic environment.

In [89], Matarić proposes a formulation of RL suitable for problems in the multi-robot domain and applies it to a foraging task. This formulation minimizes the learning space of states and actions through the use of conditions and behaviors.

In this chapter, we will use Q-Learning to learn the multi-robot obstacle avoidance task. Q-learning is a common RL method which learns the utility of performing actions in particular states to derive an optimal action-selection policy for a Markov decision process [90]. In order to deal with the continuous state-action space that is often encountered in robotics, several variations of the standard Q-learning algorithm have been proposed [57], [91].

In [91] a function approximation method based on radial basis functions and Gaussian functions is used for estimating the state value function in a biped robot control problem. The learning algorithm proposed by the authors is swarm reinforcement learning, which combines concepts from population-based methods with reinforcement learning.

In [57] a neural network is used to store the Q-values for a continuous state and discrete action problem. This formulation is shown to enhance the learning ability of the agent for solving the obstacle avoidance problem in a complicated and unpredictable environment. It will be one of the two approaches that we will use to deal with the continuous nature of the multi-robot obstacle-avoidance task both in terms of the states (sensory information) and actions (wheel speeds).

8.2 Experimental Methodology

In order to compare the two algorithms, we will use the obstacle avoidance benchmark described in Chapter 3. We will first describe the experimental parameters that are common to both algorithms, then we will detail the specific aspects of the PSO approach in Subsection 8.2.1 and finally those of the Q-learning approach in Subsection 8.2.2.

For both algorithms, we will conduct experiments with one and four robots in a square arena of 1 m^2, where walls and other robots are the only obstacles.

The performance metric used in this chapter slightly differs from the one presented in Section 4.1. Due to the fact that the controller learned with Q-learning does not have memory, it cannot learn the back and forth strategy discussed in Section 5.3. Therefore, we removed the turning factor f_t to avoid penalizing turning as an avoidance strategy, resulting in a metric with two factors, both normalized to the interval [0, 1], defined in Equation 8.1.

$$f = f_v \cdot (1 - f_i) \tag{8.1}$$

$$f_v = \frac{1}{N_e} \sum_{k=1}^{N_e} \frac{|v_{l,k} + v_{r,k}|}{2} \tag{8.2}$$

$$f_i = \frac{1}{N_e} \sum_{k=1}^{N_e} i_{max,k} \tag{8.3}$$

$\{v_{l,k}, v_{r,k}\}$ are the normalized speeds of the left and right wheels at time step k, $i_{max,k}$ is the normalized proximity sensor activation value of the most active sensor at time step k, and N_e is the number of time steps in the evaluation period (i.e., the evaluation time t_e divided by the time step of the controller, which is in our case 32 ms). This metric was also used by Floreano and Mondada in their homing experiment in an empty arena in [2].

8.2.1 PSO Approach

The controller architecture for PSO is the linear Braitenberg controller with 20 parameters denominated *brait20* in Chapter 7. We decided not to use the more powerful recurrent neural network *ann24* in order to do a fair comparison with the Q-Learning approach, which does not use memory of the previous states in the action selection policy.

For the sake of completeness and to make the chapter self-contained, we will briefly recall the controller architecture. The wheel speeds $\{v_l, v_r\}$ are calculated by Equation 8.4, based on the normalized proximity sensor values $\{i_1, \cdots, i_9\}$, and the 20 weight parameters $\{w_0, \cdots, w_{19}\}$ (one weight per proximity sensor per wheel, and the two wheel speed biases).

Table 8.1: PSO parameter values

Parameter	Value
Population size N_p	20
Iterations N_i	30
Evaluation span t_e	20 s
Re-evaluations N_{re}	1
Personal weight w_p	2.0
Neighborhood weight w_n	2.0
Dimension D	20
Inertia w_I	0.8
Initial range X_{init}	20

$$\begin{aligned} v_l &= w_0 + \sum_{k=1}^{9} i_k \cdot w_k \\ v_r &= w_{10} + \sum_{k=1}^{9} i_k \cdot w_{k+10} \end{aligned} \tag{8.4}$$

The optimization problem for PSO then becomes choosing the set of weights $\{w_0, \cdots, w_{19}\}$ such that the fitness function f as defined in Equation 8.1 is maximized.

In addition to the continuous Braitenberg controller, two discrete controller versions were implemented to analyze the impact of discretization on the final performance and to compare with the Q-learning approach. In the first case, the input proximity sensor values are discretized to binary values using a threshold of 0.5, which corresponds to half of the proximity sensors' range (10 cm), and the output speeds are discretized to the closest of the 5 values $\{\pm v_{max}, \pm v_{max}/2, 0\}$. In the second case, the proximity sensor values remain continuous and the output speeds are discretized to the closest of the 3 values $\{\pm v_{max}, 0\}$.

The PSO algorithm is the *PSO pbest* noise-resistant variation introduced by Pugh et al. [7] and described in Chapter 3.

The neighborhood presents a ring topology with one neighbor on each side. Particles' positions and velocities are initialized randomly with a uniform distribution in the [-20, 20] interval, and their maximum velocity is also limited to that interval. At the beginning of each evaluation, the robots' pose is randomized to reduce the influence of the previous evaluations. At the end of each optimization run, the best solution is tested with 40 evaluations of 20 s, and the final performance is the average of these final evaluations.

The PSO algorithmic parameters are set following the guidelines for limited-time adaptation we presented Chapter 5 and are shown in Table 8.1.

8.2.2 Q-learning Approach

To solve the problem of obstacle avoidance with RL we use the Q-learning method. Q-learning attempts to learn the quality of a state-action combination $Q(s_t, a_t)$ using the update formula shown in Equation 8.5.

$$Q(s_t, a_t) := (1-\alpha)Q(s_t, a_t) + \alpha[r + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})] \tag{8.5}$$

α is the learning rate, r is the reward at each time step, and γ is the discount factor. The states are given by the proximity sensor values at each time step, and the actions are the possible wheel speeds. s_t is the current state of the robot, a_t is the current action of the robot in s_t, s_{t+1} is the next state that the robot will encounter after performing a_t in s_t and a_{t+1} stands for all possible actions that the robot can perform in its next state.

The problem of perceptual aliasing occurs when different states in the world appear to be similar from the perception of the robot but require different responses. This aliasing is due to the partial observation that our robot has of its environment. Since the robot is not aware of its absolute position in the arena and all of its surroundings including other obstacles out of its sensing range, it can receive identical sensory information in different parts of the arena. Consider a position A in the arena such as the one shown in Figure 8.1a, where there is no obstacle within sensing range but if the robot moves one step backwards it would sense an obstacle. Now consider a second position B, shown in Figure 8.1b, where again there are no obstacles within sensing range but this time if the robot moves one step forwards it would sense an obstacle. Positions A and B are mapped to the same state from the perception of the robot whereas they require different actions.

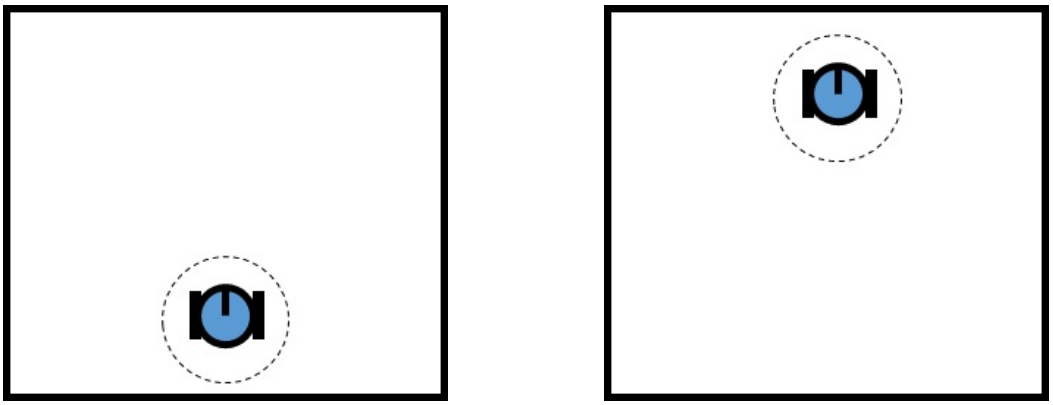
Because of this partial observation, we have chosen a Softmax probabilistic actions selection policy that allows better actions to be chosen according to how high their Q-value is, balancing exploration and exploitation [9].

The probability of selecting action a in state s is given by Equation 8.6.

$$p(s, a) = \frac{e^{Q(s,a)/T}}{\sum_{a'} e^{Q(s,a')/T}} \tag{8.6}$$

The temperature parameter T is set so that it gradually decreases and remains a small but positive value to allow the actions that are nearly as good to have a chance to be selected.

The reward signal used at each time step is the same as the fitness function that we are aiming to optimize using PSO, given by Equation 8.1. Unlike most RL problems, in this problem



(a) (b)

Figure 8.1: Depiction of perceptual aliasing: (a) and (b) are two different positions with the same sensory information that should require different actions. The black square represents the arena walls, the blue-colored shape represents a robot, and the dashed circle its sensing range.

there is not a concrete final goal. Instead, we are concerned with achieving a high fitness and maintaining it throughout the lifetime of the robot.

One key feature of RL methods which is not present in our problem formulation is the use of intermediate rewards in order to reach a goal or find a solution. Incorporating appropriate intermediate rewards can significantly speed up the learning process. However, we have chosen not to alter the reward signal from the fitness function due to two reasons. Firstly, not modifying the reward enables us to directly compare the progress of the learning in terms of fitness as a function of time with PSO. Secondly, the role of intermediate rewards in shaping the behavior of the robot makes defining an appropriate intermediate reward signal without creating misleading biases a challenging problem.

The continuous nature of the obstacle-avoidance task both in terms of the states (sensory information) and actions (wheel speeds) impedes the direct use of the standard Q-learning method. Therefore, we have chosen two different approaches to manage the complexity in the size of state and action spaces. In our first approach, state and action spaces are discretized using a fixed number of intervals. In the second approach, a neural network is used as a function approximator to store the Q-values with a continuous state space, as proposed by Huang et al. [57].

In our first Q-learning approach, the state and action space have been discretized to overcome the complexity arising from continuous state and action spaces when dealing with RL methods. Discretization decreases the size of state and action spaces, thus speeding up the learning, but may also result in a performance drop in terms of fitness.

We have discretized the sensory information using a predefined threshold to indicate safe and

Table 8.2: Q-learning parameter values for the first approach

Parameter	Value
Learning Rate α	$\alpha_0 = 1$ $\alpha_{k+1} = \alpha_k / 1.0001$ $\alpha \geq 0.3$
Discount Factor γ	0.5
Temperature T	$T_0 = 20$ $T_{k+1} = T_k / 1.0008$ $T \geq 0.05$
Binary Threshold	0.5

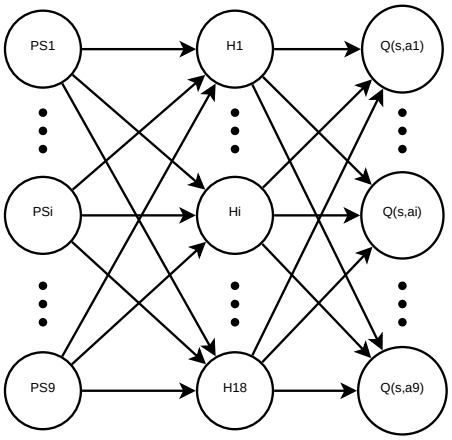


Figure 8.2: Artificial Neural Network used to calculate the Q-values for a continuous state space.

unsafe zones in terms of proximity to obstacles. This thresholding implies that we will have a binary coding of each sensor's information and the vector of all sensor values will form the state of the robot at every step.

We have set five possible speed levels for each wheel: $\{\pm v_{max}, \pm v_{max}/2, 0\}$. There are two wheels and nine distance sensors on each robot, resulting in a total number of 5^2 different possible actions and 9^2 possible states.

The learning parameter values for the first Q-learning approach are shown in Table 8.2.

In our second approach, continuous state and discrete action space Q-learning, the Q-values are stored in a neural network to allow a more compact representation of the states and also interpolation for the unvisited state-action pairs.

The structure of the fully-connected neural network is shown in Figure 8.2. There are nine continuous inputs corresponding to the nine proximity sensors $\{PS1, \ldots, PS9\}$. The hidden layer has 18 nodes: $\{H1, \ldots, H18\}$. The number of output units is given by the number of possible actions. In order to reduce the complexity of the neural network, the number of

Table 8.3: Q-learning parameter values for the second approach

Parameter	Value
Learning Rate α	$\alpha_0 = 1$
	$\alpha_{k+1} = \alpha_k / 1.000001$
	$\alpha = 0$ after episode 1000
Discount Factor γ	0.5
Temperature T	$T_0 = 20$
	$T_{k+1} = T_k / 1.000007$

possible actions was reduced with respect to the first approach. There are three action levels to choose from, $\{\pm v_{max}, 0\}$ for each wheel, which makes the total number of possible actions for every state to be $3^2 = 9$, as opposed to the $5^2 = 25$ of the first approach. The nine output nodes give the Q-values of the corresponding nine possible actions $\{Q(s,a1),\ldots,Q(s,a9)\}$. The activation function used for the hidden and output layer is sigmoid.

In every step of the simulation, the robot senses the environment through its sensors, which are the input nodes of the neural network. With this information, we calculate the outputs of the network, which specify the Q-values for that state with each output corresponding to one state-action pair. Next, a Softmax selection policy is used to select an action to be performed (Equation 8.6). Then, the reward perceived from the environment is used to calculate the new value for the selected state-action pair using the Q-learning update formula (Equation 8.5).

Given the old and new Q-values, the weights of the neural network are adjusted every step of the simulation using the backpropagation algorithm (BP) [92]. The error signal used for the adjustment is the difference between the new and old Q-values for the selected state-action pair. All other output nodes will have target values equal to their old Q-values from the previous step, and therefore the error signal will be zero for all unselected actions.

Table 8.3 contains the parameters used for the second Q-learning approach.

We have conducted 2 sets of experiments for each Q-learning approach. The first experiment involves a single robot moving in the arena, and the second experiment involves 4 robots moving in the arena at the same time. When there are 4 robots learning at the same time, there is no direct communication to assist in solving the problem. The robots play the role of dynamic obstacles for one another, creating a more complex dynamic instance of the obstacle avoidance problem. In the distributed implementation of PSO, on the other hand, there is solution sharing between neighbors which speeds up the learning process.

For the first RL approach each robot performs 1000 learning episodes of 20 s. The final performance of each robot is the average of the rewards during the last 200 episodes. We have tested every experiment 20 times and the results are the average of all runs.

For the second approach, each experiment was conducted 100 times for 1500 episodes of 20 s. The evaluation phase goes from episode 1000 to 1500.

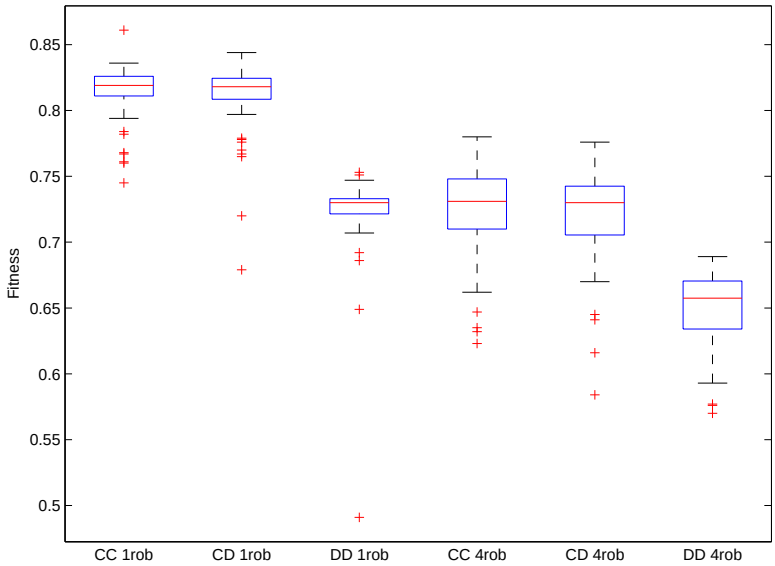


Figure 8.3: Final performance for 100 runs of PSO for three different discretization levels, each implemented using 1 and 4 robots. CC stands for continuous sensors and continuous speeds, CD continuous sensors and discrete speeds, and DD discrete sensors and discrete speeds. The box represents the upper and lower quartiles, the line across the middle marks the median, and the crosses show outliers.

8.3 PSO Results

We begin by analyzing the impact of discretization on the PSO approach. The purpose of the discretized PSO runs is to be able to perform direct comparisons with Q-learning, which works with discrete state and actions and therefore requires the discretization of proximity sensor inputs and wheel speed outputs.

Figure 8.3 shows the final performance obtained by applying PSO under six different experimental conditions: 3 different discretization levels each tested with 2 different number of robots. The 3 different discretization levels are abbreviated as follows: *CC* stands for continuous sensors and continuous speeds, *CD* continuous sensors and discrete speeds (3 possible output speeds), and *DD* discrete sensors and discrete speeds (binary sensors and 5 possible output speeds). Each discretization level was tested with 1 and 4 robots.

Discretizing only the output speeds (*CD* controllers) has no statistically significant impact on the final fitness when compared with the fully continuous controllers (*CC*), both in the single and in the multi-robot case (Mann-Whitney U test, $p = 0.32$ and $p = 0.34$ respectively).

Discretizing also the proximity sensors (*DD* controllers), however, has a noticeable impact on the fitness. For the single robot case, the mean drops from 0.82 to 0.72, a statistically significant performance difference (Mann-Whitney U test, $p = 2.7e-34$). For the multi-robot case, the mean drops from 0.73 to 0.65, which is again statistically significant ($p = 9.9e-29$).

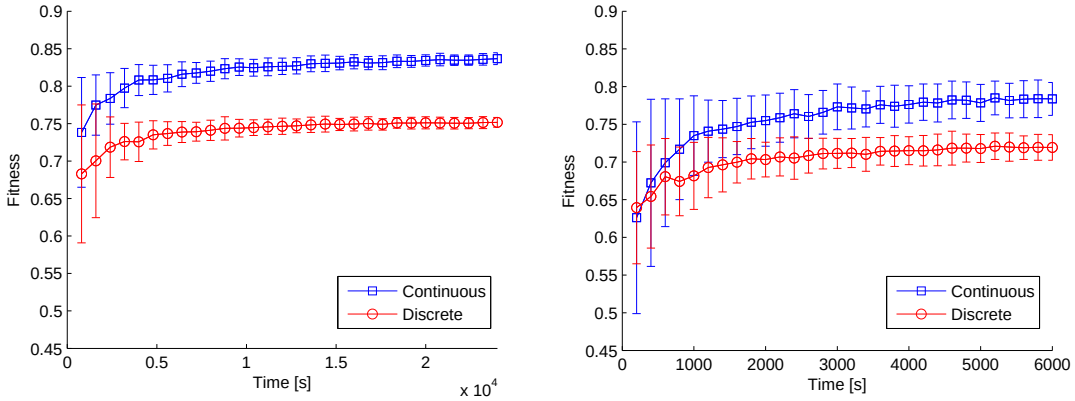


Figure 8.4: PSO best fitness as a function of time for continuous and discretized Braitenberg controllers with (a) 1 robot in the arena and (b) 4 robots in the arena. The curves are the average of 100 independent PSO runs, markers are placed at each iteration, error bars represent one standard deviation.

Figure 8.4a shows the progress of the PSO optimization as a function of evaluation time for continuous (CC) and discretized (DD) Braitenberg controllers with 1 robot in the arena. For all temporal progress graphs, the horizontal axis was converted from iterations to evaluation time in seconds to enable comparisons among algorithms regardless of how evaluation time is assigned (i.e. length of episodes, iterations, etc.).

For both controllers in Figure 8.4a, the fitness of the best solution found by PSO increases rapidly during the initial 10000 s, and then continues to increase although at a much lower pace. Also, the standard deviation between runs decreases with time as the optimization process converges towards a high-performing solution. The discretization lowers the final fitness but it does not seem to affect the convergence time of the algorithm.

Figure 8.4b shows the progress of PSO as a function of evaluation time with 4 robots in the arena, again for continuous (CC) and discretized (DD) Braitenberg controllers. When comparing between 1 and 4 robots, it can be noted that in the multi-robot case the fitness is not only lower but also much noisier due to the effect of other uncoordinated robots in the arena. Additionally, due to the distributed PSO implementation, the total evaluation time employed is reduced by a factor of 4 in the multi-robot case.

In order to separate the effect of the learning from the number of robots in the arena, we performed an additional control experiment using continuous controllers (CC) with 4 robots in the arena, where one robot is learning and the other 3 robots are avoiding obstacles with a previously optimized controller.

Figure 8.5 compares the final performance obtained with one robot learning (single robot in the arena), one robot learning and 3 other robots avoiding, and 4 robots learning in the

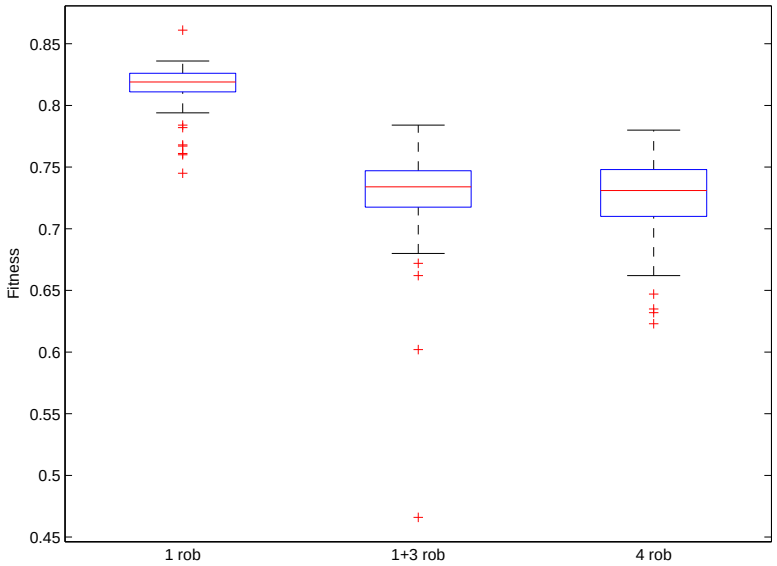


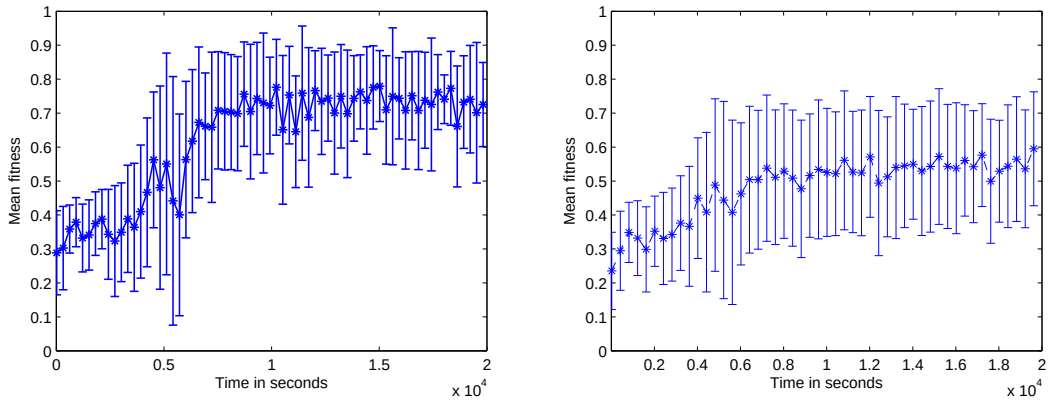
Figure 8.5: Performance of final evaluations for 100 independent runs of PSO with 1 robot learning, 1 learning and 3 avoiding, and 4 robots learning, using continuous controllers (*CC*).

arena. The performance in both cases with 4 robots in the arena is significantly lower than the case with 1 robot due to the fact that the added robots represent more obstacles in the same area. However, there is no significant difference in the final fitness between one robot learning and 3 avoiding, and 4 robots learning (Mann-Whitney U test, $p = 0.44$), which shows that distributing the adaptation process has no significant impact on the final fitness even though it reduces the required total evaluation time by a factor equal to the number of robots.

