To my family

Acknowledgments

My first thanks go to my whole family, especially to my closest relatives, that supported, motivated and endured me during all my studies. A special appreciation goes to all the people in IRIDIA, that made my work much more enjoyable and that gave me precious suggestions (in completely random order: Carlotta, Paola, Mauro, Francisco, Max, Elio, Christos, Vito, Anders, Prasanna, Tom, Alexander, Shervin, Rehan and Roderich). In particular, I'd like to thank Thomas Halva Labella, my co-advisor, for the super-human patience with which he followed and corrected my work, Dr. Marco Dorigo, for the opportunity he gave me to work in the research laboratory, and Prof. Marco Colombetti, who sent me in Belgium and approved my work. I cannot forget all the support I received also from my dearest friends, both from Italy and Belgium: Gianni, Giulio, Franz, Zumba, Silvia, Nicola, Sabrina, Céline, Anne, Laura, Gloria, Giuliano, Bear and all my neighbours in Brussels. I would like to mention here also all the friends that shared my (too) long "adventure" at the Politecnico and with which I spent great leisure time: Diego (C.G.), Helvis, Teo, Pero, Man, Rada, Stefano, Antonio, Gianni, Gadget, Turbina, Boris and Giorgia, Plato and, as special guests, Culio and Nicolas "Tamagotchi" Nick, that I mention for the moments of fun he (involuntarily) gave to me and to many people in IRIDIA.

Sommario

Scopo di questa ricerca è stata la definizione di una metodologia formale per il confronto di differenti strategie di apprendimento automatico, nell'ambito della Swarm Robotics. Questo settore della robotica, sviluppatosi negli ultimi anni, trae ispirazione dal comportamento riscontrato in alcune specie di insetti sociali, come formiche o termiti. Esso prevede l'uso di un gruppo di semplici robot (definito anche sciame o colonia) che possano portare a termine compiti complessi, potenzialmente impossibili per i singoli agenti, senza tuttavia comunicare tra loro.

Per migliorare il coordinamento tra gli individui della colonia in vari ambiti applicativi, sono stati proposti numerosi algoritmi di apprendimento automatico, che permettono di adattare il comportamento del singolo in base agli stimoli diretti da lui percepiti. Al momento in cui scriviamo, non siamo a conoscenza di alcuno studio volto alla formalizzazione di una metodologia che consenta il confronto diretto tra differenti strategie di adattamento, né dell'esistenza di confronti empirici di qualsiasi tipo tra diversi algoritmi nel campo della Swarm Robotics.

Il nostro obiettivo è stato quindi di stabilire una procedura corretta e rigorosa, basata su principi largamente accettati in molti altri campi scientifici, che permetta di valutare le performance di questi algoritmi, risparmiando anche tempo e risorse grazie alla riduzione del numero richiesto di esperimenti con robot reali. Il confronto consente di trarre utili conclusioni per la realizzazione di sistemi di adattamento più efficaci con tempistiche ridotte e costi contenuti.

Il metodo da noi descritto prevede di rincondurre tutti gli algoritmi da valutare a condizioni sperimentali omogenee, in modo tale che il confronto

possa avvenire nella maniera più equa possibile. Ciò comporta la scelta di un campo di test adeguato per la sperimentazione, la definizione del parametro o dei parametri su cui focalizzare l'adattamento e la preparazione di esperimenti che vengano ripetuti variando solo l'algoritmo di apprendimento da testare. In questa fase, è ovviamente anche necessario specificare una misura univoca per valutare le performance degli algoritmi analizzati.

La procedura contempla l'uso di un simulatore, il cui sviluppo segua alcuni principi base indispensabili per garantirne l'affidabilità. Gli esperimenti di confronto vengono quindi effettuati in simulazione. Sui risultati ottenuti si eseguono specifici test statistici, volti a evidenziare eventuali differenze tra le performance degli algoritmi e la loro eventuale entità.

Infine, si devono convalidare i valori ottenuti in simulazione con un numero adeguato di esperimenti condotti con i robot reali. Per far ciò, è sufficiente riprodurre anche nella realtà un sottoinsieme dei test condotti in simulazione che rappresenti un campione statisticamente significativo, ma tale anche da non sprecare le risorse limitate a disposizione. Si eseguono anche su questi ultimi risultati i test statistici impiegati per la simulazione e si confrontano i valori ottenuti nei due ambiti.

Se simulazione e realtà offrono valori coerenti, la validità del simulatore può essere confermata, cosí come le valutazioni da lui supportate per quanto concerne gli algoritmi analizzati.

Abbiamo applicato la metodolgia fin qui brevemente descritta a tre diversi algoritmi di apprendimento, realizzati, nel nostro caso, in ambiti diversi e per scopi leggermente differenti. Come terreno di confronto, abbiamo scelto il problema conosciuto nella letteratura specializzata come *prey retrieval* o *foraging*. Gli agenti coinvolti in questo compito devono individuare, raccogliere e riportare in una specifica posizione particolari oggetti sparsi nell'ambiente. Data una colonia di agenti impegnata in questo compito, il numero di individui che dovrebbero essere coinvolti attivamente nella ricerca dovrebbe adattarsi alla ricchezza dell'ambiente in termini di prede (gli oggetti da recuperare) e alla dimensione dello sciame.