The results of the final performance for PSO under the different experimental conditions are summarized in Table 8.4. The next section presents the Q-learning results and compares them with the PSO results discussed in this section.

Table 8.4: Mean and standard deviation of the final performance for the different experiments.

Algorithm	**Sensors**	**Speeds**	**N_{rob}**	**Mean**	**Std**
PSO	discrete	discrete	1	0.72	0.04
PSO	continuous	discrete	1	0.81	0.02
PSO	continuous	continuous	1	0.82	0.02
PSO	discrete	discrete	4	0.65	0.03
PSO	continuous	discrete	4	0.72	0.03
PSO	continuous	continuous	4	0.73	0.03
Q-learning	discrete	discrete	1	0.72	0.15
Q-learning	continuous	discrete	1	0.73	0.25
Q-learning	discrete	discrete	4	0.55	0.17
Q-learning	continuous	discrete	4	0.72	0.22



(a) (b)

Figure 8.6: Mean and standard deviation of fitness using the first RL approach with (a) 1 robot in the arena and (b) 4 robots in the arena.

8.4 Q-learning Results

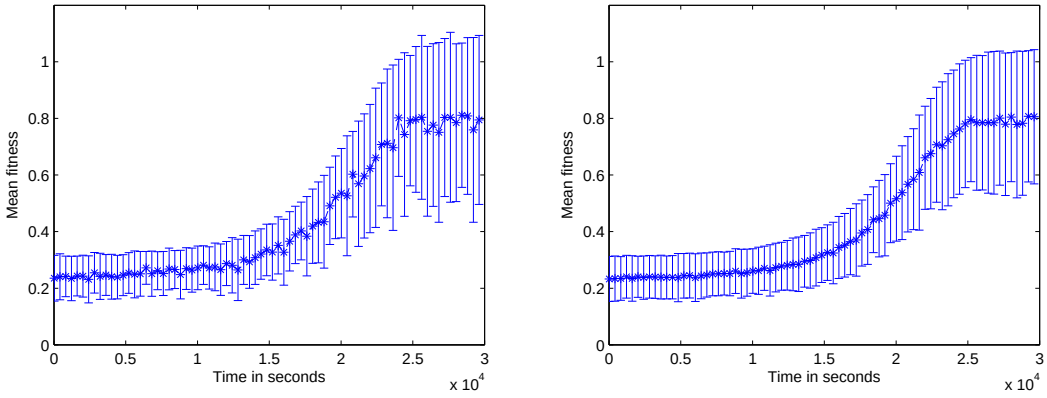
Figures 8.6a and 8.6b show the mean reward as a function of time for 20 runs of the Q-learning algorithm for single and multi-robot learning for the first RL approach. In the case of multi-robot learning, the average performance of the 4 robots is depicted.

In the single robot case (Figure 8.6a), we can see that the algorithm converges at around 8000 seconds which corresponds to episode 400 . In the 4 robot case (Figure 8.6b), the performance keeps improving after 8000 seconds until the end of the experiment, although at a much slower pace than during the initial episodes.

Both the single and the multi-robot cases show a larger standard deviation between runs than PSO (see Table 8.4 for a direct comparison of the values). This increase may be due to the probabilistic nature of the Softmax action selection policy, as other sources of uncertainties were kept constant between the different experiments.

Figure 8.7a shows the performance of the single robot using the second approach: continuous state Q-learning. We can see a convergence in the performance of the robot after the first 25000 seconds of the simulation, which shows a lower learning speed comparing to the first approach. This is partly due to the higher exploration and more gradual decrease of the temperature for the Softmax policy in the second RL approach. The standard deviation increases with time, but the coefficient of variation (ratio of the standard deviation to the mean) remains constant at a value of 0.34.

For the single robot case, the final performance obtained with both Q-learning approaches is very similar to the one obtained with PSO with discrete sensors and speeds, around 0.72 (see Table 8.4). The behavior seen is avoiding the obstacles and moving about but mainly in the center of the arena.



(a) (b)

Figure 8.7: Mean and standard deviation of fitness using the second RL approach with (a) 1 robot in the arena and (b) 4 robots in the arena.

Figure 8.7b shows the performance of the second Q-learning approach in the 4 robot scenario. The fitness and the convergence time are nearly the same as in the single robot case (compare with Figure 8.7a). The final fitness of 0.72 is very similar to the one obtained with PSO with continuous sensors and discrete wheel speeds, and it is significantly higher than the one obtained with the first Q-learning approach.

It is noteworthy to mention that Q-learning tries to find a probabilistic mapping from states to actions, whereas the Braitenberg controller calculates the output as a linear function of the inputs. Thus, the smoother behaviors seen with PSO are partly due to the nature of the controller, whereas it is easier to see behaviors with discontinuous movements like going back and forth with Q-learning. It is therefore difficult to decouple the effects of the learning from the influence of the underlying control structure. The second Q-learning approach reduces these differences with the use of a continuous state space, but the outputs of the neural network are the Q-values of every action, and not the action itself. Therefore, the neural network should not be interpreted as the robots' controller, as the output speeds are still determined with a probabilistic state-to-action mapping.

8.5 Chapter Summary

The goal of this chapter was to compare the population-based PSO approach to a different evaluative learning technique from the Reinforcement Learning family: Q-learning. For this purpose, we used the multi-robot obstacle avoidance benchmark task.

PSO was used to optimize 20 parameters of a linear Braitenberg controller. Three levels of discretization were implemented to compare with the Q-learning approaches: continuous sensors and speeds, continuous sensors and discrete speeds, and discrete sensors and speeds.

In the case of Q-learning, two different approaches were presented. In the first approach, a probabilistic policy that maps discrete states to actions was learned. For the second approach, a neural network enabled us to store the Q-values for continuous states and use the conventional Q-learning method to find an appropriate policy.

We showed that the discretization of the proximity sensors had the highest impact on the fitness for both learning algorithms. Continuous PSO had the highest fitness overall, and Q-learning with continuous states significantly outperformed Q-learning with discrete states.

Regarding the learning time, PSO and Q-learning with discrete states required a similar amount of total evaluation time for the single robot case. Both techniques converged to a solution in less than 10000 seconds. Q-learning with continuous states required more time to converge, but achieved a higher final fitness than Q-learning with discrete states. In the multi-robot case, both Q-learning approaches converged in a similar amount of time as in the single robot case but the time required by PSO was significantly reduced due to the distributed nature of the algorithm.

Results from this chapter were presented at the IEEE Congress on Evolutionary Computation [93].

Part III

Beyond Obstacle Avoidance: Learning Collaborative Behaviors and Enhancing Noise-Resistance

9 Centralized and Distributed Learning of Flocking Behaviors

So far we have been analyzing the different variables of the PSO algorithm on obstacle avoidance, a non-collaborative task. In this part of the thesis, based on what we learned in the previous analysis, we will attempt to use the same framework to learn a more complex, collaborative task such as flocking.

In Chapter 2 we have already mentioned previous work that has shown that it is feasible to use learning to generate collaborative behaviors, either by centralized approaches [49]–[51], or by distributed approaches on low-dimensional search spaces [47], [48]. However, in this chapter we will attempt to distribute the learning of a collaborative task on a large parameter space, which, to the best of our knowledge, has not been attempted before. In order to achieve this goal, we will first design a global metric and perform the learning in a centralized manner to establish a baseline or reference point. Then, we will attempt to design a local or individual performance metric that can be evaluated by each robot but also leads to the desired collaborative behavior, which will allow us to perform the learning in a distributed manner.

The collaborative task chosen for this study is a loosely-coordinated collective movement or flocking [58]–[64], in which a set of robots move together as a group. Several researchers have applied different optimization techniques to improve the performance of manually designed flocking controllers, such as PSO [94], [95], Gradient Descent [96], Reinforcement Learning [97], [98], Genetic Algorithms [99], and Evolutionary Strategies [100]. Our approach in this chapter differs in that our behaviors are generated by a general feed-forward artificial neural network with a large number of parameters and not by a flocking-specific controller.

It should be noted that the task as implemented in this chapter is harder than those from previous work in that the robots are not physically connected to each other [50], they are required not only to aggregate but also move together [51], and there is no environmental template or goal to guide their movement [2].

We will tackle this task in two iterations with different perspectives. First, we will aim at

learning the task with the lowest possible requirements in terms of complexity of sensing, complexity of control architecture, and learning time. Then, we will relax these requirements, increasing the complexity of sensing, the complexity of the control architecture, and learning time in order to obtain an improved performance.

The remainder of this chapter is organized as follows. Section 9.1 describes the experimental methodology, including learning algorithms, performance metrics and controller used for the first iteration. In Section 9.2 we present and discuss the results obtained in the first iteration both in simulation and with real robots. Section 9.3 describes the controller and performance metrics used for the second iteration, and Section 9.4 the results obtained in it. Finally, in Section 9.5 we summarize our findings.

9.1 Experimental Methodology

As mentioned in the introduction, we will implement two different learning schemes, in relation to how the particles are distributed among the robots and how the fitness function is defined. The first, global homogeneous, copies the same candidate solution (or set of weights) to every robot, and uses a global fitness function that evaluates the group behavior. We will refer to this approach as *centralized*. The second, local heterogeneous, distributes a different candidate solution (or set of weights) to each robot, and uses a local fitness function that is evaluated independently and individually on each robot. We will refer to this approach as *distributed*. The distributed approach allows to speed up the evaluations by a factor equal to the number of robots, yet it makes the learning harder, especially when the local and global performance metrics are not trivially aligned (e.g., the global performance cannot be represented by a linear combination of local performances).

9.1.1 Performance Metrics

In this subsection, we will give the mathematical definition of the performance metrics used for centralized and distributed learning in the first iteration. The way inputs are measured during the experiments in simulation and reality is described in Subsection 9.1.3.

Both the local and global performance metrics used in Sections 9.1 and Sections 9.2 mimic in their components two of Reynolds' flocking rules [101]: avoiding collisions and attraction to neighboring flock-mates. The alignment or velocity matching rule is not directly reflected in the performance metric of the first iteration because our goal is to accomplish the task with the lowest possible requirements in terms of sensing, complexity of control architecture, and learning time. We will add the alignment rule in the second iteration (Sections 9.3 and 9.4), expecting to obtain increased performance at the cost of increased controller complexity, learning time, and more complex implementation on real robots, as the Khepera III does not have a compass to determine absolute headings and estimating relative heading using the relative positioning board requires additional communication at each controller time step.

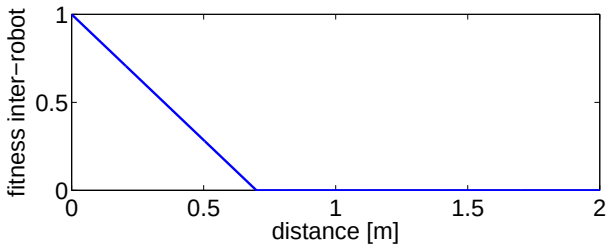


Figure 9.1: Inter-robot fitness as a function of the distance between two robots.

Both global and local performance metrics have three factors: movement, compactness, and collision avoidance. These factors reward robots that move as far as possible from their initial positions, stay close to each other, and avoid collisions between them. The factors are all normalized to the interval [0, 1].

The movement factor of the global performance metric (f_{1g}) is the normalized distance between the initial and the final positions of the center of mass of the group of robots. The normalization factor is the maximum distance that a robot can travel in one evaluation, i.e., the robot's maximum speed multiplied by the evaluation time.

$$f_{1g} = \frac{|\vec{x}_c(t_f) - \vec{x}_c(t_0)|}{D_{max}} \tag{9.1}$$

The global compactness factor (f_{2g}) is the average over the evaluation time and over each pair of robots of the inter-robot fitness. We define the inter-robot fitness between two robots as a function of the distance between them, as shown in Fig. 9.1. The fitness is maximum at 0 m, and it is zero when the robots are further apart than 0.7 m.

At each time step, we calculate the inter-robot fitness for each pair of robots, and then average across all pairs to calculate the global compactness factor as given by Equation 9.2:

$$f_{2g} = \frac{1}{N_e} \sum_{k=1}^{N_e} \left(\frac{1}{N_{pairs}} \sum_{j=1}^{N_{pairs}} fit\_inter_{j,k} \right) \tag{9.2}$$

where N_e is the number of time steps in the evaluation period, N_{pairs} is number of inter-robot pairs and $fit\_inter_{j,k}$ is the inter-robot fitness for inter-robot pair j at time step k.

The global collision-avoidance factor (f_{3g}) is the average for every robot and over the evaluation time of the maximum value of the proximity sensors at each time step:

$$f_{3g} = \frac{1}{N_{rob}} \sum_{j=1}^{N_{rob}} \left(\frac{1}{N_e} \sum_{k=1}^{N_e} i_{max,j,k} \right) \tag{9.3}$$

where $i_{max,j,k}$ is the normalized proximity sensor activation value of the most active sensor at time step k for robot j, and N_{rob} is the number of robots.

The local performance metric is calculated individually by each robot, using exclusively on-board resources. The local movement factor (f_{1l}) is the normalized distance travelled by the robot, based on the final position, which is calculated with odometry using the wheel encoders.

$$f_{1l} = \frac{|\vec{x}_i(t_f) - \vec{x}_i(t_0)|}{D_{max}} \tag{9.4}$$

The local compactness factor (f_{2l}) is also based on the inter-robot fitness as defined in Figure 9.1 and used in Equation 9.2. However, in the local case the number of pairs N_{pairs} in Equation 9.2 is modified so that each robot only measures the distance to the other three using an on-board range and bearing module, and then averages the inter-robot fitness only for those other three robots, as opposed to averaging across all pairs of robots. Another difference worth noting between the local and global compactness factors is that the local inter-robot distance measurements are affected by occlusion, while the global ones are not.

Finally, the local collision-avoidance factor (f_{3l}) is the single robot version of the global factor:

$$f_{3l} = \frac{1}{N_e} \sum_{k=1}^{N_e} i_{max,j,k} \tag{9.5}$$

Both global and local fitness are obtained by aggregating the three corresponding factors using the generalized aggregation functions described by Zhang et al. [68]:

$$F = \left(\frac{\omega_1 f_1^s + \omega_2 f_2^s + \omega_3 f_3^s}{\omega_1 + \omega_2 + \omega_3} \right)^{\frac{1}{s}} \tag{9.6}$$

where f_i are the individual fitness factors (with $f_i = f_{il}$ for the local fitness, and $f_i = f_{ig}$ for the global), ω_i their corresponding aggregation weights, and s is the degree of compensation or trade-off strategy (higher s means that a high value for a certain factor can compensate for lower values in the others).

For all experiments in this article we set $s = 0$, i.e., the highest degree of compensation in design-appropriate aggregation functions, simplifying Equation 9.6 to:

$$F = (f_1^{\omega_1} f_2^{\omega_2} f_3^{\omega_3})^{\frac{1}{\omega_1+\omega_2+\omega_3}} \tag{9.7}$$

This equation is obtained by taking the limit of s approaching to zero in Equation 9.6 (see [67] for the derivation).

Since the three factors (f_i) are in the interval [0, 1], the fitness function F will also be in the same range. The different combinations of aggregation weights explored in this chapter are as follows: $\{\omega_1 = 0.25, \omega_2 = 0.5, \omega_3 = 0.25\}$, $\{1/3, 1/3, 1/3\}$, and $\{0.1, 0.8, 0.1\}$.

In Chapter 5 we showed that the fitness evaluations for learning a simpler robotic task such as obstacle avoidance had a large standard deviation, and that performing re-evaluations was an effective way of dealing with this challenge in the learning. Given the more complex behavior to be learned in this article and the difficulties encountered while doing so, we decided to perform multiple internal evaluations of the fitness and average them in order to make the learning more robust. Concretely, each candidate solution is evaluated four times during 45 s and its performance averaged before consideration by the learning algorithm ($F' = \frac{1}{4}\sum_{i=1}^{4} F_i$).

9.1.2 Control Architecture

The controller is a feed-forward artificial neural network of two units which uses only local, on-board measurements regardless of the performance metric. Its inputs are the range and bearing measurements and the infrared proximity sensors, and it outputs the two wheel speeds.

Instead of using the Khepera III's nine infra-red proximity sensors as inputs, the neural network inputs use four virtual sensors ir_k (front-left, front-right, back-left and back-right) obtained from averaging in pairs and normalizing the sensor values of eight sensors and discarding the central sensor in the back part. This grouping allows us to reduce the number of weight parameters while still being able to detect and avoid obstacles, as seen in Chapter 7.

The eight range and bearing inputs rb_k are obtained by dividing the bearing into eight sectors, and calculating the activation of each sector by taking the minimum range value measured in that sector and dividing it by the maximum possible range, which is 3.3 meters.

The diagram of the resulting controller is shown in Figure 9.2. Each neuron has 13 input connections: 4 corresponding to the infrared proximity sensors, 8 corresponding to the range and bearing sensor, and one constant bias speed, resulting in 26 weight parameters (w_k) in total. The outputs of the neurons define the wheel speeds $\{v_l, v_r\}$ as given by Equations 9.8 and 9.9. $s(\cdot)$ represents the sigmoidal activation function.

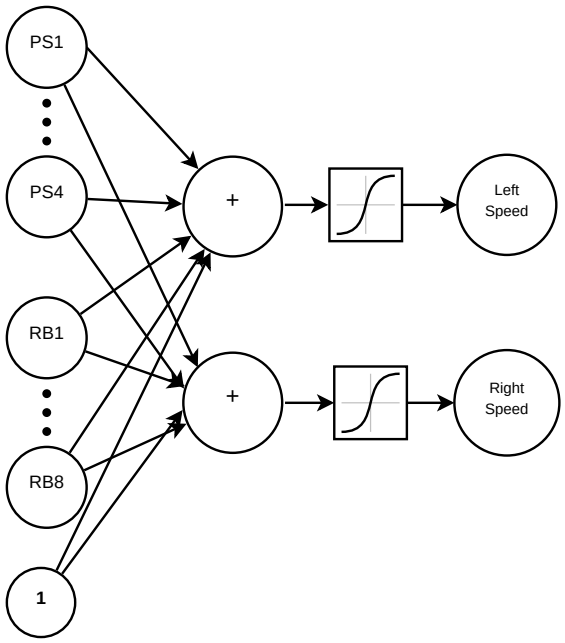


Figure 9.2: Artificial Neural Network controller. *PS1* to *PS4* represent four virtual IR proximity sensors obtained by sensor grouping, and *RB1* to *RB8* are the eight inputs from the relative positioning board.

$$v_l = s(w_1 + \sum_{k=1}^{4} i_k \cdot w_{k+1} + \sum_{k=1}^{8} rb_k \cdot w_{k+5}) \tag{9.8}$$

$$v_r = s(w_{14} + \sum_{k=1}^{4} i_k \cdot w_{k+14} + \sum_{k=1}^{8} rb_k \cdot w_{k+18}) \tag{9.9}$$

9.1.3 Experiments

The learning process is performed completely in simulation. The learning algorithm is *PSO pbest*, the noise-resistant variation introduced by Pugh et al. [7] described in Subsection 3.6.2.

The PSO neighborhood presents a ring topology with one neighbor on each side. Particles' positions and velocities are initialized randomly with a uniform distribution in the $[-20, 20]$ interval, and their maximum velocity is also limited to that interval. The PSO algorithmic parameters are set following the guidelines for limited-time adaptation we presented in Chapter 5 and are shown in Table 9.1. These guidelines recommend a population size equal to the dimension of the search space. Since the dimension of the search space is 26 and four robots are used, we round up to 28 particles in order to have exactly seven particles per robot in the distributed implementation. The evaluation time is set to 45 s to allow the robots to

Table 9.1: PSO Parameter Values for the First Approach

Parameter	Value
Number of robots N_{rob}	4
Population size N_p	28
Iterations N_i	50
Evaluation span t_e	4x45 s
Re-evaluations N_{re}	1
Personal weight w_p	2.0
Neighborhood weight w_n	2.0
Dimension D	26
Inertia w_I	0.8
Initial range X_{init}	20

spread in a measurable amount if compactness is not enforced by the controller.

Each evaluation during the learning process takes place in an unbounded arena. Four robots are placed forming a square of side length equal to two robot diameters with random orientations. In order to calculate the local fitness function, robots only use their internal measurements (simulated range and bearing sensor, infra-red proximity sensors and wheel encoders, all with added noise). The global fitness function is calculated using the robots' global positions provided by the simulator, which have no errors or noise.

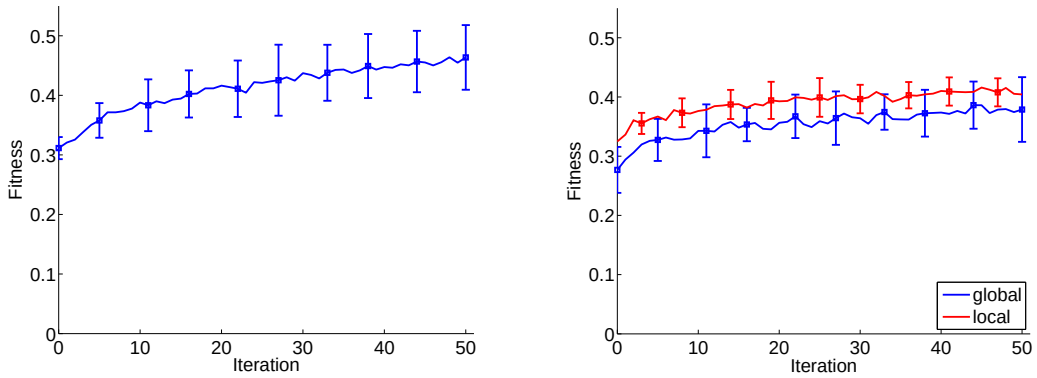
We perform 20 optimization runs for global homogeneous (centralized) learning and another 20 runs for local heterogeneous (distributed) learning for statistical significance. After the learning process is finished, the performance of the best solution in the population from each of the 20 learning runs is evaluated systematically in simulation, running 20 experiments of 45 s for each solution.

It is worth noticing that in the distributed learning, groups of four particles are always evaluated together as a flock of four robots. Therefore, when testing the controller from distributed learning, we find the particle with the best local performance and test it together with the other three particles of its group.

Based on the results of these tests in simulation, the best solution for global homogeneous learning and the best solution for local heterogeneous learning are selected for systematic tests with real robots. We run 20 experiments for each solution.

In these experiments, the global positions are monitored using an overhead camera connected to a computer running SwisTrack [53] (see Figure 3.2a). The initial positions and number of robots are the same as used for learning in simulation, but the evaluation time is reduced to 10 s in order to be able to keep track of the robots' positions during the whole evaluation due to the limited field of view of the fixed overhead camera and the ideally unbounded arena.

Following the same scheme as done in simulation, the local fitness function is computed on



(a) (b)

Figure 9.3: Learning progress measured using (a) the global metric for global homogeneous (centralized) learning and (b) the global (blue) and local (red) metrics for local heterogeneous (distributed) learning.

each robot using only its on-board resources, while the global fitness is computed externally given the information provided by the overhead camera and complemented with the local measures for the avoidance factor obtained from the robots.

The two selected best controllers are also re-evaluated in simulation using the reduced time of 10 *s* in order to perform valid quantitative comparisons and validate our models.

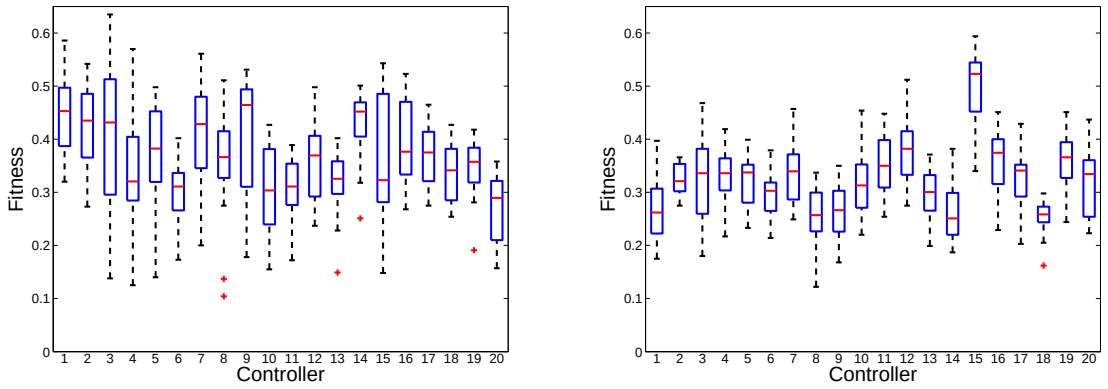
9.2 Results from First Iteration

We begin by presenting the results from the learning in simulation. Figure 9.3 shows the progress of the best solution found at each iteration for the two different learning approaches. The curves show the average of the 20 runs, and the error bars represent one standard deviation.

Comparing Figure 9.3b with Figure 9.3a we can notice that the performance of the local heterogeneous learning measured with the global metric is not as high as the global homogeneous one measured with the same metric. However, it should be noted that in homogeneous learning each iteration uses four times the number of evaluations as heterogeneous learning, since in homogeneous learning each candidate solution must be copied to all robots while in heterogeneous learning each robot tests a different candidate solution at the same time.

In addition to the global metric, Figure 9.3b shows the progress of the local performance metric for local heterogeneous learning. The global and local metrics are correlated, in the sense that learning with the local one leads to an improvement in the global one.

After the learning is finished, each of the 20 solutions found in the learning runs is tested in simulation for 20 evaluations of 45 seconds. Figure 9.4 shows the performance measured



(a) (b)

Figure 9.4: Performance measured with the global metric in simulation for the 20 solutions found with (a) global homogeneous (centralized) learning and (b) local heterogeneous (distributed) learning.

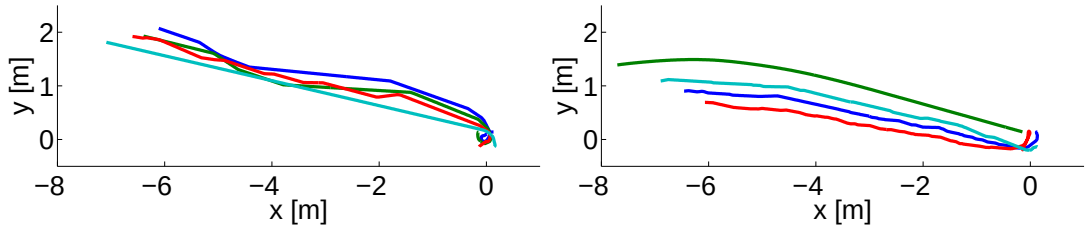
using the global metric obtained in this testing. The solutions from homogeneous learning outperform the ones from heterogeneous learning on average. However, the best solution from heterogeneous learning (number 15 in Figure 9.4b) has the highest performance over all.

All solutions present a high variation between evaluations (note that in this case the boxplots represent the variation in the performance of each best solution during the 20 evaluation runs, which is different from the variation in the learning shown in Figure 9.3). This variation between individual evaluations implies that the controllers are sensitive to the initial conditions, i.e. the initial random orientations of the robots.

The initial orientations affect the time it takes for the robots to find a common direction of movement, and therefore the total distance that the center of mass is able to travel in the 45 seconds. When robots fail to find a common direction of movement, they either aggregate close to their initial positions in a very compact group or split in smaller groups and go in separate directions.

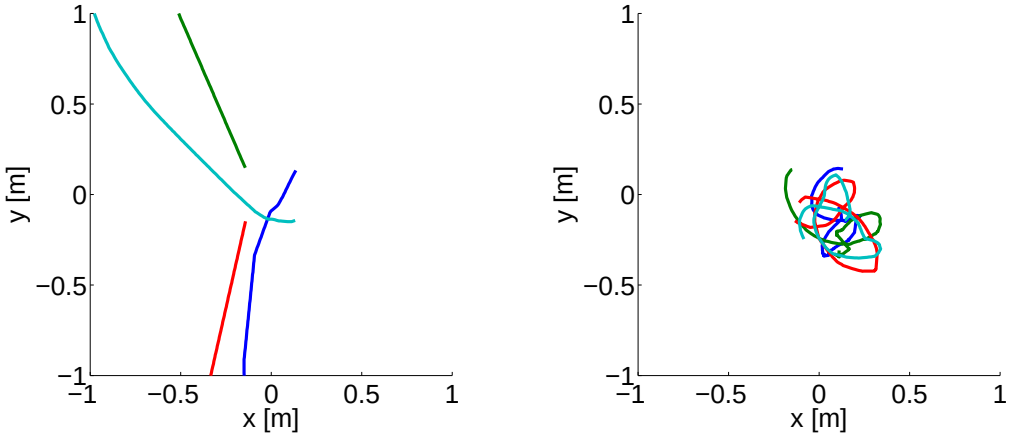
Figure 9.5 shows two example trajectories where a common direction of movement was found relatively quickly, allowing the robots to travel a large distance while staying close to each other.

It should be noted that the weights used in the fitness aggregation function also have a significant effect on the behavior of the resulting controllers. Before choosing the final values of $\{0.25, 0.5, 0.25\}$ for movement, compactness, and avoidance respectively, preliminary tests were conducted in simulation with two other set of weights: $\{1/3, 1/3, 1/3\}$ and $\{0.1, 0.8, 0.1\}$. Figure 9.6 shows the effect of these fitness aggregation weights on the resulting behaviors. Figure 9.6a has a low compactness weight, causing the robots to spread out, while Figure 9.6b has a high compactness weight, causing robots to stay together without moving far. In order



(a) (b)

Figure 9.5: Example of trajectories from simulation for successful runs with (a) local heterogeneous and (b) global homogeneous learning.



(a) (b)

Figure 9.6: Example of trajectories for different fitness aggregation weights: (a) $\{1/3, 1/3, 1/3\}$ and (b) $\{0.1, 0.8, 0.1\}$, for compactness, movement, and avoidance, respectively.

to keep the plots clear and avoid clutter, only the initial 10 s of the trajectories are shown.

Based on the results from the tests in simulation, the solutions with the highest medians were chosen to be tested on real robots (number 9 in Figure 9.4a, homogeneous learning, and number 15 in Figure 9.4b, heterogeneous learning). We conducted 20 evaluation runs of 10 seconds for each solution, both in simulation and reality. The evaluation time was reduced to 10 seconds to keep the robots in the overhead camera's field of view.

The quantitative performance measured in these evaluations is presented in Figure 9.7. Overall, both controllers showed a satisfactory performance when tested in reality, even though there are differences between simulation and reality that are statistically significant according to the Mann-Whitney U test (5% significance level). The fact that the heterogeneous solution has higher performance in simulation than in reality while the homogeneous shows the inverse implies that there are unmodeled factors that affect the two controllers in a different way due to their different behaviors.

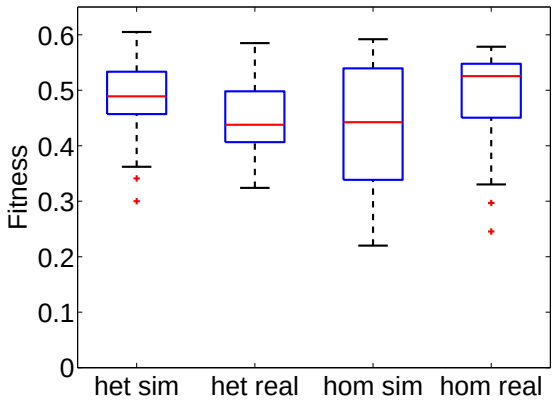
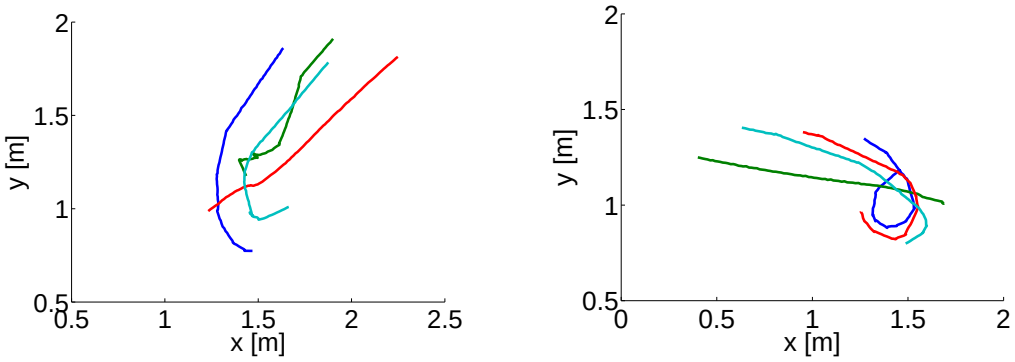


Figure 9.7: Performance measured with the global metric in simulation and reality for the best controllers found with global homogeneous and local heterogeneous learning.



(a) (b)

Figure 9.8: Example of trajectories for successful runs with (a) local heterogeneous and (b) global homogeneous learning tested on the real robots.

Qualitatively, the behaviors observed in reality were similar to the ones obtained in simulation. Figure 9.8 depicts two example trajectories from these 10 s evaluations. Note that the trajectories shown here for real robots last 10 s, representing only a fraction of the 45 s from those in Figure 9.5, but the initial steps are very similar.

During the real robot experiments, it became evident that for the heterogeneous solution the robot in front of the group was always the same, while the other three robots followed, meaning that heterogeneous learning led to specialized roles. On the other hand, for the homogeneous solution, the robot in front changed every time based on the initial random orientations.

9.3 Improving Flocking Performance

In the previous section we have shown that it is feasible to learn a collaborative task such as flocking in a fully distributed way using local, on-board, noisy sensors. However, even though the best solutions achieved a high performance, others did not perform as well, i.e., there was a noticeable variation among solutions.

Our goal in the last two sections of this chapter is to make the solutions for the coordinated motion task more consistent and robust. In order to achieve this, we will increase the complexity of the controller, hoping to obtain an increase in performance as seen in Chapter 7. We will add a hidden layer to the neural network, based on a well-known result that states that feedforward neural networks with a single internal, hidden layer and sigmoidal nonlinearities are universal approximators, that is, they can uniformly approximate arbitrary functions [69], [70]. We will also add the relative orientation to other robots as an additional input to the neural network, which will require additional communication at every time step to estimate relative headings to neighbors, as the Khepera III does not have a compass to determine its heading. Finally, given that we now have measurements of relative headings to neighbors, we will introduce a velocity alignment term in the local and global learning metrics, enforcing the third Reynolds' rule [101] that was missing in the previous section.

The improvements presented in these two sections were led by Dr. Iñaki Navarro and published in [102]. Dr. Navarro contributed with his expertise in flocking and in particular his experience with flocking metrics and alignment, while I contributed with the PSO algorithmic aspects, and assisted him in the experiments and in interpreting the resulting data.

The following subsections will describe the modifications in the controller, metric, and parameters from the first approach.

9.3.1 Control Architecture

The first modification to the controller architecture is the addition of a new input, which corresponds to the average of the headings among all the neighboring robots, in the robot's own coordinate system and normalized to the interval [-1,1]. The use of a single averaged input instead of one input per robot allows the controller to generalize to any number of robots.

The second modification is the addition of a hidden layer consisting of four hidden units with sigmoidal activation functions. The hidden layer, not present in the previous section, increases the generalization capabilites of the neural network [69], [70].

The third modification is the addition of recurrent and lateral connections in the two output units. And the final modification is the removal of the inputs from the proximity sensors, as the only obstacles present in the arena are the other robots and the distance to them can be measured with the range and bearing inputs.

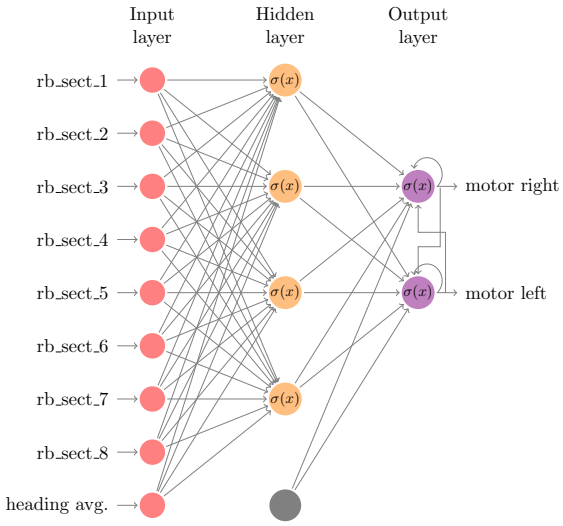


Figure 9.9: Diagram of the neural network controller. In red are the inputs, yellow the hidden layer with sigmoidal outputs, in blue the sigmoidal outputs which control the motor speed, and in gray the bias input.

Table 9.2: PSO Parameter Values for the Second Approach

Parameter	Value
Dimension D	50
Population Size N_p	52
Iterations N_i	200

Therefore, the new controller, shown in Figure 9.9, has nine input units, a hidden layer of four units, and two output units. The total number of parameters is 50 (36 from the inputs to the hidden layer, eight from the hidden layer to the output neurons, four from the recurrent connections, and two from the bias speeds).

9.3.2 Algorithmic Parameters

Given the increased complexity of the controller, we need to allocate more time for the learning process. In particular, since the dimension of the problem is now 50 and four robots are used, we round up the population size to 52 particles in order to have exactly 13 particles per robot in the distributed implementation (as opposed to the 28 used previously). Also, the number of iterations are increased from 50 to 200. These modifications, shown in Table 9.2, result in an increase in the total evaluation time of approximately 7.4 times when compared to the first approach.

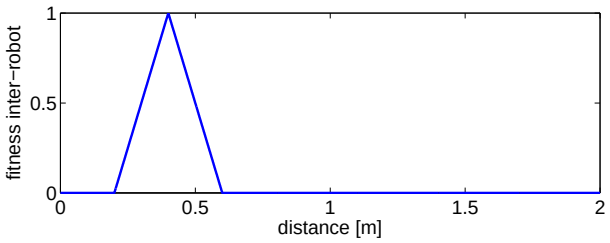


Figure 9.10: Inter-robot fitness as a function of the distance between two robots.

9.3.3 Performance Metrics

The new local and global performance metrics have three factors as in the previous case. However, the old avoidance factor will be integrated into compactness, which will now penalize robots that are too close or too far, and we will add the alignment factor. Therefore, the three new factors are: movement, compactness, and alignment.

The movement factor of the global performance metric (f_{1g}) is the same as in the previous case, the normalized distance between the initial and the final positions of the center of mass of the group of robots defined in Equation 9.1.

The new global alignment factor (f_{2g}) quantifies the heading difference between two robots (H_{diff}) averaged between every pair of robots and during the evaluation time. It has a maximum value of 1 when all the robots are aligned and tends to 0 when robots are not aligned. It is defined as:

$$f_{2g} = 1 - \frac{1}{N_e} \sum_{k=1}^{N_e} (\frac{1}{N_{pairs}} \sum_{j=1}^{N_{pairs}} abs(H_{diff_{j,k}})/\pi) \tag{9.10}$$

where N_e is the number of time steps in the evaluation period, N_{pairs} is number of inter-robot pairs and $H_{diff_{j,k}}$ is the difference of heading between pair j at time step k. Note that if there are more than two robots its value can never be 0.

As in the previous case, the new global compactness factor (f_{3g}) is based on the inter-robot fitness between pairs of robots. However, we redefine the inter-robot fitness to now penalize both robots that are too close and too far, as shown in Figure 9.10.

The fitness is maximum at the desired inter-robot distance of 0.4 m, and it is zero when the robots are closer than 0.2 m (slightly larger than the robots' diameter) or further apart than 0.6 m. Therefore, it aggregates two of Reynolds' rules: collision avoidance and flock centering (staying close to flockmates). The global compactness factor is obtained by calculating the new inter-robot fitness for each pair of robots, and then averaging accross all pairs in the same manner as in the old case, given by Equation 9.2

The new local movement factor (f_{1l}) is the normalized distance traveled by the center of mass

of the group of robots, calculated using the odometry of the robot and the relative position to neighboring robots, and given by Equation 9.11.

$$f_{1l} = \frac{|\vec{x}_c(t_f) - \vec{x}_c(t_0)|}{D_{max}} \tag{9.11}$$

If a neighboring robot position can not be estimated (due to occlusions or limited range of the relative positioning system), the last absolute position where the robot was seen is used as final position. This new local movement factor differs from the previous one in that it takes into account the distance traveled by the center of mass and not only by one robot. Therefore, it matches the global movement factor better than the previous one, but tends to evaluate all the particles in the group of robots with a very similar performance, even though the controllers could be very different.

The new local alignment factor (f_{2l}) is calculated in the same manner as the global alignment factor, with Equation 9.10. However, the difference here is that each robot calculates its own metric only by measuring the heading difference between itself and the other three robots, using the relative positioning system and communication. These measurements might be affected by occlusions and range limitations. If for a time step no neighbor is seen then H_{diff} is set to 1 for that time instant.

The new local compactness factor (f_{3l}) is based on the new inter-robot fitness shown in Figure 9.10. However, when averaging the inter-robot fitness between pairs of robots, the number of pairs N_{pairs} in Equation 9.2 is modified so that each robot only measures the distance to the other three using the relative positioning system and then averages the inter-robot fitness only for those other three robots, as opposed to averaging across all pairs of robots. Another difference between the local and global compactness factors is that the local inter-robot distance measurements are affected by occlusion, while the global ones are not.

Both global and local fitness are obtained by aggregating the three corresponding factors using the generalized aggregation function given by Equation 9.7. The aggregation weights used are: $\omega_1 = 0.4$, $\omega_2 = 0.5$, and $\omega_3 = 0.1$.

9.4 Results from Second Iteration

We conducted the same experiments as described in Subsection 9.1.3 with the new controller and metrics.

The progress from learning in simulation using both the centralized and the distributed approach is shown in Figure 9.11. The curves show the average over the 20 runs of the performance of the best solution found at each iteration, and the error bars represent one standard deviation. In the case of distributed learning it shows not only the local metric, which

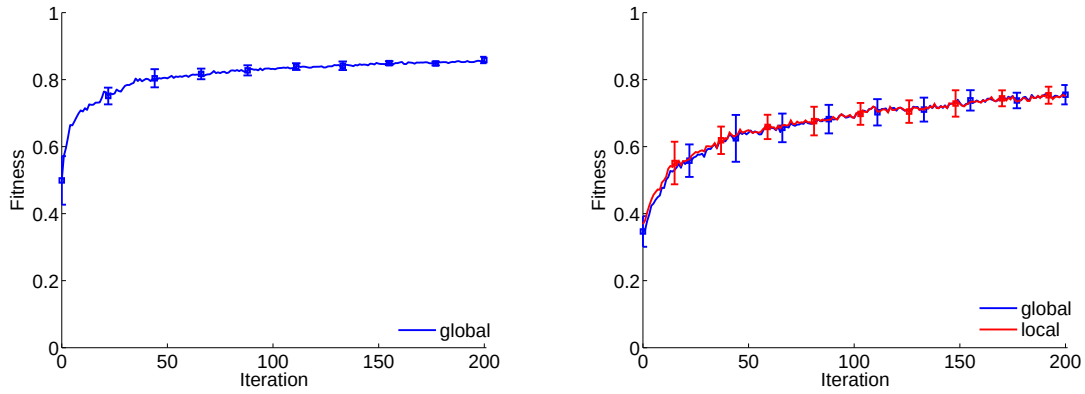


Figure 9.11: (a) Learning progress measured using the global metric for global homogeneous (centralized) learning . Learning progress measured using the global (blue) and local (red) metrics for local heterogeneous (distributed) learning. The curves show the average of the 20 runs, and the error bars represent the standard deviation.

is the one used for the learning, but also the global one, since it is designed to reflect the quality of the flocking behavior and allows for comparison with centralized learning.

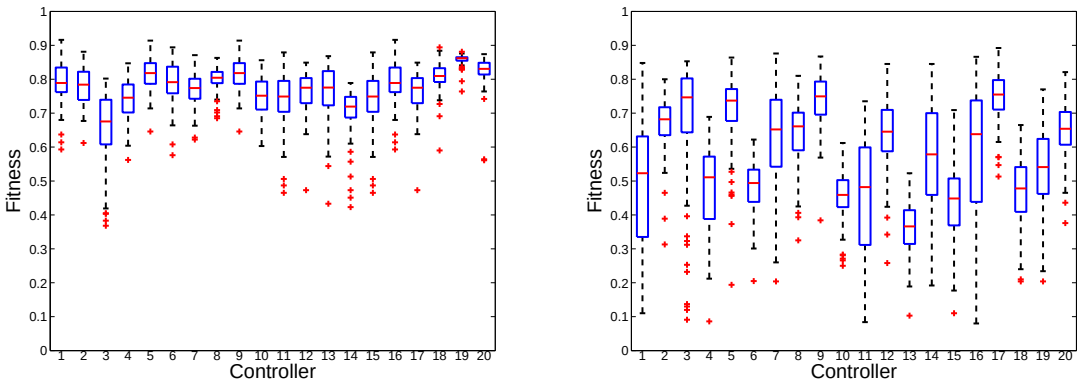
Comparing Figure 9.11a with Figure 9.11b, we can see that global homogeneous learning achieves a better global metric performance and lower standard deviation. Also, it requires less iterations to learn as the learning curve becomes flatter faster, although the homogeneous approach employs four times the evaluation time of the heterogeneous approach for each iteration.

In Figure 9.11b there is a perfect matching of local and global metrics, not just correlation as was the case for the previous metrics shown in Figure 9.3b.

After the learning process is finished, the fitness of the best solution from each of the 20 independent learning runs is evaluated systematically in simulation, running 100 experiments of 60 s for each solution.

From Figure 9.12a we can see that centralized learning achieves a high performance with low standard deviation for all the runs. Its performance is considerably more consistent than in the case of the centralized learning from the first iteration (Figure 9.4a).

The distributed learning shown in Figure 9.12b achieves the desired behavior in most runs, but sometimes fail, resulting in a high standard deviation. We observed that this happens when robots aggregate close to their initial positions in a very compact group and fail to travel far, resulting in low performance. This might be caused by obtaining very similar evaluations of the four particles tested together, which does not reflect the differences among the four controllers, discarding potential good solutions and promoting bad ones. In spite of these



(a) (b)

Figure 9.12: Performance measured with the global metric in simulation for the best solution found in each of the 20 independent learning runs with (a) global homogeneous (centralized) learning and (b) local heterogeneous (distributed) learning. The box represents the upper and lower quartiles, the line across the middle marks the median, and the crosses show outliers for 100 evaluations of each controller.

failures, there are more successful runs than in the case of the distributed learning from the first iteration (Figure 9.4b).

In order to validate the results obtained in simulation, we select the controller with highest median for each learning approach and test it on real robots. We conducted 20 evaluation runs of 10 seconds for each solution, both in simulation and reality. The evaluation time was reduced to 10 seconds to keep the robots in the overhead camera's field of view.

Figure 9.13 shows the performance for the best controller from each learning approach both in simulation and reality. Both in simulation and reality the homogeneous controller achieves the highest median. There is a higher variance in the results with real robots that suggests that some modeling details are missing in the simulation.

9.5 Chapter Summary

In this chapter we have applied the distributed PSO framework to learn flocking, a collaborative task. For this purpose, we based ourselves on the insights gained in the analysis of non-collaborative behaviors performed in previous chapters of this part. In particular, we have applied the guidelines to select the PSO parameters proposed in Chapter 5, used re-evaluations to deal with noise, and used sensor grouping to reduce the parameter space (Chapter 7).

Our results have shown that it is feasible to learn a collaborative task in a fully distributed way with a local performance metric measured using local, on-board, noisy sensors. On average, the performance of the solutions found with the distributed approach measured with the global metric was not as high as the ones from centralized learning. This difference was not

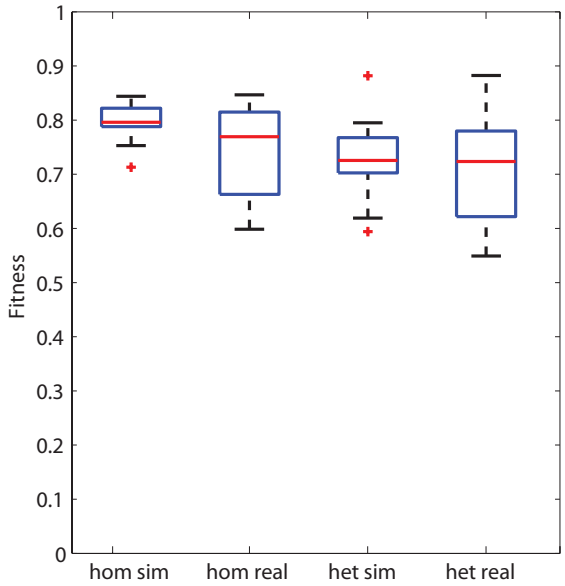


Figure 9.13: Performance measured with the global metric in simulation and reality for the best controllers found with global homogeneous and local heterogeneous learning.

only due to the metric chosen for learning but also to the increased difficulty in coordination arising from heterogeneous controllers. However, the best solutions found for centralized and distributed learning performed similarly, both in simulation and in experiments with real robots. Additionally, the best solution from distributed learning exhibited specialized roles in which one robot consistently led the group while the others followed.

We have also seen that regardless of the learning method, the coordinated motion task was very sensitive to the initial configuration of the robots, and therefore the performance evaluations were noisy. We addressed this issue in the learning by using different initial orientations for each evaluation and averaging their performances.

In addition, we saw how the performance on the task could improved by adding the relative orientation to other robots as an additional input, adding a hidden layer to the neural controller, and introducing a velocity alignment term in the performance metric. This improvement came at the cost of more learning time required due to the larger parameter space in the more complex controller.

Results from this chapter were presented at the International Symposium on Experimental Robotics [103], and in a publication with Iñaki Navarro as first author at the European Conference on Artificial Life [102].

10 Challenges Arising from Noisy Evaluations

There are several sources of randomness that make performance evaluations of robotic controllers inherently noisy. In addition to sensor and actuator noise, other factors that can contribute to the uncertainty in performance evaluations are varying initial conditions, manufacturing tolerances, and changes in the environment. In the previous chapters, we have seen examples of noisy performance measurements both in the obstacle avoidance task as well as the flocking task.

Small amounts of noise may actually be desirable in order to facilitate the transition between simulation and reality, and to obtain more general, robust controllers [104]. However, large amounts of noise may disrupt the learning process, causing stagnation and even leading to divergence [105].

In Chapter 2 we have introduced several previous studies on the performance of population-based learning techniques under noise, for algorithms such as PSO [33], [34], [105], Genetic Algorithms [30], and Evolutionary Strategies [29], [31], [32]. However, most of these studies were conducted on benchmark functions, usually with an additive Gaussian noise model. In this chapter we would like to test whether the results derived from tests on the benchmark functions can be extended to the noisy performance evaluations encountered in multi-robot learning.

In order to achieve this goal, first we are going to revisit the fitness distributions found in the multi-robot obstacle avoidance benchmark and analyze their impact on the learning process in Section 10.1. Then, we will attempt to model the effects of noise found on the robotic case study in two numerical benchmark functions with added noise, and discuss the differences with previous results on benchmark functions (Section 10.2). Finally, Section 10.3 concludes the chapter with a summary of our findings.

Table 10.1: PSO Parameter Values for Obstacle Avoidance

Parameter	*PSO std*	*PSO pbest*
Number of robots N_{rob}	4	4
Population size N_p	24	24
Iterations N_i	40	20
Evaluation span t_e	30 s	30 s
Re-evaluations N_{re}	0	1
Personal weight w_p	2.0	2.0
Neighborhood weight w_n	2.0	2.0
Dimension D	24	24
Inertia w_I	0.8	0.8
Initial range X_{init}	20	20

10.1 Robotic Learning

One of the aims of this chapter is to understand and analyze the randomness in robotic performance evaluations and how it affects the learning process. We will use two different opimization algorithms: standard PSO [4], which we will denote *PSO std*, and the noise-resistant variation *PSO pbest* introduced by Pugh et al. [7]. Both algorithms are described in detail in Chapter 3.

The PSO algorithmic parameters for the obstacle avoidance benchmark task are set following the guidelines for limited-time adaptation we presented in Chapter 5 and are shown in Table 10.1. The differences between *PSO std* and *PSO pbest* are the number of re-evaluations N_{re}, which is 0 for *PSO std* and 1 for *PSO pbest*, and the number of iterations N_i, which is 40 for *PSO std* and 20 for *PSO pbest*. The purpose of conducting half the number of iterations for *PSO pbest* is to have the same total number of evaluations as *PSO std*, since each iteration of *PSO pbest* requires twice as many function evaluations due to the re-evaluations of the personal best.

We conduct experiments in two different environments. The first one is an empty square arena of 2 m x 2 m, where the walls and the other robots are the only obstacles. The second environment is the same bounded arena with 15 cylindrical obstacles added (diameter 10 cm). The obstacles are randomly repositioned before each fitness evaluation, meaning that the second environment is not only more complex but also variable from evaluation to evaluation, and so more noisy. The initial robots positions are set randomly with a uniform probability distribution, verifying that they do not overlap with obstacles or other robots. All experiments are conducted with 4 Khepera III robots in simulation.

The controller used is the recurrent artificial neural *ann24* with 24 weight parameters described in Section 4.3 of Chapter 4. All other experimental details also follow the explanation of the obstacle avoidance benchmark given in that chapter.

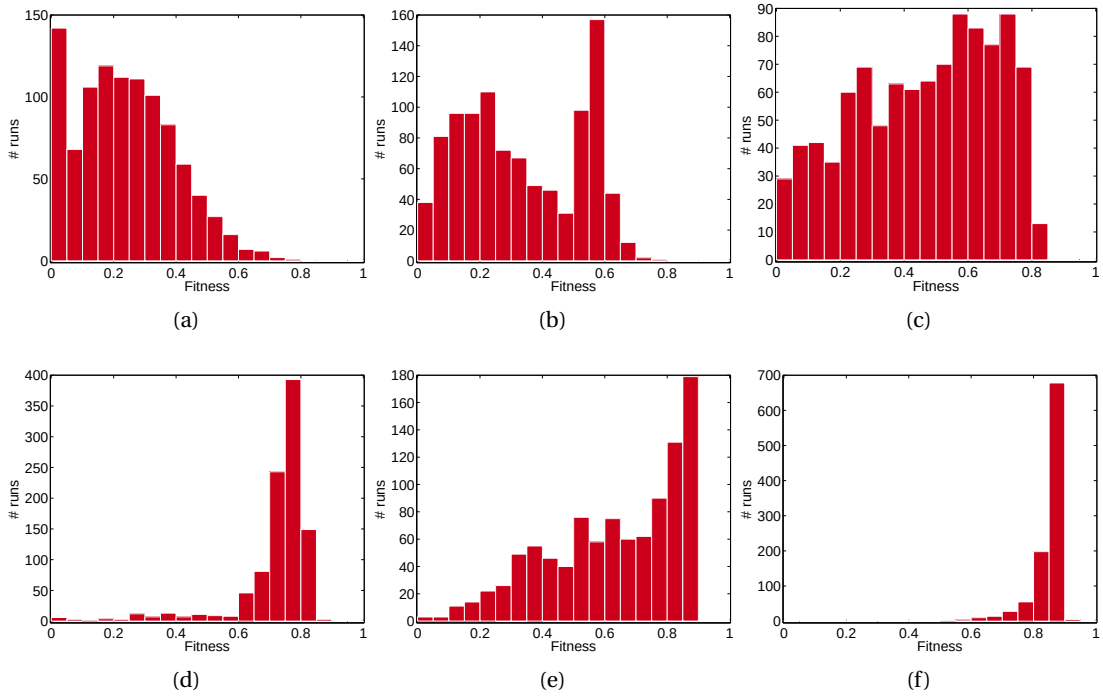


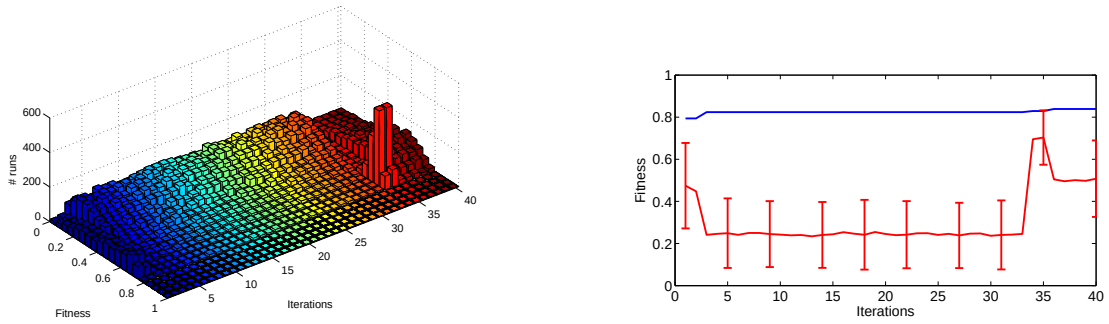
Figure 10.1: Distributions of 1000 a posteriori evaluations of the fitness of the global best solution at different iterations corresponding to two *PSO pbest* runs, one in each environment. p value of the one-sample Kolmogorov-Smirnov test in parenthesis. (a) Iteration 1 in arena with obstacles ($p = 0.000891$). (b) Iteration 3 in arena with obstacles ($p < 10^{-6}$). (c) Iteration 11 in arena with obstacles ($p = 0.000017$). (d) Iteration 20 in arena with obstacles ($p < 10^{-6}$). (e) Iteration 16 in empty arena ($p < 10^{-6}$). (f) Iteration 19 in empty arena ($p < 10^{-6}$).

10.1.1 Fitness Distributions

A major challenge in comparing robotic learning algorithms that is not present in benchmark functions is that it is not possible to separate the deterministic and random components of the fitness evaluations, i.e., there is no single true fitness value for a given position. This implies that a single evaluation does not provide sufficient information on the goodness of a particular solution.

Therefore, in order to test the outcome of a given optimization technique, we characterize each candidate solution by repeatedly evaluating the fitness a large number of times and look at the probabilistic distribution of those evaluations. In particular, for the results presented in this section, we do 1000 a posteriori evaluations of the candidate solutions given by the optimization algorithm.

Figure 10.1 shows examples of the distributions obtained for global best positions at several iterations of *PSO pbest* in the two aforementioned environments (with and without obstacles). The distributions shown in Figures 10.1a, 10.1b, 10.1c, and 10.1e are not gaussian according to



(a) (b)

Figure 10.2: Evolution of the global best fitness for a single run of *PSO std* in the environment with obstacles. (a) The distributions of 1000 a posteriori evaluations of global best fitness. (b) In blue, the global best value calculated by standard PSO algorithm. In red, the average of global best fitness over 1000 a posteriori evaluations (vertical bars indicate the standard deviation).

a one-sample Kolmogorov-Smirnov statistical test. This test was performed as well for every distribution of the *PSO pbest* learning in the two environments with negative results.

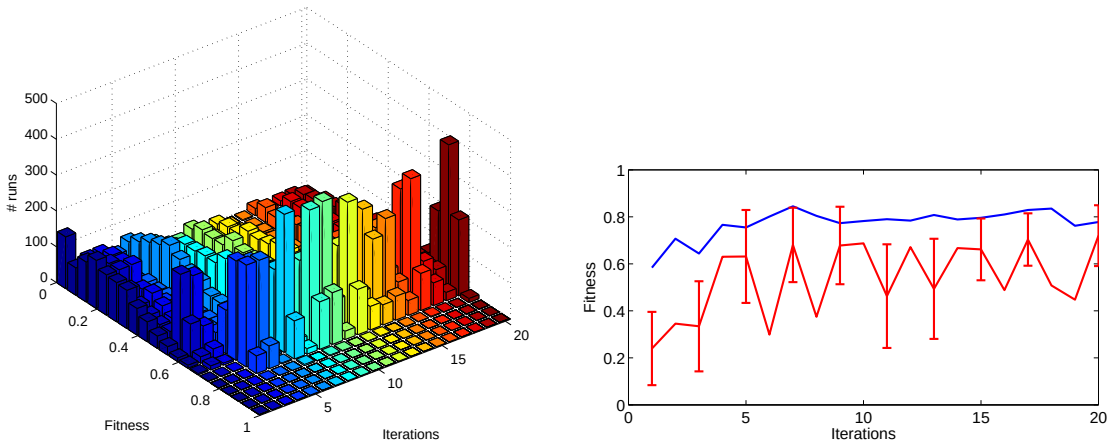
Looking at every distribution of the fitness of the solutions obtained by *PSO pbest* in the two environments we see that those corresponding to the empty environment have lower standard deviations (ranges from 0.039 to 0.226, averaging 0.099 over all tested distributions) than those in the environment with obstacles (ranges from 0.115 to 0.221, averaging 0.175 over all tested distributions). This is something that could be expected from the experimental setup given the randomized placement of obstacles and robots in each evaluation.

It is important to mention that the distributions shown here and found in our analysis can not be generalized directly to the whole search space, since they only correspond to the positions explored by the PSO algorithm.

10.1.2 Learning with Noise

Figure 10.2a shows the distributions of the global best solutions (re-evaluated 1000 times) for each iteration during a *PSO std* run. In Figure 10.2b, we plot the average value of these distributions in red, and the standard deviation with vertical bars. The global best value obtained by *PSO std* during the optimization is shown in blue. The first thing we can notice is that *PSO std* consistently over-estimates the average of the 1000 a posteriori evaluations. In addition, we can see how *PSO std* is not able to learn in a proper way, since while the global best reported by PSO is monotonically increasing, the average of the 1000 a posteriori evaluations stays stable for a long period (stagnation) or even decreases sometimes.

In order to better understand this effect, we can look back to Figure 10.2a to see that the distribution at iteration 3 has a large standard deviation. What happened is that the given



(a) (b)

Figure 10.3: Evolution of the global best fitness for a single run of *PSO pbest* in the environment with obstacles. (a) The distributions of 1000 a posteriori evaluations of global best fitness. (b) In blue, the global best value calculated by the *PSO pbest* algorithm. In red, the average of global best fitness over 1000 a posteriori evaluations (vertical bars indicate the standard deviation).

position was evaluated by PSO with a high value (0.824), becoming the global best and staying as such until iteration 34. Since *PSO std* does not re-evaluate the personal best positions this global best candidate was never filtered out.

A similar analysis can be done for *PSO pbest* by looking at Figure 10.3. It can be observed how, due to the re-evaluation of the personal bests, there is less difference between the global best calculated by *PSO pbest* (in blue) and its a posteriori estimation (in red) than in standard PSO. Therefore, the estimated fitness of the global bests (Figure 10.3b in red), increases during the learning process, which implies that the overall quality of the solutions is improving.

Comparing the distributions of the global best solutions learned with *PSO std* and *PSO pbest* (Figures 10.2a and 10.3a respectively) we notice that in the early iterations they both have a similar noise profile, since they correspond to the same environment with obstacles and the initial learning conditions are the same for both algorithms. However, for later iterations the distributions of *PSO pbest* tend to show a peak at high fitness values, which imply better solutions.

Nevertheless, even for *PSO pbest* there are cases where good solutions are replaced by bad ones (see for example iteration 6 and iteration 19 in both Figures 10.3a and 10.3b). Even though the algorithm is very fast to recover from the drop, this could mean a very poor final solution if the learning is stopped at that time.

Table 10.2: PSO Parameter Values for Obstacle Avoidance

Parameter	*PSO std*	*PSO pbest*
Population size N_p	24	24
Iterations N_i	4000	2000
Re-evaluations N_{re}	0	1
Personal weight w_p	2.0	2.0
Neighborhood weight w_n	2.0	2.0
Dimension D	24	24
Inertia w_I	0.8	0.8
Initial range X_{init}	5.12	5.12

10.2 Benchmark Functions

We perform PSO runs on two of the standard benchmark functions mentioned in Section 3.4: the sphere function (f_1 in Equation 3.1) and Rosenbrock's function (f_2 in Equation 3.2).

In order to reproduce the effects encountered in robotic learning, we normalize the function values to the interval $[0,1]$ by dividing the original functions by the maximum value of each function within the initial position range $[-X_{init}, X_{init}] = [-5.12,\ 5.12]$, which we denote by $\max f_i$. We then add noise from two distributions: a Gaussian distribution with zero mean and standard deviation σ (Equation 10.1), and a Bernoulli distribution with probability p and amplitude A (Equation 10.2).

$$f_i^g(\mathbf{x}) = \frac{f_i(\mathbf{x})}{\max f_i} + \mathcal{N}(0,\sigma) \tag{10.1}$$

$$f_i^b(\mathbf{x}) = \frac{f_i(\mathbf{x})}{\max f_i} + A \cdot \mathcal{B}(p) \tag{10.2}$$

If after adding noise the function values are greater than one or less than zero, they are set to one and zero respectively to maintain the fitness bounded in the interval $[0,1]$.

The parameters for PSO on the benchmark functions are mostly the same as the ones used for robotic learning in order to obtain the same algorithmic behavior. The main differences are the number of iterations $N_i = 4000$, which is usually higher in benchmark functions, and the initial range $X_{init} = 5.12$ which is usually lower [7]. Another exception is the absence of two parameters that are specific to the robotic case-study and are therefore omitted in the benchmark functions: number of robots and evaluation span (one benchmark function evaluation is assumed to be equivalent to the evaluation of a controller for the whole evaluation span). The parameters used on the benchmark functions are shown in Table 10.2.

As it was the case for robotic learning, we perform half the number of iterations for *PSO pbest* than for *PSO std*. The purpose is again to maintain the total number of evaluations equal and compare both algorithms fairly, since each iteration of *PSO pbest* requires twice as many function evaluations as *PSO std* due to the re-evaluations of the personal bests. In addition, we define a *step* of an algorithm to be equal to one iteration of PSO, and half an iteration of PSOavg, so that a fixed number of steps represents the same number of function evaluations for both algorithms. This allows us to plot the performance of both algorithms as a function of *steps* on the same graph in order to do direct comparisons with the same total number of function evaluations.

A significant difference between benchmark functions and robotic learning is that it is possible to remove the noise from the benchmark functions to see the real performance of the algorithm. Therefore, all results reported in this section show the fitness function values obtained when evaluating the functions without noise (there is no need of 1000 a posteriori evaluations since the fitness is obtained in a deterministic way). Another difference that should be considered when comparing results is that benchmark functions are minimized as opposed to maximized.

10.2.1 Gaussian Distribution

The purpose of the tests with added Gaussian noise is to study the effect of large variances relative to the initial fitness values in the optimization process. We used four increasing values of the standard deviation: $\sigma = \{0, 0.01, 0.05, 0.1\}$.

Figure 10.4 shows the progress of the learning on benchmark function f_1. For low levels of noise, the algorithm makes progress for a large number of steps. However, for the levels of noise observed in the experiments with robots, the optimization process quickly stagnates, and increasing the number of steps does not improve the performance further.

This effect is not mentioned in previous works on benchmark functions with added noise because the standard deviation values used are much lower than the ones considered in this chapter. For example, on the unnormalized sphere function (without dividing by the maximum value as described in the beginning of this section), an unnormalized variance of 1.0 might affect the final stages of the optimization process when the fitness values become small, but is not significant in the initial phases where the values of the function are in the order of $D \cdot x_{init}^2$. However, when the fitness values are normalized to $[0, 1]$ and $\sigma = 0.1$, the noise is also disruptive in the initial stages of the learning. This might explain why the number of iterations used in the robotic learning literature is considerably lower than the ones used on numeric benchmark functions (order of hundreds versus order of thousands).

We observe a similar behavior in the normalized benchmark function f_2 (see Figure 10.5), which suggests that this effect is not particular to an individual fitness function but mainly caused by the amount of noise added.

Under the high-amplitude noise conditions observed in these experiments with $\sigma = 0.05$ and

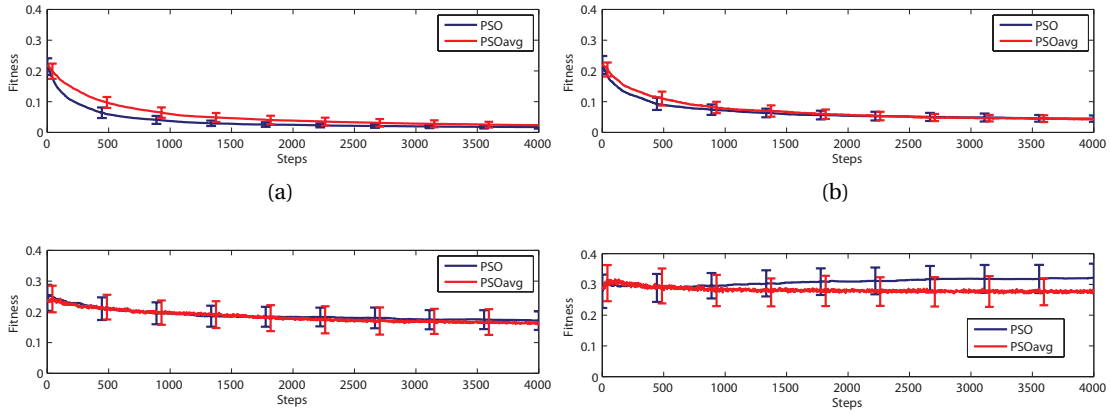


Figure 10.4: Fitness of best solution at each step on function f_1 with added Gaussian noise for *PSO std* and *PSO pbest*. A step is equal to one iteration of *PSO std*, and half an iteration of *PSO pbest*. (a) $\sigma = 0$. (b) $\sigma = 0.01$. (c) $\sigma = 0.05$. (d) $\sigma = 0.1$.

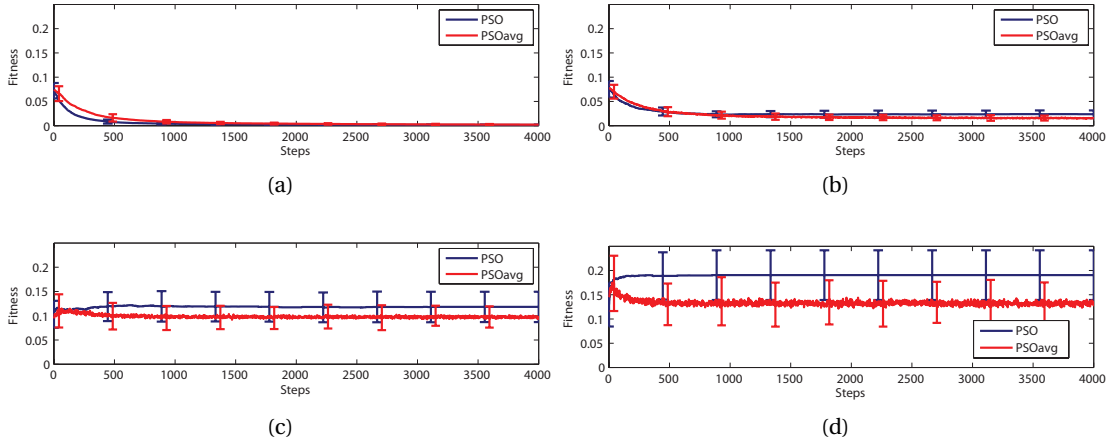
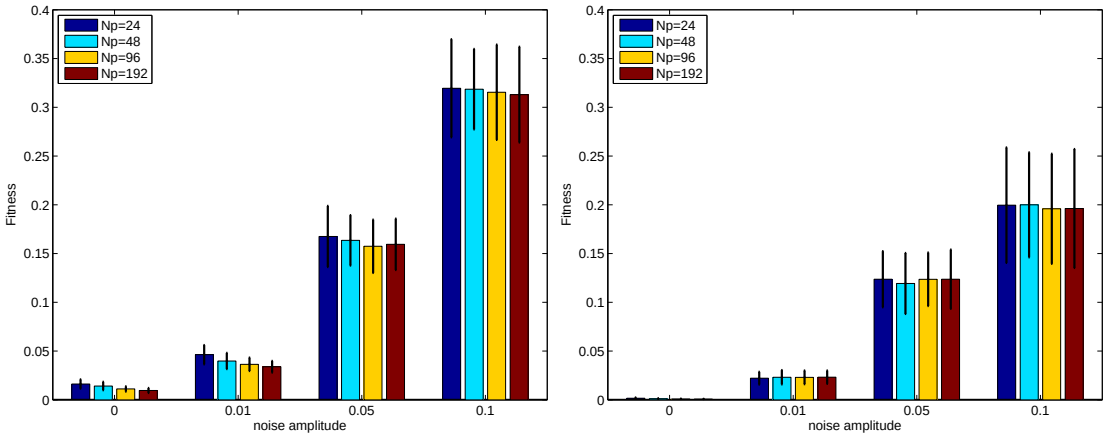


Figure 10.5: Fitness of best solution at each step on function f_2 with added Gaussian noise for *PSO std* and *PSO pbest*. A step is equal to one iteration of *PSO std*, and half an iteration of *PSO pbest*. (a) $\sigma = 0$. (b) $\sigma = 0.01$. (c) $\sigma = 0.05$. (d) $\sigma = 0.1$.



(a) (b)

Figure 10.6: Final fitness on functions f_1 and f_2 with added Gaussian noise for increasing population sizes. (a) Function f_1. (b) Function f_2.

$\sigma = 0.1$, *PSO pbest* significantly outperforms *PSO std*.

In the Genetic Algorithms literature, increasing the population size is often mentioned as an effective technique to deal with noise[30], [31], due to the fact that a large population is more likely to have similar solutions, and the effect of noise in fitness evaluations is likely to be compensated across them. In their survey of evolutionary optimization in uncertain environments [29], Jin and Branke refer to this approach as implicit averaging, as opposed to the explicit averaging that consists in sampling each candidate solution's fitness a fixed number of times, and then averaging all samples.

In order to test whether the implicit averaging approach works on PSO under the high-amplitude noise conditions described previously, we ran *PSO std* on benchmark functions f_1 and f_2 with added Gaussian noise and increased the population size from 24 to $\{48, 96, 192\}$ while holding the other parameters constant (i.e., the larger population sizes require a larger number of total function evaluations).

Figure 10.6 shows the final fitness obtained after 4000 iterations for both functions and Figure 10.7 shows the progress on function f_1. It can be seen from Figure 10.6 that increasing the population size achieves better fitness for low amounts of noise, but it does not improve the performance for high-amplitude noise, even though the total number of functions evaluations is much higher. Figure 10.7 shows that even though the population size was increased eight times the algorithm failed to make progress for $\sigma = 0.1$.

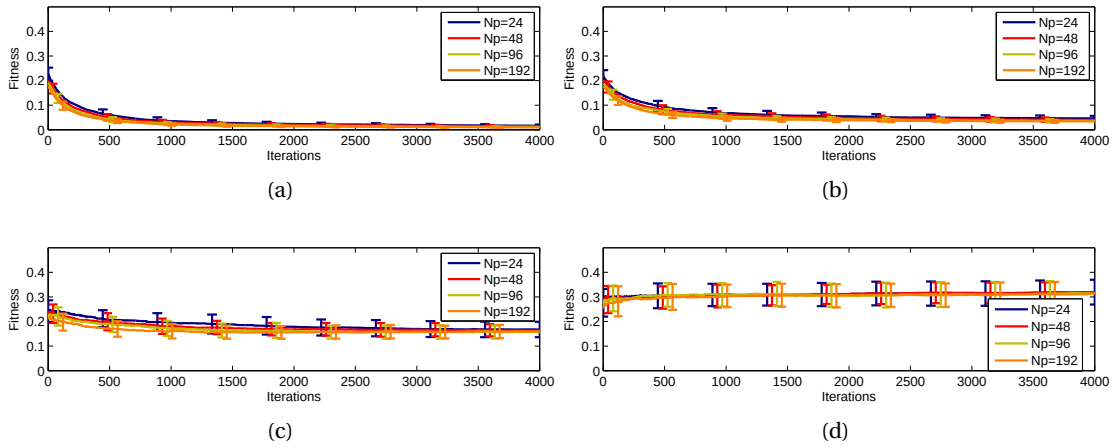


Figure 10.7: Fitness of best solution at each iteration on function f_1 with added Gaussian noise for increasing population sizes. (a) $\sigma = 0$. (b) $\sigma = 0.01$. (c) $\sigma = 0.05$. (d) $\sigma = 0.1$.

10.2.2 Bernoulli Distribution

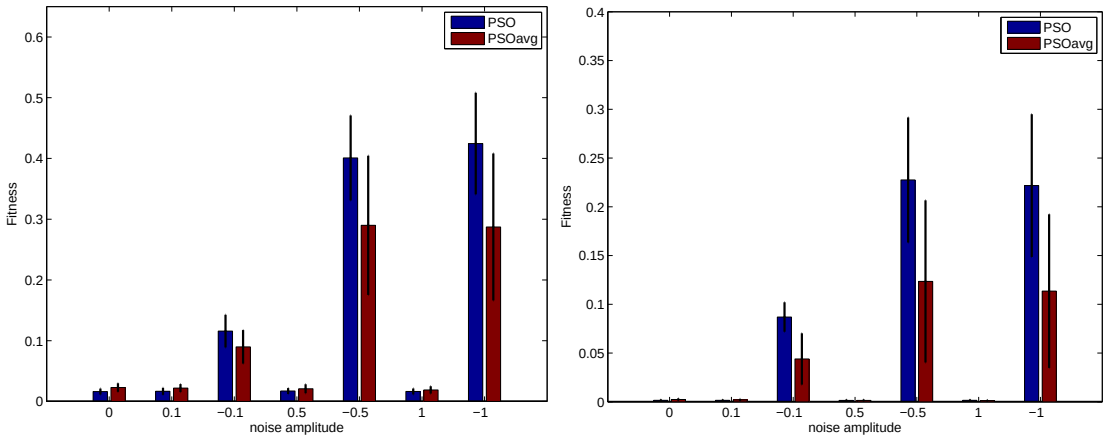
We employ the Bernoulli distribution to study the effect of skewed noise with positive and negative amplitudes. This is a simplified model of both type of outliers that we observed in robotic learning: bad evaluations of good solutions (e.g., hardware failures), and good evaluations of bad solutions (e.g., unusually advantageous initial conditions).

The variance σ^2 of a Bernoulli distribution with probability p and amplitude A is given by:

$$\sigma^2 = A^2 p(1-p) \tag{10.3}$$

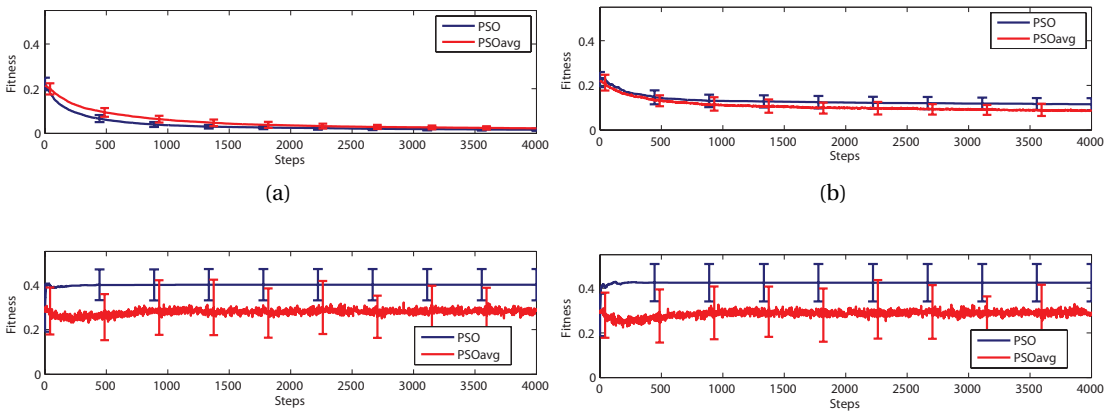
We set $p = 0.01$ and $A = \{0, \pm 0.1, \pm 0.5, \pm 1\}$ in order to obtain the same standard deviation $\sigma = \{0, 0.01, 0.05, 0.1\}$ as used in the experiments with Gaussian Noise. Figure 10.8 shows the final fitness obtained after 4000 steps on both benchmark functions. In all cases, negative amplitudes perform significantly worse than positive amplitudes of the same magnitude. This means that there is a non-symmetric effect of the noise: good spurious evaluations of bad solutions (noise with large negative values in minimization problems) are worse than bad evaluations of good solutions. *PSO pbest* helps to reduce this effect by discarding bad solutions that had a high fortuitous evaluation through the re-evaluations of the personal best.

Figure 10.9 shows the progress of the optimization on benchmark function f_1 with added Bernoulli noise of several amplitudes and $p = 0.01$. The algorithm fails to make progress in high-noise settings, as we have shown with the Gaussian noise distribution in both benchmark functions (Figures 10.4 and 10.5). We also conducted tests on benchmark function f_2 with added Bernoulli noise and observed the same behavior (graphs not shown).



(a) (b)

Figure 10.8: Final fitness on functions f_1 and f_2 with added Bernoulli noise of positive and negative amplitudes ($p = 0.01$) for *PSO std* and *PSO pbest*. (a) Function f_1. (b) Function f_2.



(a) (b)

(c) (d)

Figure 10.9: Fitness of best solution at each step on function f_1 with added Bernoulli noise ($p = 0.01$) for *PSO std* and *PSO pbest*. A step is equal to one iteration of *PSO std*, and half an iteration of *PSO pbest*. (a) $A = 0$. (b) $A = -0.01$. (c) $A = -0.5$. (d) $A = -1$.

10.3 Chapter Summary

In this chapter we analyzed the problem of noise in multi-robot learning, and contrasted it with that of benchmark functions. We have shown that fitness evaluations in multi-robot learning have a large-amplitude noise that is disruptive to the initial phases of the learning process of PSO. We were able to reproduce this behavior on standard benchmark functions by normalizing the fitness values and adding Gaussian noise with a large standard deviation relative to the fitness values obtained at the beginning of the learning process.

We have also modeled two kind of outliers that we observed in multi-robot learning with a Bernoulli distribution using positive and negative amplitudes. We showed that PSO is more sensitive to good spurious evaluations of bad solutions than bad evaluations of good solutions.

Under these conditions, neither increasing the population size nor increasing the number of iterations were able to improve the performance of the learning. On the other hand, we have seen that re-evaluations led to an improvement in performance and are therefore an alternative to deal with noise in multi-robot settings.

Results from this chapter were presented at the IEEE Congress on Evolutionary Computation [106].

11 Improving Performance under Noise: PSO OCBA

In the previous chapter, we have analyzed the performance of *PSO std* and *PSO pbest* under noise and identified issues such as early stagnation and overestimation of the goodness of solutions. This chapter will present a distributed noise-resistant PSO algorithm for multi-robot learning that addresses those issues. It is based on Optimal Computing Budget Allocation (OCBA), a statistical sample allocation method introduced by Chen et al. [107].

The idea behind the application of OCBA to PSO is to improve the noise-handling capabilities of previous PSO algorithms. For instance, standard PSO has no explicit mechanism for dealing with noise, and implicit averaging, which consists in increasing the population size, is not effective for the large noise amplitudes encountered in multi-robot learning, as we saw in Chapter 10. The naïve approach of evaluating every new candidate a fixed number of times results in a better performance estimation for new candidates, but invests as many resources in good candidates as in poor ones which could be immediately discarded [105]. Another disadvantage of this method is that the number of repetitions of each evaluation is fixed and should be selected based on the amount of noise, which must be known in advance. The noise-resistant approach of *PSO pbest* that evaluates best candidates multiple times [6] has the advantage of placing more computation on the most promising solutions and therefore achieves a high performance, but it is sensitive to "lucky" good evaluations of bad new solutions, which might displace a consistently better old solution, generating random performance drops during the learning [106].

OCBA automatically adjusts the evaluation budget between old and new solutions to maximize the probability of correct selection of good candidates. In addition, as the iterations increase, good candidates tend to accumulate a large number of samples, thereby producing accurate performance estimates of the best solutions, and leaving a larger proportion of the allocation budget to accurately test new candidates.

This chapter is organized as follows. First, we provide some background on OCBA in Section 11.1 in order to facilitate subsequent explanations. Then, Section 11.2 describes the algorithms incorporating OCBA to PSO for multi-robot implementations and their differences

with the ones from the literature. In Section 11.3, we give details on the experiments on the obstacle avoidance task that will be used to compare the algorithms and controllers. Section 11.4 presents the results from applying the different algorithms for learning in simulation. In Section 11.5, we show the results from real robot experiments, and compare the learned controller to one obtained using potential fields [108], a common approach in mobile platforms with limited sensing. Finally, Section 11.6 summarizes the contributions of this chapter.

11.1 Background on OCBA

Optimal Computing Budget Allocation (OCBA) is a technique based on Bayesian statistics for allocating samples to different candidate solutions introduced by Chen et al. [107]. Given k candidates with means $\{\bar{X}_1,\ldots,\bar{X}_k\}$ and variances $\{\sigma_1^2,\ldots,\sigma_k^2\}$, and a total number of samples T, OCBA aims at maximizing the probability of correct selection $P\{CS\}$ of candidate b as the best (in a minimization problem, the one with the lowest mean):

$$P\{CS\} = P\{\bar{X}_B < \bar{X}_i, i \neq b\} \tag{11.1}$$

by applying the following allocation rules:

$$\frac{N_i}{N_j} = \left(\frac{\sigma_i/\delta_{b,i}}{\sigma_j/\delta_{b,j}}\right)^2, i \neq j \neq b \tag{11.2}$$

$$N_b = \sigma_b\sqrt{\sum_{i=1,i\neq b}^{k}\frac{N_i^2}{\sigma_i^2}} \tag{11.3}$$

where N_i is the number of samples for candidate i, and $\delta_{i,j} = \bar{X}_i - \bar{X}_j$ the difference between the means of candidate i and candidate j. An intuitive way of interpreting Equations 11.2 and 11.3 is that candidate i will get more samples N_i when it has larger variance σ_i^2 and when its mean is closer to the mean of the best solution found so far (small $\delta_{b,i}^2$). To switch the type of problem from minimization to maximization, we can simply consider $\bar{X}_i = -\bar{X}_i'$ where $\bar{X}_i'$ corresponds to the mean of the maximization problem.

This allocation procedure has been proven to be optimal in the sense that it maximizes an asymptotic approximation to the probability of correct selection $P\{CS\}$ as the number of samples tends to infinity, but it was also shown to be very efficient for limited sampling budgets in numerical experiments [107].

OCBA has previously been applied to PSO on numerical benchmark functions in a centralized manner [34], [105], [109], where it outperformed other techniques for dealing with noise. Here, we will first detail how we apply PSO OCBA in a centralized manner to the multi-robot obstacle avoidance benchmark task. Then, we will propose a new distributed version which requires only local information and communication, and compare it to other algorithms in the same task.

In addition to PSO, OCBA has also been applied to improve the performance under noise of evolutionary algorithms [110], [111]. Schmidt et al. [110] used OCBA on an evolutionary algorithm for ranking selection on noisy functions. They allocated as many evaluations as needed to obtain an arbitrary quality of the estimations. In [111], the authors used a version of OCBA to select a set of best potential solutions on each iteration of an Evolution Strategy (ES) algorithm. The difference between the ES OCBA and PSO OCBA is that in ES it is only needed to estimate the performance of the subset with the best solutions while PSO requires to estimate the performance of all candidate solutions.

11.2 Learning Algorithms

As a baseline for our work, we will use the following three algorithms previously presented in the PSO literature:

- *PSO std*, the standard PSO described in Figure 3.5, Section 3.6.
- *PSO rep*, the naïve approach of evaluating every new candidate a fixed number of times [105], in this case 10 (determined experimentally on preliminary runs),
- *PSO pbest*, the noise-resistant variant by Pugh et al. [6] which re-evaluates personal bests at each iteration.

Table 11.1 shows the parameters that are common to all PSO algorithms used in this paper. They are set following the guidelines for limited-time adaptation presented in Chapter 5. Table 11.2 shows the parameters that vary between the three previously mentioned variants, and serves as a quick summary of the differences between them.

In order to perform fair comparisons, the total number of evaluations for each algorithm was held constant, which implies that the number of iterations was set to be inversely proportional to the number of evaluations performed at each iteration. This is why 500 iterations of *PSO std* were performed, 50 for *PSO rep*, and 250 for *PSO pbest*.

We will now describe the PSO OCBA variants. The pseudocode for the centralized version of PSO OCBA, *PSO ocbaC*, is shown in Figure 11.1. The main difference from the standard PSO shown in Fig. 3.5 is the evaluation step, which now involves an allocation procedure using OCBA (Lines 3 to 12 in Figure 11.1).

Table 11.1: Parameters common to all PSO algorithms

Parameter	Value
Number of robots N_{rob}	4
Population size N_p	24
Evaluation span t_e	30 s
Personal weight w_p	2.0
Neighborhood weight w_n	2.0
Neighborhood size N_n	3
Dimension D	24
Inertia w_I	0.8
Initial range X_{init}	20

Table 11.2: Parameters for *PSO std*, *PSO rep*, and *PSO pbest*

Parameter	*PSO std*	*PSO rep*	*PSO pbest*
Evaluations of new candidates	1	10	1
Re-evaluations of pbests	0	0	1
Iterations N_i	500	50	250

```
1: Initialize particles
2: for N_i iterations do
3:    for N_p particles do
4:       Evaluate new particle position n_0 times
5:    end for
6:    remaining budget := iteration budget - n_0 · N_p
7:    while remaining budget> 0 do
8:       Allocate Δ samples among current positions and personal bests using OCBA
9:       Evaluate allocated samples
10:      Recalculate mean and variance for new evaluations
11:      remaining budget := remaining budget - Δ
12:   end while
13:   for N_p particles do
14:      Update personal best
15:      Update neighborhood best
16:      Update particle position
17:   end for
18: end for
```

Figure 11.1: Pseudocode for the *PSO ocbaC* algorithm.

Table 11.3: Parameters for *PSO ocbaC* and *PSO ocbaD*

Parameter	*PSO ocbaC*	*PSO ocbaD*
Iterations N_i	50	50
Iteration budget B_i	240	10
Initial number of samples n_0	2	2
Additional number of samples Δ	4	1

First, n_0 samples of the new positions are taken to estimate their mean and variance (in our case, $n_0 = 2$). Then the remaining samples are allocated among all the new positions and all the personal bests (48 candidates total) using Equations 11.2 and 11.3. Note that since all personal bests were new positions at some time, they already have at least n_0 samples at the moment of the OCBA allocation.

The parameters for *PSO ocbaC* are shown in Table 11.3. Again, the number of iterations was calculated to have the same total number of evaluations as the other algorithms, and in this case it is 50, the same value as *PSO rep* since they both perform 240 evaluations per iteration.

The pseudocode for the distributed version of PSO OCBA, *PSO ocbaD*, is shown in Figure 11.2. In this case, each particle is running its own algorithm, so the pseudocode is written from the point of view of an individual particle. First, the particle takes n_0 samples of its new position in order to estimate its mean and variance. Next, the particle collects the mean, variance, and number of samples of all candidates in the neighborhood (new positions and personal bests). In our case, for comparison purposes, we are using the same neighborhood size as Pugh et al. [6], which is one neighbor on each side of a ring topology. Then, the particle allocates the remaining budget among the shared new positions and personal bests in the neighborhood (in this case, 6 candidates in total: own position, own personal best, 2 shared new positions, and 2 shared personal bests) using OCBA Equations 11.2 and 11.3. Finally, the particle evaluates the candidates with the number of samples given by the OCBA allocation and shares the results in the neighborhood.

The parameters for *PSO ocbaD* are displayed alongside those for *PSO ocbaC* in Table 11.3. The main difference is that since each particle is performing its own OCBA allocation, the iteration budget B_i for that allocation is 1/24 the budget of the centralized version, and the additional number of samples Δ is reduced to 1 in order to share and receive the results from other particles after each evaluation.

The information shared by each particle is the mean, variance, and sample size of its current position and personal best position, which are required to compute the OCBA allocation. The mean, variance, and sample size can be calculated online incrementally every time a new

```
1: Initialize particle
2: for N_i iterations do
3:    Evaluate new particle position n_0 times
4:    Share evaluation results in neighborhood
5:    Receive and store evaluation results from neighborhood
6:    remaining budget := iteration budget - n_0 · N_p
7:    while remaining budget> 0 do
8:       Allocate Δ samples among current positions and personal bests in neighborhood
         using OCBA
9:       Evaluate allocated samples
10:      Recalculate mean and variance for new evaluations
11:      Share evaluation results in neighborhood
12:      Receive and store evaluation results from neighborhood
13:      remaining budget := remaining budget - Δ
14:   end while
15:   Update personal best
16:   Update neighborhood best
17:   Update particle position
18: end for
```

Figure 11.2: Pseudocode for the *PSO ocbaD* algorithm.

sample is added using the following equations:

$$\bar{X}_n = \frac{(n-1)\bar{X}_{n-1} + X_n}{n} \tag{11.4}$$

$$\sigma_n^2 = \frac{(n-2)}{(n-1)}\sigma_{n-1}^2 + \frac{(x_n - \bar{x}_{n-1})^2}{n} \tag{11.5}$$

Therefore, the history from previous evaluations can be incorporated without the need to store or share the entire vector of samples, which can become large towards the end of the learning, especially in the case of good solutions (e.g., we have observed several runs where the best solution had more than 100 samples at the end). Thus, the memory and communication requirements for the distributed algorithm are significantly reduced and they remain constant for the entire learning process.

11.3 Experimental Methodology

We conduct experiments in a square arena of 2 m x 2 m with walls, where a different number of cylindrical obstacles of diameter 10 cm are added (0 to 15 obstacles). In simulation, the obstacles are randomly repositioned before each fitness evaluation, and the initial robots'

Figure 11.3: Arena with 15 obstacles and four Khepera III robots performing one of the obstacle avoidance algorithms learned.

positions are set randomly with a uniform probability distribution, verifying that they do not overlap with obstacles or other robots. In real robot experiments, the obstacles are randomly positioned the first time, and then kept in this fixed position for the rest of the experiments, while the robots are manually repositioned in random locations between evaluations.

All experiments are conducted with four Khepera III robots. Robots can be seen in Figure 11.3 in the arena with 15 obstacles.

Following the methods described in Chapter 4, the controller used for the learning is *ann24*, the recurrent artificial neural network with 24 parameters described in Section 4.3, and the performance metric is given by Equation 4.1.

In addition to the learned controllers, in this chapter we employ a manually designed controller based on potential fields. This controller has two purposes: firstly, to show that the obstacle avoidance task in this cluttered environment and with this limited sensing range is challenging in terms of performance and noise, independently of the learning; and secondly, to provide a baseline performance value for comparisons with the learned controllers.

The total virtual force generated by the potential field controller is the weighted sum of a constant vector moving the robot forwards and repulsive proportional forces to obstacles measured by any of the infra-red proximity sensors. This resulting virtual force is translated into wheel speeds, and the different parameters are optimized manually for maximization of the performance metric. The performance metric for this controller is the same as used for the learning (Equation 4.1). The same metric is used in all experiments in this chapter. both in simulation and reality, to allow for direct comparisons.

Figure 11.4 shows the performance of this controller for 2000 runs in simulation for different obstacle densities. As we increase the number of obstacles in the arena from 0 to 15, the median performance decreases, and both the variance and the number of outliers increase. Outliers represent situations in which the robots get stuck, meaning that with 15 obstacles the avoidance task becomes quite challenging.

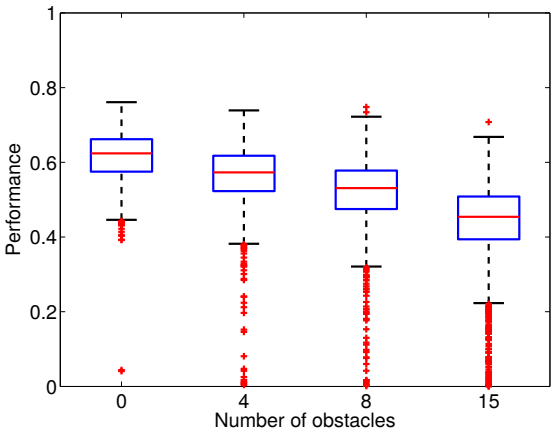


Figure 11.4: Performance of the potential field controller tested with an increasing number of obstacles in the arena. The box represents the upper and lower quartiles, the line across the middle marks the median, and the crosses show outliers.

11.4 Performance of Algorithms in Simulation

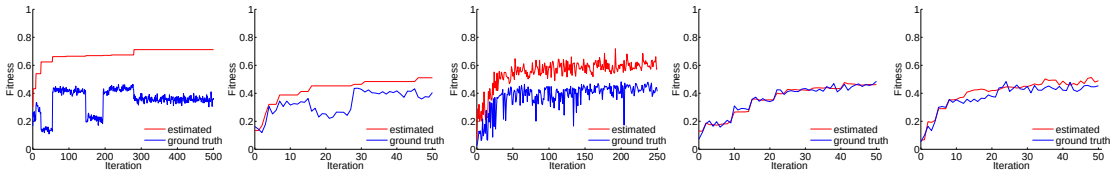
We compared the performance of the algorithms on the previously described obstacle avoidance benchmark with 15 obstacles. We chose 15 obstacles based on the results of Chapter 6 where we showed that learning in the most cluttered environment generalizes well to simpler ones. Due to the stochastic nature of PSO, we repeated each algorithm for 20 runs for statistical significance.

Figure 11.5 shows the progress of the learning for a single run of each of the five algorithms. We selected these runs because we considered them to be representative of the behavior of each algorithm, and they will allow us to discuss certain characteristics that cannot be appreciated in the aggregated results for all runs presented later in this section.

The red curve in Figure 11.5 is the performance of the best solution found so far as estimated by the algorithm and stored in its internal state. Due to the presence of noise, the fitness value of the best solution as reported by the algorithms may not be an accurate representation of the actual performance of the solution. Therefore, in order to accurately judge the performances for comparison purposes, during the learning we store the positions of the best solutions found at each iteration. After the learning is finished, we perform 100 a posteriori evaluations of each of the stored best solutions. We then calculate the mean of the 100 evaluations at each iteration and consider this as the “ground truth" performance of that solution (blue curve).

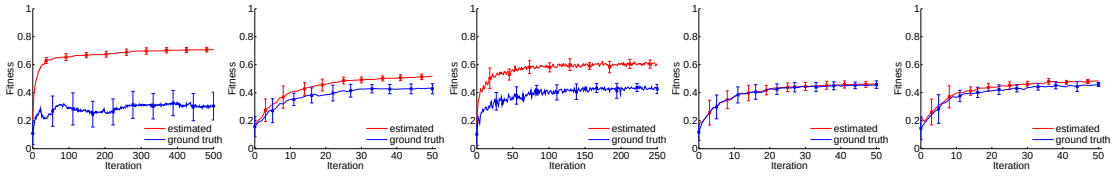
In general, the estimated performance is higher than the ground truth due to the fact that the learning tries to maximize the performance, and therefore evaluations with positive noise (values higher than the mean) are more likely to be selected than evaluations with negative noise.

We can see that the largest discrepancy between estimated and ground truth performances



(a) (b) (c) (d) (e)

Figure 11.5: Progress during a single run for each of the five algorithms. The red curve represents the performance of the best solution as estimated by the algorithm, and the blue curve represents the ground truth performance obtained as the average of 100 a posteriori evaluations. (a) PSO std. (b) PSO rep. (c) PSO pbest. (d) PSO ocbaC (e) PSO ocbaD.



(a) (b) (c) (d) (e)

Figure 11.6: Progress averaged over 20 runs for each of the five algorithms. The red curve represents the performance of the best solution as estimated by the algorithm, and the blue curve represents the ground truth performance obtained as the average of 100 a posteriori evaluations. Error bars represent one standard deviation (a) PSO std. (b) PSO rep. (c) PSO pbest. (d) PSO ocbaC (e) PSO ocbaD.

occurs for *PSO std* (Figure 11.5a), which bases its estimate on a single sample of each solution. In addition, even though the estimated performance for *PSO std* is monotonically increasing, the actual performance is rather erratic, with jumps and drops, and stagnates for several number of iterations. These effects are partially mitigated in *PSO rep* (Figure 11.5b) through the multiple evaluations, but jumps, drops, and stagnation still occur.

PSO pbest (Figure 11.5c) has a more sustained improvement, interrupted by sudden drops. These drops are due to the fact that *PSO pbest* does not re-evaluate new positions, and therefore a lucky evaluation of a bad new solution can displace a more consistent older candidate. Both *PSO ocbaC* (Figure 11.5d) and *PSO ocbaD* (Figure 11.5e) show much more consistent improvement along each iteration, with a good estimate of the ground truth performance and few drops or jumps.

Figure 11.6 shows the averaged results over the 20 runs for each of the five algorithms. In this case, the individual jumps and drops are averaged out as they occur at different iterations, but the difference between the estimated and ground truth performances becomes evident for the first three algorithms. It is also clear that even though the distributed implementation with a limited communication neighborhood of *PSO ocbaD* outperforms the first three algorithms in terms of estimating the ground truth, it does not perform as well as the centralized *PSO ocbaC*.

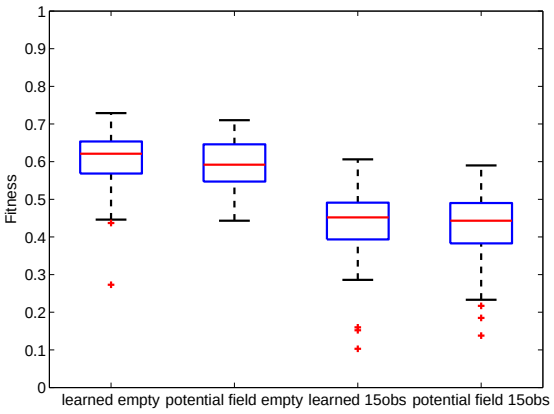


Figure 11.7: Potential field and learned controllers tested with 0 and 15 obstacles. The box represents the upper and lower quartiles, the line across the middle marks the median, and the crosses show outliers.

11.5 Experiments with Real Robots

In order to validate the results obtained from learning in simulation we tested the best controller from the *PSO ocbaD* approach and a traditional potential field controller in the two extremes of the benchmark task presented in Section 11.3: arena with no obstacles and arena with 15 obstacles. In the latter case, the obstacles were placed in random positions and then kept fixed for all the runs. We conducted 20 runs with 4 robots for each controller and environment, which results in 80 performance measurements per setting. The results from these experiments are shown in Figure 11.7.

Both controllers perform well in both environments and do not collide with walls, obstacles, or other robots. The median performance of the learned controller is slightly higher than the potential field one in both cases, although it is only statistically significant in the environment with no obstacles (Mann Whitney U test, $p = 0.02$ for no obstacles and $p = 0.80$ for 15 obstacles). This difference is due to the fact that when the potential field controller is approaching an obstacle or wall straight ahead, it gradually slows before turning, which results in lower speed factor and more time spent near obstacles (lower proximity factor). We also observed that the potential field controller oscillated in place or got stuck when there was a narrow passage between an obstacle and a wall.

On the other hand, the learned controller switched between two states: moving forwards at full speed and turning counter-clockwise in place at full speed. This behavior of switching between boundary points in the control space (*bang-bang control*) is similar to the behaviors seen when applying optimal control theory to minimize trajectory time for bounded velocity differential drive vehicles [112]. In addition, the fact that robots have a preferred turning direction (always turn counter-clockwise) avoids getting stuck, resulting in a more robust controller.

11.6 Chapter Summary

In this chapter, we have applied OCBA, a statistical technique for sampling budget allocation, to improve the performance of PSO in the presence of noise for the high-dimensional optimization of controllers for multiple robots with limited resources. In addition to the centralized budget allocation, we have introduced a distributed PSO OCBA algorithm suitable for resource-constrained mobile robots since it requires only local communication and sensing, and the amount of information shared is limited and constant for the whole learning process.

Results in an obstacle avoidance benchmark show that both PSO OCBA variants outperform other techniques for dealing with noise from the literature, achieving a more consistent progress and a better estimate of the ground-truth performance of candidate solutions.

In order to validate our simulations, we compared on real robots the controller learned with our proposed algorithm to a potential field controller from the literature. They both achieved a high performance through different avoidance strategies.

Results from this chapter were presented at the International Conference on Robotics and Automation [113].

12 Parameters in PSO OCBA

In this chapter, we will look more in detail into the distributed PSO OCBA algorithm.

In particular, we will discuss the role of the neighborhood size, which is of interest in distributed implementations because small neighborhoods imply lower requirements in communication range and better scalability for large number of robots as they reduce the number of connections [6].

We will also analyze two parameters specific to OCBA: n_0, the number of samples used for the initial estimates of the mean and variance, and Δ, the additional number of samples used in the internal OCBA iterations. We will look at how their values affect the performance of PSO OCBA and test whether there are any differences in their optimal values with the guidelines proposed for stand-alone OCBA, i.e., when OCBA is used just by itself and not within the PSO loop.

The following section will describe the methodology for the experiments conducted in this chapter. Next, Section 12.2 will present the results from applying the different algorithms and parametrizations for learning in simulation. Finally, Section 12.3 provides a short summary of the chapter.

12.1 Experimental Methodology

We will perform our analysis on the multi-robot obstacle avoidance benchmark task described in Chapter 4.

All experiments are conducted in simulation using 4 robots. We use a square arena of 2 m x 2 m where 15 cylindrical obstacles of diameter 10 cm are randomly placed at the beginning of each fitness evaluation. The initial robots' positions at the beginning of each evaluation are also set randomly with a uniform probability distribution, verifying that they do not overlap with obstacles or other robots.

Table 12.1: Parameters common to all PSO algorithms

Parameter	Value
Number of robots N_{rob}	4
Population size N_p	24
Evaluation span t_e	30 s
Personal weight w_p	2.0
Neighborhood weight w_n	2.0
Neighborhood size N_n	3
Dimension D	24
Inertia w_I	0.8
Initial range X_{init}	20

Each particle evaluation consists of a robot moving in the arena for a fixed time (t_e =30 s) running the controller with the weights given by that particle's position. At the end of each evaluation, robots communicate the number of samples, mean, and variance of the solution they have evaluated, and they synchronize for the start of the next evaluation. The time required for the communication and synchronization is negligible in comparison to the evaluation time of the controllers (less than 1 second vs 30 seconds).

The controller used is *ann24*, the recurrent artificial neural network with 24 parameters described in Section4.3.

We will compare the performance of the distributed PSO OCBA, denominated *PSO ocbaD*, with two other algorithms that serve as reference: *PSO ocbaC*, the centralized PSO OCBA variant where the OCBA allocation is performed with full information, and *PSO pbest*, the noise-resistant variant by Pugh et al. [6] that re-evaluates personal bests at each iteration. For a description of the algorithms, please refer to Chapter 11. We will not use *PSO std* and *PSO rep* as we saw in Chapter 11 that their performance under noise is not competitive with the three algorithms mentioned first.

Table 12.1 shows the parameters that are common to all PSO algorithms used in this paper. They are set following the guidelines for limited-time adaptation presented in 5. For comparison purposes, we are using the same neighborhood size as Pugh et al. [6], which is one neighbor on each side of a ring topology.

The parameters for *PSO ocbaC* and *PSO ocbaD* are displayed in Table 12.2. The main difference in the values of *PSO ocbaC* and *PSO ocbaD* is that in *PSO ocbaD* each particle is performing its own OCBA allocation, therefore the iteration budget B_i for that allocation is 1/24 the budget of the centralized version, and the additional number of samples Δ is reduced to 1 in order to share and receive the results from other particles after each evaluation.

For each configuration of the algorithms under study we perform 20 learning runs for statistical significance. The number of iterations for each learning experiment is 250 in the case of *PSO pbest*, and 50 for *PSO ocbaC* and *PSO ocbaD*. This results in the same total number of

Table 12.2: Parameters for *PSO ocbaC* and *PSO ocbaD*

Parameter	*PSO ocbaC*	*PSO ocbaD*
Iterations N_i	50	50
Iteration budget B_i	240	10
Initial number of samples n_0	2	2
Additional number of samples Δ	4	1

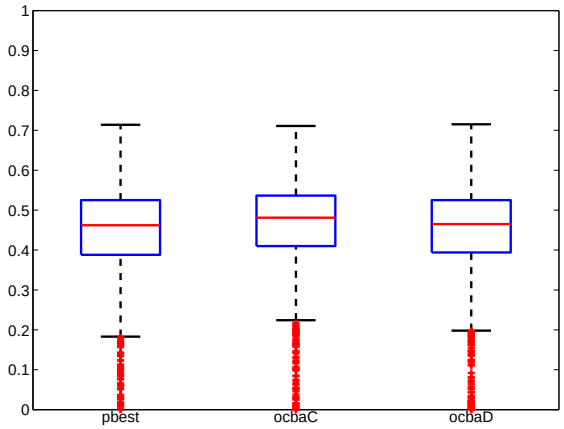


Figure 12.1: Final ground truth performance with default parameter values for *PSO pbest*, *PSO ocbaC*, and *PSO ocbaD*. The box represents the upper and lower quartiles, the line across the middle marks the median, the bars extend to the most extreme data points not considered outliers, and the red crosses show outliers.

evaluations for every algorithm.

Due to the presence of noise, the fitness value of the best solution as reported by the algorithms may not be an accurate representation of the actual performance of the solution. Therefore, in order to accurately judge the performances for comparison purposes, we perform 100 a posteriori evaluations of the best solution at each iteration and consider the mean of the 100 evaluations as the *ground truth* performance of that solution.

12.2 Results

We begin by comparing the final ground truth performance of the three algorithms, shown in Fig. 12.1. Due to the adequate selection of parameters, all algorithms achieve the desired robotic behavior with high performances, with *PSO ocbaC* slightly outperforming the two distributed algorithms. There is no statistically significant difference in ground truth performance between *PSO pbest* and *PSO ocbaD* (Mann Whitney U test, 5% significance level), but we will show next that there is a significant difference in their estimates of the ground truth.

We now compare the performances estimated by the distributed algorithms with the ground truth for different neighborhood sizes. Figures 12.2 and 12.3 show the progress of *PSO ocbaD*

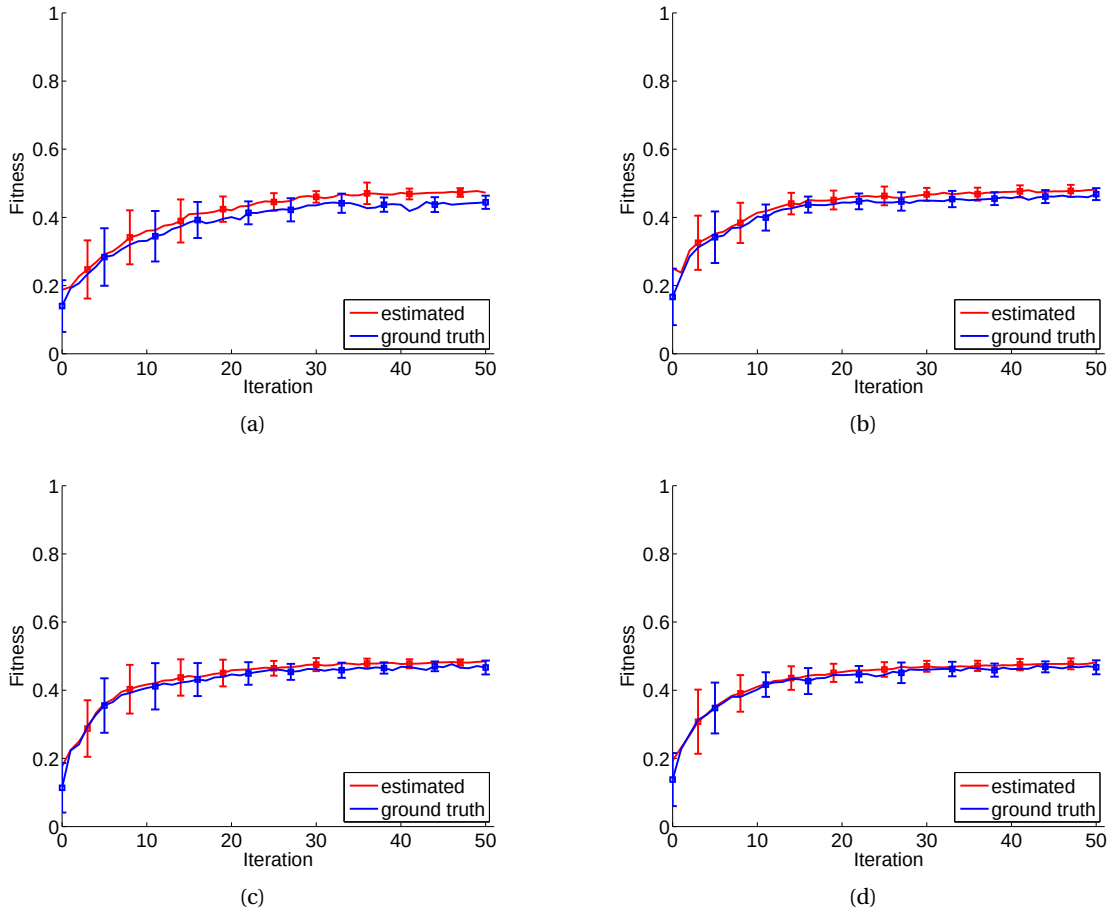


Figure 12.2: Progress averaged over 20 runs for *PSO ocbaD* with increasing neighborhood size. The red curve represents the performance of the best solution as estimated by the algorithm, and the blue curve represents the ground truth performance obtained as the average of 100 a posteriori evaluations. Error bars represent one standard deviation (a) Neighborhood size 3. (b) Neighborhood size 7. (c) Neighborhood size 15. (d) Neighborhoood size 24.

and *PSO pbest* for neighborhoods of size 3, 7, 15, and 24 (all other parameters keep the values mentioned in Section 12.1).

By comparing these two figures, we can see that the difference between estimates and ground truth is much lower in the case of *PSO ocbaD* than in *PSO pbest*, i.e., *PSO pbest* reports a much higher performance than the one it actually achieves. Also, for *PSO ocbaD* the difference becomes smaller for larger neighborhood sizes, which is not the case for *PSO pbest*.

In order to measure the differences between the performance estimated by the algorithm and the ground truth from a posteriori measurements we calculate the root-mean-squared error

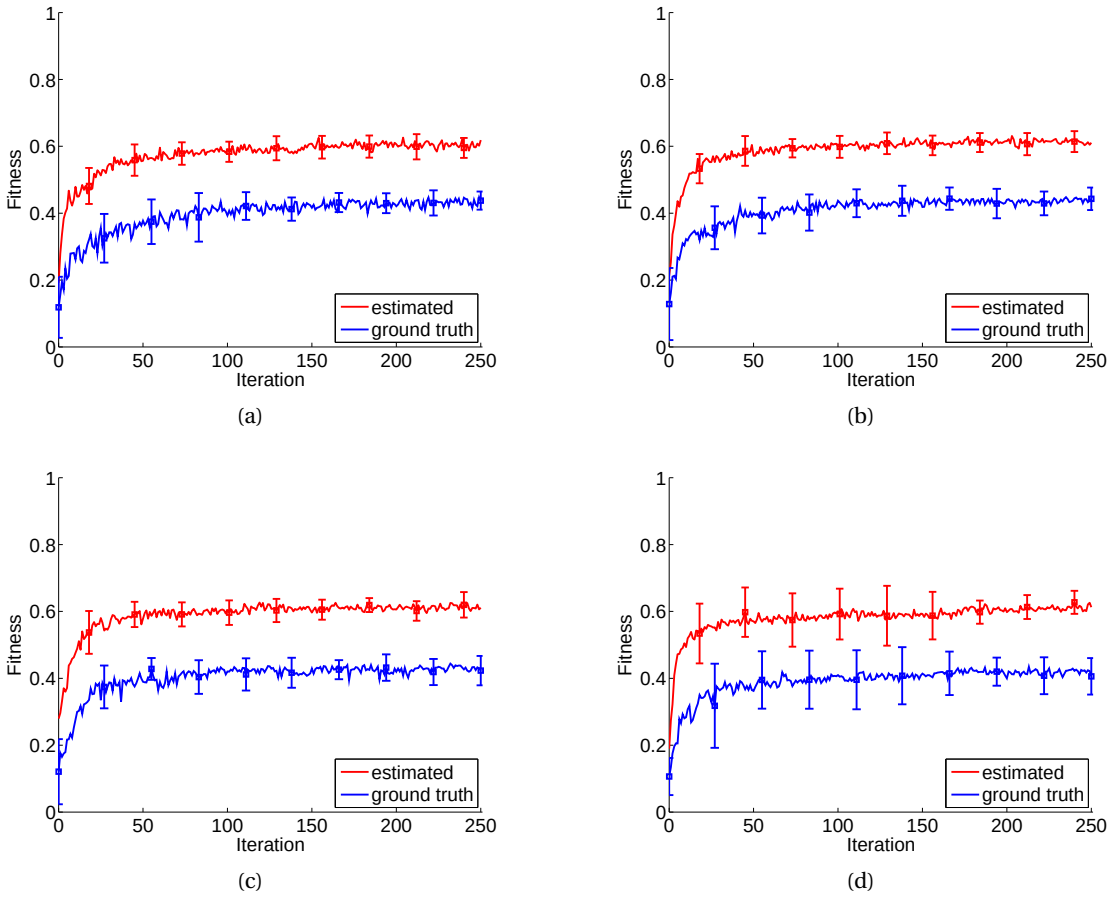


Figure 12.3: Progress averaged over 20 runs for *PSO pbest* with increasing neighborhood size. The red curve represents the performance of the best solution as estimated by the algorithm, and the blue curve represents the ground truth performance obtained as the average of 100 a posteriori evaluations. Error bars represent one standard deviation (a) Neighborhood size 3. (b) Neighborhood size 7. (c) Neighborhood size 15. (d) Neighborhoood size 24.

Table 12.3: Root-mean-squared error between estimates and ground truth for increasing neighborhood size.

Algorithm	Neighborhood size	RMSE
PSO ocbaC	3	0.009
PSO ocbaC	7	0.013
PSO ocbaC	15	0.012
PSO ocbaC	24	0.015
PSO ocbaD	3	0.033
PSO ocbaD	7	0.023
PSO ocbaD	15	0.024
PSO ocbaD	24	0.026
PSO pbest	3	0.19
PSO pbest	7	0.18
PSO pbest	15	0.19
PSO pbest	24	0.21

(RMSE):

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{X}_i - X_i)^2} \tag{12.1}$$

where $N = 20$ is the number of runs, $\hat{X}_i$ is the performance estimated by the algorithm for run i, and X_i is the ground truth for run i.

Table 12.3 shows the calculated RMSE for *PSO ocbaC*, *PSO ocbaD*, and *PSO pbest* using different neighborhood sizes. *PSO ocbaC* provides the best estimate of the ground truth performance, followed by *PSO ocbaD*. The errors for both PSO OCBA variants are one order of magnitude smaller than for *PSO pbest*.

We now switch to the analysis of two parameters that are specific to the OCBA algorithm: n_0 and Δ.

The first OCBA parameter, n_0, is the number of samples used for the initial estimates of the mean and variance. Low values can lead to poor initial estimates, while high values may waste function evaluations on poor solutions. Chen et al. suggested values of n_0 between 5 and 20 for stand-alone OCBA in [107].

We ran the two PSO OCBA algorithms with three different n_0 values: 2, 4, and 8 (all other parameters keep the values mentioned in Section 12.1); results are shown in Table 12.4. From these experiments, we found that when applying OCBA in PSO, the lowest possible value of $n_0 = 2$ led to the best final estimate of the ground truth performance, which is different from the guideline for stand-alone OCBA. This result suggests that for PSO OCBA it is better to

Table 12.4: Root-mean-squared error between estimates and ground truth for increasing n_0.

Algorithm	n_0	Δ	RMSE
PSO ocbaC	2	4	0.009
PSO ocbaC	4	4	0.019
PSO ocbaC	8	4	0.025
PSO ocbaD	2	1	0.033
PSO ocbaD	4	1	0.035
PSO ocbaD	8	1	0.051

Table 12.5: Root-mean-squared error between estimates and ground truth for increasing Δ.

Algorithm	n_0	Δ	RMSE
PSO ocbaC	2	4	0.009
PSO ocbaC	2	8	0.014
PSO ocbaC	2	16	0.022
PSO ocbaC	2	32	0.017
PSO ocbaD	2	1	0.033
PSO ocbaD	2	2	0.039
PSO ocbaD	2	4	0.065
PSO ocbaD	2	8	0.056

save function evaluations in the initial estimates and perform more iterations of the OCBA algorithm.

The second OCBA parameter that we analyze is Δ, the additional number of samples used in the internal OCBA iterations. A smaller delta means that more OCBA calculations are performed, which is more computationally expensive but provides better estimates because mean and standard deviation are updated more often. The guideline given for this parameter for stand-alone OCBA is to select a number bigger than 5 but smaller than 10% of the number of candidates [107]. This guideline cannot be directly applied to our case, as the number of candidates is 48 (24 new particle positions and 24 personal bests), so the guideline would suggest a number bigger than 5 but smaller than 4.8.

Table 12.5 shows the results from experiments with the two PSO OCBA algorithms and different values of Δ, where it can be seen that lower values of Δ produce a lower error. In robotic learning the computational cost of the OCBA procedure is negligible compared to the cost of evaluations (both with high-fidelity simulation and real robot experiments), which is why the guidelines from stand-alone OCBA do not necessarily apply. Therefore, we can select the lowest possible value for Δ, which is one for the distributed case and four (the number of robots in the system) for the centralized one (in order to avoid idle robots and evaluate a maximum number of solutions in parallel).

12.3 Chapter Summary

This chapter analyzed the role of the different parameters of the distributed PSO OCBA algorithm, and compared it with the centralized PSO OCBA and *PSO pbest*, the distributed noise-resistant PSO of Pugh et al. [7].

Even though controllers learned with the three algorithms obtained similar performance in the obstacle avoidance benchmark task, there were significant differences in their estimates of the ground truth performance, as measured by the root mean square error between the estimates and the ground truth.

Both PSO OCBA versions provided estimates of the ground truth performance that were an order of magnitude better than that of *PSO pbest*. The distributed version's estimate of the ground truth was slightly worse than the centralized one, but this difference was reduced when the neighborhood size was increased. This was not the case for the previous distributed noise-resistant PSO, whose estimation error remained more or less constant regardless of the neighborhood size.

We also analyzed two parameters specific to OCBA: n_0 and Δ. We showed that the lowest possible values ($n_0 = 2$, Δ = number of robots) performed best, and these values differ considerably from the guidelines for stand-alone OCBA present in the literature.

Results from this chapter were presented at the IEEE Congress on Evolutionary Computation [114].

Conclusion Part IV

13 Discussion: Approaching New Learning Problems

In this chapter we will try to integrate what we have learned in previous chapters and discuss how we would approach a new multi-robot learning problem using the distributed PSO framework.

The first step when approaching a new problem is to define an appropriate performance metric. As mentioned in Subsection 3.5.1, performance metrics are usually designed by combining factors for more specific, simpler behaviors. Surveys of common metrics used in robotics such as [65] can provide individual factors for several basic behaviors, or inspiration on how to design new ones by analogy with preexisting ones.

Once the individual factors are selected, we need to define a way to aggregate them. Floreano and Mondada [2] adjusted the factors in their metric for obstacle avoidance manually, by adding a square root in the turning factor to obtain the desired behavior. When designing the flocking metrics, we preferred a similar but more general approach, the generalized aggregation functions of Scott and Antonsson [67], which employ weight parameters to fine tune the behaviors. An advantage of this approach is that if the full Pareto frontier is needed, it can be obtained by systematically exploring the entire range of aggregation weights [68], which would give the same outcome as using a multi-objective optimization algorithm [66]. Still, obtaining the entire Pareto frontier is an extremely computationally expensive process since a full optimization run must be performed for each possible combination of the aggregation weights.

Next, we need to select the controller architecture, which is tied to the sensing and actuating capabilities of the underlying hardware, and the way it is going to be parametrized. Throughout this thesis we have used Artificial Neural Networks due to the fact that they have been shown to be able uniformly approximate arbitrary functions [69], [70] and their complexity can be adjusted by adding or removing units, as we saw in Chapter 7. Nevertheless, PSO could be applied to the optimization of any control architecture that can be defined by a finite set of real parameters (there are specific variations of PSO for discrete ones [115]–[117]), and it has been used for example on dedicated controllers for bio-inspired search [118] and flocking [94],

[95].

It is important to normalize inputs, outputs, and parameters, since having a variable with a range significantly larger than the others makes the optimization problem ill-conditioned and thus considerably harder [38]. Normalization can be done by subtracting the mean and dividing by the range or the standard deviation of a given variable. This should not constitute a major problem as inputs and outputs in robotics are related to sensors and actuators which usually have a pre-defined operating range.

After defining the performance metric and the controller, we should select the learning environment. In order to do so, we should consider the final environment where the robots will be deployed. If it is very clearly defined and structured, as in for instance a manufacturing robot with a fixed position in an assembly line, we should try to reproduce it as much as possible during the learning, which will result in a specific controller very efficient for the task at hand. On the other hand, if the final environment is not clearly defined or variable, such as in the case of domestic robots, we should try to perform the learning in several environments encompassing all the difficulties that a robot may encounter, in order to produce more general, robust controllers, at the cost of more learning time, as we saw in Chapter 6.

At this point, we can formulate the learning as an optimization problem, i.e., find the set of parameters of the controller that maximize the performance metric (or minimize a cost metric). Therefore, we can now apply PSO, using the guidelines given in Chapter 5 to select the algorithmic parameters that make the best out of the available evaluation time window.

We suggest running the algorithm in simulation first, even if the modelling is not perfect, because it allows for quick iteration when performing adjustments such as follows. First, we can verify if the performance metric results in the desired behavior, and adjust the weights of the different factors if needed. Second, we can estimate the amount of noise in function evaluations and, if using an appropriate noise-resistant algorithm such as the one presented in Chapter 11, adjust the sampling budget. Third, by plotting the learning progress as a function of the iterations, we can monitor convergence and decide if more learning time is needed since, even though given adequate parameter selection PSO has been proven to converge [119], [120] when $t \to \infty$, there is no guarantee of the amount of time required to do so. Last, simulations avoid damaging the real hardware in case of collisions or other potentially dangerous situations. In fact, this is a limitation of the approach because even if constraints in the search space can easily be added to PSO [121], it is hard to map all possible dangerous situations to those kind of constraints. A solution to this problem is to have low-level safety procedures that override the controller to be learned in case of dangerous situations. For instance, in the obstacle avoidance case, the wheel speeds can be set to zero when the proximity sensors reach a certain threshold, thus avoiding collisions and penalizing the controller at the same time because of the lack of movement. Again, these kind of safety procedures can be verified in simulation before moving to the real robots.

Once we have satisfactory results in simulation, we can address any errors in our simulation

model by conducting additional learning iterations on the real robots. For this purpose, instead of learning again from the start we can use the hybrid approach proposed in Chapter 5 where the pool of candidate solutions is transferred to the real robots for final refinement after the initial simulation stages.

14 Conclusion and Outlook

We can classify the contributions of this thesis into three groups.

First, we have the contributions arising from the benchmarking of the multi-robot obstacle avoidance task. We analyzed the algorithmic parameters determining the total evaluation time in distributed, multi-robot implementations and proposed guidelines to select these parameters given a limited total evaluation time. We conducted the learning both in simulation and reality, and addressed the differences between them through a hybrid approach in which the pool of candidate solutions is transferred to real robots for final refinement after the initial simulation stages. We have experimentally shown that such approach, if based on a reasonably faithful simulator, achieved a similar performance as pure real-robot learning while requiring considerably less real-robot evaluation time. We also looked at ways of manipulating controller complexity by grouping the sensors used as input, adding non-linear functions, and adding memory in the form of recurrence in the neural network. The neural network controller was able to generate a wide range of behaviors, adapting to the environment where the learning took place. While learning in more cluttered environments resulted in more robust controllers, it also made the optimization process harder in the sense that performance measurements were noisier, the optimal parameter region was smaller, and more iterations were required for the optimization process to converge.

The second group of contributions come from applying the distributed PSO framework to a collaborative task such as flocking. In order to do so, we designed a local performance metric that could be evaluated using on-board sensing exclusively, and compared the performance of the distributed learning with the centralized one using a global metric. We showed that it was possible to learn this collaborative task in a fully distributed manner, and we validated the simulations with real robot experiments where the best solutions from distributed and centralized learning achieved similar performances.

The third group of contributions are related to the enhancement of the performance of PSO under noise. We first modeled and reproduced the effects of noise from robotic learning on numerical benchmark functions, and identified issues in previous noise-resistant algorithms

such as early stagnation and overestimation of the goodness of solutions. We then presented a distributed PSO OCBA algorithm that addresses those issues, and that is suitable for resource-constrained mobile robots due to its low requirements in terms of memory and limited local communication.

As future work, we would like to apply the distributed PSO OCBA algorithm to collaborative tasks such as flocking, and, more generally, to different tasks in the multi-robot domain, in order to test its general potential for robotic problems. Furthermore, we would like to increase the number of robots to see how the algorithm performs on larger scales, and what are the limits that we can achieve in terms of learning time per parameter.

We would also like to explore alternative control architectures to artificial neural networks. In particular, we are interested in graph-based distributed control [122], which, since it can be implemented in a fully distributed manner, it is a suitable candidate for the optimization approaches presented in this thesis.

Lastly, we are interested in exploring PSO variations and other population-based algorithms that can be applied to distributed learning. Even though in this thesis we focused on PSO, we believe that the evaluation time, noisy performance measurements, and simulation transfer issues discussed are relevant to population-based multi-robot learning in general and not limited to a particular algorithm.

Bibliography

[1] J. J. Liang, B. Y. Qu, P. N. Suganthan, and A. G. Hernández-Díaz, *Problem definitions and evaluation criteria for the CEC 2013 special session on real-parameter optimization*, 2013.

[2] D. Floreano and F. Mondada, "Evolution of homing navigation in a real mobile robot", *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 26, no. 3, pp. 396–407, 1996.

[3] 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*, vol. 39, no. 1, pp. 1–18, 2002.

[4] J Kennedy and R Eberhart, "Particle swarm optimization", in *IEEE International Conference on Neural Networks*, 1995, 1942 –1948 vol.4.

[5] B. Al-Kazemi and S. Habib, "Complexity analysis of problem-dimension using PSO", in *WSEAS International Conference on Evolutionary Computing*, 2006, pp. 45–52.

[6] J. Pugh and A. Martinoli, "Distributed scalable multi-robot learning using particle swarm optimization", *Swarm Intelligence*, vol. 3, no. 3, pp. 203–222, 2009.

[7] J. Pugh, Y. Zhang, and A. Martinoli, "Particle swarm optimization for unsupervised robotic learning", in *IEEE Swarm Intelligence Symposium*, 2005, pp. 92–99.

[8] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: overview and conceptual comparison", *ACM Computing Surveys*, vol. 35, no. 3, pp. 268–308, 2003.

[9] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, ser. Adaptive Computation and Machine Learning 5. MIT Press, 1998, vol. 9, ch. Dynamic pr, p. 360.

[10] J. Kober and J. Peters, "Reinforcement learning in robotics : a survey", *Reinforcement Learning*, pp. 579–610, 2012.

[11] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Ed. Addison-Wesley, 1989, p. 432.

[12] H. Beyer and H. Schwefel, "Evolution strategies: a comprehensive introduction", *Natural Computing*, pp. 3–52, 2002.

[13] M. Dorigo and L. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem", *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, 1997.

[14] R. Poli, "Analysis of the publications on the applications of particle swarm optimisation", *Journal of Artificial Evolution and Applications*, vol. 2008, no. 2, pp. 1–10, 2008.

[15] S Doctor, G. K. Venayagamoorthy, and V. G. Gudise, "Optimal PSO for collective robotic search applications", in *IEEE Congress on Evolutionary Computation*, 2004, pp. 1390 –1395.

[16] J. Hereford and M. Siebold, "Using the particle swarm optimization algorithm for robotic search applications", in *IEEE Swarm Intelligence Symposium*, 2007, pp. 53–59.

[17] J. Pugh and A. Martinoli, "Inspiring and modeling multi-robot search with particle swarm optimization", in *IEEE Swarm Intelligence Symposium*, 2007, pp. 332–339.

[18] X. Chen and Y. Li, "Smooth path planning of a mobile robot using stochastic particle swarm optimization", in *IEEE International Conference on Mechatronics and Automation*, 2006, pp. 1722–1727.

[19] P. Huang and Y. Xu, "PSO-based time-optimal trajectory planning for space robot with dynamic constraints", in *IEEE International Conference on Robotics and Biomimetics*, 2006, pp. 1402–1407.

[20] K Lei and Y Qiu, "A novel path planning for mobile robots using modified particle swarm optimizer", *International Symposium on Systems and Control in Aerospace and Astronautics*, pp. 981–984, 2006.

[21] Y.-Q. Qin, D.-B. Sun, N. Li, and Y.-G. Cen, "Path planning for mobile robot using particle swarm optimization with mutation operator", in *International Conference on Machine Learning and Cybernetics*, 2004, pp. 2473 –2478.

[22] L. Marques, U. Nunes, and A. T. Almeida, "Particle swarm-based olfactory guided search", *Autonomous Robots*, vol. 20, no. 3, pp. 277–287, 2006.

[23] M. Turduev and Y. Atas, "Cooperative chemical concentration map building using decentralized asynchronous particle swarm optimization based search by mobile robots", in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 4175–4180.

[24] W Jatmiko, K Sekiyama, and T Fukuda, "A mobile robots PSO-based for odor source localization in dynamic advection-diffusion environment", in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006, pp. 4527–4532.

[25] R. Moeckel, Y. N. Perov, A. T. Nguyen, M. Vespignani, S. Bonardi, S. Pouya, A. Sproewitz, J. Van Den Kieboom, F. Wilhelm, and A. J. Ijspeert, "Gait optimization for Roombots modular robots - matching simulation and reality", *IEEE International Conference on Intelligent Robots and Systems*, pp. 3265–3272, 2013.

[26] J. Chang, S. Chu, and J. Roddick, "A parallel particle swarm optimization algorithm with communication strategies", *Journal of Information Science*, pp. 809–818, 2005.

[27] S. B. Akat and V. Gazi, "Decentralized asynchronous particle swarm optimization", in *IEEE Swarm Intelligence Symposium*, DOI: 10.1109/SIS.2008.4668304, 2008.

[28] J. Rada-Vilela, M. Zhang, and W. Seah, "Random asynchronous PSO", *International Conference on Automation, Robotics and Applications*, pp. 220–225, 2011.

[29] Y. Jin and J Branke, "Evolutionary optimization in uncertain environments: a survey", *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 3, pp. 303–317, 2005.

[30] J. Fitzpatrick and J. Grefenstette, "Genetic algorithms in noisy environments", *Machine Learning*, vol. 3, no. 2, pp. 101–120, 1988.

[31] P. Stagge, "Averaging efficiently in the presence of noise", in *Parallel Problem Solving from Nature — PPSN V*, ser. Lecture Notes in Computer Science, vol. 1498, Springer Berlin / Heidelberg, 1998, pp. 188–197.

[32] D. Arnold and H.-G. Beyer, "A general noise model and its effects on evolution strategy performance", *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 4, pp. 380–391, 2006.

[33] K. E. Parsopoulos and M. N. Vrahatis, "Particle swarm optimizer in noisy and continuously changing environments", *Artificial Intelligence and Soft Computing*, pp. 289–294, 2001.

[34] H Pan, L Wang, and B Liu, "Particle swarm optimization for function optimization in noisy environment", *Applied Mathematics and Computation*, vol. 181, no. 2, pp. 908–919, 2006.

[35] R Eberhart and J Kennedy, "A new optimizer using particle swarm theory", in *International Symposium on Micro Machine and Human Science.*, 1995, pp. 39–43.

[36] J Kennedy and W. M. Spears, "Matching algorithms to problems: an experimental test of the particle swarm and some genetic algorithms on the multimodal problem generator", in *IEEE Congress on Evolutionary Computation*, 1998, pp. 78–83.

[37] P. Fourie and A. Groenwold, "The particle swarm optimization algorithm in size and shape optimization", *Structural and Multidisciplinary Optimization*, vol. 23, no. 4, pp. 259–267, 2002.

[38] N. Hansen, R. Ros, N. Mauny, M. Schoenauer, and A. Auger, "Impacts of invariance in search: when CMA-ES and PSO face ill-conditioned and non-separable problems", *Applied Soft Computing*, vol. 11, no. 8, pp. 5755–5769, 2011.

[39] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization", *Evolutionary Computation, IEEE Transactions on*, vol. 1, no. 1, pp. 67–82, 1997.

[40] A. Hayes, A Martinoli, and R. Goodman, "Swarm robotic odor localization: off-line optimization and validation with real robots", *Robotica*, vol. 21, no. 4, pp. 427–441, 2003.

[41] R. Salustowicz, M. Wiering, and J. Schmidhuber, "Learning team strategies: soccer case studies", *Machine Learning*, vol. 33, no. 2, pp. 263–282, 1998.

[42] C. Versino and L. Gambardella, "Learning real team solutions", *Distributed Artificial Intelligence Meets Machine Learning Learning in Multi-Agent Environments*, pp. 40–61, 1997.

[43] L. Panait and S. Luke, "Cooperative multi-agent learning: the state of the art", *Autonomous Agents and Multi-Agent Systems*, vol. 11, no. 3, pp. 387–434, 2005.

[44] L Li, A Martinoli, and Y Abu-Mostafa, "Learning and measuring specialization in collaborative swarm systems", *Adaptive Behavior*, vol. 12, no. 3-4, pp. 199–212, 2004.

[45] A. Murciano, J. del R. Millán, and J. Zamora, "Specialization in multi-agent systems through learning", *Biological Cybernetics*, vol. 76, no. 5, pp. 375–382, 1997.

[46] A. Martinoli, "Swarm intelligence in autonomous collective robotics: from tools to the analysis and synthesis of distributed control strategies", PhD thesis, EPFL, 1999.

[47] M. Matarić, "Learning in behavior-based multi-robot systems: policies, models, and other agents", *Cognitive Systems Research*, vol. 2, pp. 81–93, 2001.

[48] L. E. Parker, "L-ALLIANCE : task-oriented multi-robot learning in behavior-based systems", in *Advanced Robotics, Special Issue on Selected Papers from IROS'96*, 1997, pp. 305–322.

[49] M. Quinn, L. Smith, G. Mayley, and P. Husbands, "Evolving controllers for a homogeneous system of physical robots: structured cooperation with minimal sensors", *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 361, no. 1811, p. 2321, 2003.

[50] 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. Part B, Cybernetics*, vol. 37, no. 1, pp. 224–39, 2007.

[51] M. Gauci, J. Chen, T. Dodd, and R. Groß, "Evolving aggregation behaviors in multi-robot systems with binary sensors", *International Symposium on Distributed Autonomous Robotic Systems, Springer Tracts in Advanced Robotics*, vol. 104, pp. 355–367, 2014.

[52] J. Pugh, X. Raemy, C. Favre, R. Falconi, and A. Martinoli, "A fast on-board relative positioning module for multi-robot systems", *IEEE/ASME Transactions on Mechatronics, Focused Section on Mechatronics in Multi Robot Systems*, pp. 151–162, 2009.

[53] T. Lochmatter, P. Roduit, C. Cianci, N. Correll, J. Jacot, and A. Martinoli, "Swistrack: a flexible open source tracking software for multi-agent systems", in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 4004–4010.

[54] O. Michel, "Webots: professional mobile robot simulation", *Advanced Robotic Systems*, vol. 1, no. 1, pp. 39–42, 2004.

[55] K. A. De Jong, "An analysis of the behavior of a class of genetic adaptive systems", PhD thesis, University of Michigan, 1975.

[56] G. Jornod, E. Di Mario, I. Navarro, and A. Martinoli, "SwarmViz: an open-source visualization tool for particle swarm optimization", in *IEEE Congress on Evolutionary Computation*, 2015.

[57] B. Huang, G. Cao, and M Guo, "Reinforcement learning neural network to the problem of autonomous mobile robot obstacle avoidance", in *International Conference on Machine Learning and Cybernetics*, 2005, pp. 85–89.

[58] T. Balch and R. Arkin, "Behavior-based formation control for multi-robot teams", *IEEE Transactions on Robotics and Automation*, vol. 14, no. 6, pp. 926–939, 1998.

[59] J. Fredslund and M. J. Mataric, "A general algorithm for robot formations using local sensing and minimal communication", *IEEE Transactions on Robotics and Automation, Special Issue on Multi Robot Systems*, vol. 18, no. 5, pp. 837–846, 2002.

[60] R. Olfati-Saber, "Flocking for multi-agent dynamic systems: algorithms and theory", *IEEE Transactions on Automatic Control*, vol. 51, pp. 401–420, 2006.

[61] G. Antonelli, F. Arrichiello, and S. Chiaverini, "Flocking for multi-robot systems via the null-space-based behavioral control", in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 1409–1414.

[62] I. Navarro and F. Matía, "A framework for collective movement of mobile robots based on distributed decisions", *Robotics and Autonomous Systems*, vol. 59, no. 10, pp. 685–697, 2011.

[63] H. Çelikkanat and E. Şahin, "Steering self-organized robot flocks through externally guided individuals", *Neural Computing and Applications*, vol. 19, no. 6, pp. 849–865, 2010.

[64] A. E. Turgut, H. Çelikkanat, F. Gökçe, and E. Şahin, "Self-organized flocking in mobile robot swarms", *Swarm Intelligence*, vol. 2, no. 2-4, pp. 97–120, 2008.

[65] A. L. Nelson, G. J. Barlow, and L. Doitsidis, "Fitness functions in evolutionary robotics: a survey and analysis", *Robotics and Autonomous Systems*, vol. 57, no. 4, pp. 345–370, 2009.

[66] C. A. Coello Coello and M. S. Lechuga, "MOPSO: a proposal for multiple objective particle swarm optimization", in *IEEE Congress on Evolutionary Computation*, 2002, pp. 1051–1056.

[67] M. J. Scott and E. K. Antonsson, "Aggregation functions for engineering design trade-offs", *Fuzzy Sets and Systems*, vol. 99, no. 3, pp. 253–264, 1998.

[68] Y. Zhang, E. Antonsson, and A. Martinoli, "Evolutionary engineering design synthesis of on-board traffic monitoring sensors", *Research in Engineering Design*, vol. 19, no. 2, pp. 113–125, 2008.

[69] G. Cybenko, "Approximation by superpositions of a sigmoidal function", *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, 1989.

[70] K. Hornik, "Approximation capabilities of multilayer feedforward networks", *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991.

[71] S. B. Akat and V. Gazi, "Particle swarm optimization with dynamic neighborhood topology: three neighborhood strategies and preliminary results", in *IEEE Swarm Intelligence Symposium*, DOI: 10.1109/SIS.2008.4668298, 2008.

[72] R. E. Palacios-Leyva, R. Cruz-Alvarez, F. Montes-Gonzalez, and L. Rascon-Perez, "Combination of reinforcement learning with evolution for automatically obtaining robot neural controllers", in *IEEE International Conference on Evolutionary Computation*, 2013, pp. 119–126.

[73] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp, "A racing algorithm for configuring metaheuristics", in *Genetic and Evolutionary Computation Conference*, 2002, pp. 11–18.

[74] M. Gagliolo and J. Schmidhuber, "Algorithm portfolio selection as a bandit problem with unbounded losses", *Annals of Mathematics and Artificial Intelligence*, vol. 61, no. 2, pp. 49–86, 2011.

[75] E. Di Mario, G. Mermoud, M. Mastrangeli, and A. Martinoli, "A trajectory-based calibration method for stochastic motion models", in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 4341–4347.

[76] D. Bratton and J. Kennedy, "Defining a standard for particle swarm optimization", in *IEEE Swarm Intelligence Symposium*, 2007, pp. 120–127.

[77] E. Di Mario and A. Martinoli, "Distributed particle swarm optimization for limited time adaptation in autonomous robots", in *International Symposium on Distributed Autonomous Robotic Systems 2012; Springer Tracts in Advanced Robotics 2014*.

[78] E. Di Mario and A. Martinoli, "Distributed particle swarm optimization for limited time adaptation with real robots", *Robotica*, vol. 32, no. 02, pp. 193–208, 2014.

[79] S. Nolfi, "Behaviour as a complex adaptive system: on the role of self-organization in the development of individual and collective behaviour", *ComPlexUs*, vol. 2, no. 3-4, pp. 195–203, 2005.

[80] S. Nolfi and D. Parisi, "Learning to adapt to changing environments in evolving neural networks", *Adaptive Behavior*, vol. 5, pp. 75–98, 1996.

[81] J. E. Auerbach and J. C. Bongard, "On the relationship between environmental and morphological complexity in evolved robots", in *Genetic and Evolutionary Computation Conference*, ACM Press, 2012, pp. 521–528.

[82] M. M. Islam and K. Murase, "Chaotic dynamics of a behavior-based miniature mobile robot: effects of environment and control structure", *Neural Networks*, vol. 18, no. 2, pp. 123 –144, 2005.

[83] A. Berlanga, A. Sanchis, P. Isasi, and J. M. Molina, "Neural network controller against environment: a coevolutive approach to generalize robot navigation behavior", *Journal of Intelligent and Robotics Systems*, vol. 33, no. 2, pp. 139–166, 2002.

[84] A. Nelson, E. Grant, G. Barlow, and M. White, "Evolution of complex autonomous robot behaviors using competitive fitness", in *International Conference on Integration of Knowledge Intensive Multi-Agent Systems*, 2003, pp. 145–150.

[85] E. Di Mario, I. Navarro, and A. Martinoli, "The effect of the environment in the synthesis of robotic controllers: a case study in multi-robot obstacle avoidance using distributed particle swarm optimization", in *European Conference on the Synthesis and Simulation of Living Systems*, MIT Press, 2013, pp. 561–568.

[86] E. Di Mario, I. Navarro, and A. Martinoli, "The role of environmental and controller complexity in the distributed optimization of multi-robot obstacle avoidance", in *IEEE International Conference on Robotics and Automation*, 2014, pp. 571–577.

[87] W. Smart and L. Pack Kaelbling, "Effective reinforcement learning for mobile robots", in *IEEE International Conference on Robotics and Automation*, 2002, pp. 3404–3410.

[88] T. Yasuda and K. Ohkura, "A reinforcement learning technique with an adaptive action generator for a multi-robot system", *From Animals to Animats*, pp. 250–259, 2008.

[89] M. Matarić, "Reinforcement learning in the multi-robot domain", *Autonomous Robots*, vol. 4, no. 1, pp. 73–83, 1997.

[90] C. J. C. H. Watkins, "Learning from delayed rewards", PhD thesis, University of Cambridge England, 1989.

[91] H. Iima, Y. Kuroe, and K. Emoto, "Swarm reinforcement learning methods for problems with continuous state-action space", in *IEEE International Conference on Systems, Man, and Cybernetics*, 2011, pp. 2173–2180.

[92] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation", DTIC Document, Tech. Rep., 1985.

[93] E. Di Mario, Z. Talebpour, and A. Martinoli, "A comparison of PSO and reinforcement learning for multi-robot obstacle avoidance", in *IEEE Conference on Evolutionary Computation*, vol. 1, 2013, pp. 149–156.

[94] S.-M. Lee and H. Myung, "Particle swarm optimization-based distributed control scheme for flocking robots", *Robot Intelligence Technology and Applications*, vol. 208, pp. 517–524, 2013.

[95] S. Etemadi, R. Vatankhah, A. Alasty, G. Vossoughi, and M. Boroushaki, "Leader connectivity management and flocking velocity optimization using the particle swarm optimization method", *Scientia Iranica*, vol. 19, no. 5, pp. 1251 –1257, 2012.

[96] Y.-H. Chang, C.-L. Chen, W.-S. Chan, H.-W. Lin, and C.-W. Chang, "Fuzzy formation control and collision avoidance for multiagent systems", *Mathematical Problems in Engineering*, vol. 2013, 2013.

[97] A. T. Hayes and P. Dormiani-Tabatabaei, "Self-organized flocking with agent failure: off-line optimization and demonstration with real robots", in *IEEE International Conference on Robotics and Automation*, 2002, pp. 3900–3905.

[98] K. Morihiro, T. Isokawa, H. Nishimura, and N. Matsui, "Emergence of flocking behavior based on reinforcement learning", *Knowledge-Based Intelligent Information and Engineering Systems*, vol. 4253, pp. 699–706, 2006.

[99] T. I. Zohdi, "Computational design of swarms", *International Journal for Numerical Methods in Engineering*, vol. 57, pp. 2205–2219, 2003.

[100] H. Celikkanat, "Optimization of self-organized flocking of a robot swarm via evolutionary strategies", in *International Symposium on Computer and Information Sciences*, 2008.

[101] C. W. Reynolds, "Flocks, herds, and schools: a distributed behavioral model", *Computer Graphics*, vol. 21, no. 4, pp. 25–34, 1987.

[102] I. Navarro, E. Di Mario, and A. Martinoli, "Distributed vs centralized particle swarm optimization for learning flocking behaviors", in *European Conference on the Synthesis and Simulation of Living Systems*, 2015, pp. 302–309.

[103] E. Di Mario, I. Navarro, and A. Martinoli, "Distributed learning of cooperative robotic behaviors using particle swarm optimization", in *International Symposium on Experimental Robotics, to appear in Springer Tracts in Advanced Robotics*, 2014.

[104] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: the use of simulation in evolutionary robotics", *Advances in artificial life*, 1995.

[105] T Bartz-Beielstein, D. Blum, and J Branke, "Particle swarm optimization and sequential sampling in noisy environments", *Metaheuristics*, vol. 39, pp. 261–273, 2007.

[106] E. Di Mario, I. Navarro, and A. Martinoli, "Analysis of fitness noise in particle swarm optimization: from robotic learning to benchmark functions", *IEEE Congress on Evolutionary Computation*, pp. 2785–2792, 2014.

[107] C. Chen, J. Lin, E. Yücesan, and S. E. Chick, "Simulation budget allocation for further enhancing the efficiency of ordinal optimization", *Discrete Event Dynamic Systems: Theory and Applications*, pp. 251–270, 2000.

[108] R. C. Arkin, "Motor schema – based mobile robot navigation", *The International Journal of Robotics Research*, vol. 8, no. 4, pp. 92–112, 1989.

[109] J. Rada-Vilela, M. Zhang, and M. Johnston, "Optimal computing budget allocation in particle swarm optimization", in *Genetic and Evolutionary Computation Conference*, ACM, 2013, pp. 81–88.

[110] C. Schmidt, J. Branke, and S. Chick, "Integrating techniques from statistical ranking into evolutionary algorithms", *Applications of Evolutionary Computing*, vol. 3907, pp. 752–763, 2006.

[111] G. LaPorte, J. Branke, and C.-H. Chen, "Optimal computing budget allocation for small computing budgets", in *Winter Simulation Conference*, 2012, pp. 1–13.

[112] D. Balkcom and M. Mason, "Time optimal trajectories for bounded velocity differential drive vehicles", *The International Journal of Robotics Research*, vol. 21, no. 3, pp. 199–217, 2002.

[113] E. Di Mario, I. Navarro, and A. Martinoli, "A distributed noise-resistant particle swarm optimization algorithm for high-dimensional multi-robot learning", in *IEEE International Conference on Robotics and Automation*, 2015, pp. 5970–5976.

[114] E. Di Mario, I. Navarro, and A. Martinoli, "Distributed particle swarm optimization using optimal computing budget allocation for multi-robot learning", in *IEEE Congress on Evolutionary Computation*, 2015.

[115] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm", in *IEEE International Conference on Systems, Man, and Cybernetics*, IEEE, vol. 5, 1997, pp. 4104–4108.

[116] M. Clerc, "Discrete particle swarm optimization, illustrated by the traveling salesman problem", in *New Optimization Techniques in Engineering*, Springer, 2004, pp. 219–239.

[117] J. Pugh and A. Martinoli, "Discrete multi-valued particle swarm optimization", in *IEEE Swarm Intelligence Symposium*, 2006, pp. 103–110.

[118] J. Pugh and A. Martinoli, "Distributed adaptation in multi-robot search using particle swarm optimization", in *From Animals to Animats 10*, Springer, 2008, pp. 393–402.

[119] M Clerc and J Kennedy, "The particle swarm - explosion, stability, and convergence in a multidimensional complex space", *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 58–73, 2002.

[120] F. Van den Bergh and A. P. Engelbrecht, "A convergence proof for the particle swarm optimiser", *Fundamenta Informaticae*, vol. 105, no. 4, pp. 341–374, 2010.

[121] X. Hu and R. Eberhart, "Solving constrained nonlinear optimization problems with particle swarm optimization", in *Conference on Systemics, Cybernetics and Informatics*, vol. 5, 2002, pp. 203–206.

[122] S. A. Gowal, "A framework for graph-based distributed rendezvous of nonholonomic multi-robot systems", PhD thesis, EPFL, 2013.

[123] X. Chen and Y. Li, "Neural network predictive control for a mobile robot using PSO with controllable random exploration velocity", *International Journal of Intelligent Control and Systems*, vol. 12, no. 3, pp. 217–229, 2007.

[124] H. Lund and O. Miglino, "From simulated to real robots", in *IEEE International Conference on Evolutionary Computation*, 1996, pp. 362–365.

[125] R. Vatankhah, S. Etemadi, M. Honarvar, A. Alasty, M. Boroushaki, and G. Vossoughi, "Online velocity optimization of robotic swarm flocking using particle swarm optimization (PSO) method", in *International Symposium on Mechatronics and its Applications*, 2009, pp. 1–6.

Curriculum Vitae

Ezequiel Di Mario

ezequiel.dimario@epfl.ch

Education

2010-2015	Ph.D. in Electrical Engineering *École Polytechnique Fédérale de Lausanne (EPFL)*
2004-2009	Engineer's Degree, Electronic Engineering and Telecommunications *Buenos Aires Institute of Technology (ITBA), Argentina*

Experience

2010	Research Assistant *Distributed Intelligent Systems and Algorithms Laboratory (DISAL), EPFL*
2009	Network Operations Center Operator *Global Crossing (now Level 3 Communications), Buenos Aires, Argentina*
2007-2008	Research Assistant *Digital Applied Electronics Group (GEDA), ITBA, Argentina*
2007-2008	Teaching Assistant *Buenos Aires Institute of Technology, ITBA, Argentina*

Publications

1. E. Di Mario and A. Martinoli, "Distributed particle swarm optimization for limited time adaptation with real robots", *Robotica*, vol. 32, no. 02, pp. 193–208, 2014
2. E. Di Mario, I. Navarro, and A. Martinoli, "A distributed noise-resistant particle swarm optimization algorithm for high-dimensional multi-robot learning", in *IEEE International Conference on Robotics and Automation*, 2015, pp. 5970–5976
3. E. Di Mario, I. Navarro, and A. Martinoli, "Distributed particle swarm optimization using optimal computing budget allocation for multi-robot learning", in *IEEE Congress on Evolutionary*

Computation, 2015

4. I. Navarro, E. Di Mario, and A. Martinoli, "Distributed vs centralized particle swarm optimization for learning flocking behaviors", in *European Conference on the Synthesis and Simulation of Living Systems*, 2015, pp. 302–309

5. G. Jornod, E. Di Mario, I. Navarro, and A. Martinoli, "SwarmViz: an open-source visualization tool for particle swarm optimization", in *IEEE Congress on Evolutionary Computation*, 2015

6. E. Di Mario, I. Navarro, and A. Martinoli, "The role of environmental and controller complexity in the distributed optimization of multi-robot obstacle avoidance", in *IEEE International Conference on Robotics and Automation*, 2014, pp. 571–577

7. E. Di Mario, I. Navarro, and A. Martinoli, "Distributed learning of cooperative robotic behaviors using particle swarm optimization", in *International Symposium on Experimental Robotics, to appear in Springer Tracts in Advanced Robotics*, 2014

8. E. Di Mario, I. Navarro, and A. Martinoli, "Analysis of fitness noise in particle swarm optimization: from robotic learning to benchmark functions", *IEEE Congress on Evolutionary Computation*, pp. 2785–2792, 2014

9. E. Di Mario, I. Navarro, and A. Martinoli, "The effect of the environment in the synthesis of robotic controllers: a case study in multi-robot obstacle avoidance using distributed particle swarm optimization", in *European Conference on the Synthesis and Simulation of Living Systems*, MIT Press, 2013, pp. 561–568

10. E. Di Mario, Z. Talebpour, and A. Martinoli, "A comparison of PSO and reinforcement learning for multi-robot obstacle avoidance", in *IEEE Conference on Evolutionary Computation*, vol. 1, 2013, pp. 149–156

11. E. Di Mario and A. Martinoli, "Distributed particle swarm optimization for limited time adaptation in autonomous robots", in *International Symposium on Distributed Autonomous Robotic Systems 2012; Springer Tracts in Advanced Robotics 2014*

12. E. Di Mario, G. Mermoud, M. Mastrangeli, and A. Martinoli, "A trajectory-based calibration method for stochastic motion models", in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 4341–4347

Reviews

- IEEE/RSJ International Conference on Intelligent Robots and Systems
- International Conference on Distributed Autonomous Robotic Systems
- European Conference on Artificial Life
- Robotica (journal)
- Artificial Life (journal)

Honors and Awards

2009	Honor Diploma, Buenos Aires Institute of Technology (ITBA)
2009	ADITBA Award (2nd highest average), Buenos Aires Institute of Technology (ITBA)
2008	Santander Río Bank Academic Achievement Award
2007	Santander Río Bank Academic Achievement Award