Abbiamo scelto di valutare le performance di tali algoritmi in base alla efficienza della colonia (ossia il rapporto tra il numero di prede recuperate e il costo che tale operazione ha comportato per la colonia stessa), la quantità di oggetti effettivamente recuperati e il grado di specializzazione raggiunto dallo sciame alla fine degli esperimenti.

Abbiamo quindi lievemente adattato, quando necessario, la struttura degli algoritmi per focalizzarne le dinamiche di apprendimento sul tempo che ciascun agente trascorre in inattività. Gran parte dei nostri sforzi sono stati volti alla realizzazione di un simulatore adeguato alle specifiche esigenze del dominio applicativo scelto e dell'implementazione hardware dei robot impiegati.

Agli esperimenti condotti in simulazione, sono stati poi affiancati quelli che prevedono l'uso dei robot reali, tecnicamente più impegnativi e di lunga preparazione. I risultati ottenuti nei due ambiti hanno dimostrato una perfetta coerenza, permettendoci di confermare le differenze tra le performance apprezzate tra i tre algoritmi grazie ai già citati test statistici.

La nostra metodologia formale di confronto si è quindi dimostrata molto efficace e adattabile a differenti ambiti della Swarm Robotics.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

With the increase in interest in robotics, researchers have considered the advantages of using a group of simpler robots (Multi Robot Systems), instead of a single, complex agent, to solve a number of different problems.

This kind of approach leads to systems that are more robust and have a superior fault tolerance to hardware failures. The main drawback to the new robotic conception has proved to be the need of a robust, complex coordination among the agents deployed on the field.

Swarm Robotics offers new solutions to this kind of problems. It takes deep inspiration from biological examples, especially from social insects like ants and bees. These animals show complex collective behaviours even if they use very little direct communication. Only locally available information is exploited and indirect communication is obtained by modifications of the environment (*stigmergy*).

The agents used in Swarm Robotics replicate the simple insects that inspired this new field. They have mostly reactive behaviours, their hardware implementation is kept as simple as possible and they are intended to work only in scalable swarms. In order to improve the performances of these swarms, many adaptation strategies have been proposed, based on dynamic learning.

The complexity of Multi Robot Systems in general, and of Swarm Robotic Systems in particular, makes any comparison among the learning strategies a very hard task.

Our work intends to propose a new analysis methodology that allows different learning strategies to be compared on a common test field. We also want to define specific parameters that could be used to evaluate the performances of each new algorithm.

1.1 Goals

Our research aims to define a methodology to compare different learning strategies designed for Multi Robot Systems and, more specifically, for swarms of robots. We show the effectiveness of our methodology by comparing three different learning algorithms.

The application domain used for the experiments is prey retrieval, also known as foraging. It is a widely studied task in the robotic literature. We decided to focus the learning process on the *Time in Nest* parameter, that is, the time each robot spends in the nest before starting a new prey retrieval trial. We analyse the algorithms in terms of their efficiencies, performances and the degree of specialisation that they can induce in the group of robots.

We tested the algorithms both in simulation and with real robots. Both were intentionally developed for this purpose. The hardware implementation of the agents is made with the MindS-bots, simple robots built with Lego bricks, while we developed our simulator using also a physics library, in order to reproduce in the most accurate way the interactions among agents, prey and environment.

Most of the experiments were carried out in simulation. The simulator can run experiments faster than real time and allows us to test each algorithm in several setups, thus sparing a lot of resources. The results are then validated replicating some experiments also with the real robots.

1.2 Motivations

In the last years, many researchers have focused their attention on Swarm Robotics. This field takes a different approach to traditional Multi Robot Systems and it has received deep inspiration from biological examples. It studies how collectively intelligent behaviours can emerge from complex local interactions among agents and between agents and environment.

Swarm Robotics has been applied to a wide range of tasks and many learning algorithms have been developed as well. The goal was to automatically improve the performances of the agents on a specific task.

Nevertheless, the studies conducted so far, to the best of our knowledge, have never introduced a method to evaluate the performance of the swarm comparing it with other researches. "There are no generally accepted global criteria to evaluate a swarm system's performance" (Rybski et al., 2003).

There are in literature some authors that propose analytical models of Swarm Robotic Systems. (Lerman et al., 2001). Following this approach, it is possible to derive a set of equations and study, for instance, the effect of different parameters and/or algorithms. The major drawback of this approach is that usually one needs to make a number of assumptions and simplifications in order to be able to obtain the equations. Given the importance of the complexity of interactions in Swarm Robotics, it might happen that some important features of the system get lost in the simplification phase.

Our aim is to propose another approach to the evaluation and analysis of Swarm Robotic Systems. Our approach tests the systems "on the field", therefore no previous assumption is required. It can be considered orthogonal to the analytical approach and it introduces more formality in Swarm Robotics research.

1.3 Innovative Contributions

The main contribution of our work is the introduction of a rigorous experimental methodology in the field of Swarm Robotics. To the best of our knowledge, none of the works in literature make use of the concepts that we introduce in this thesis, or at least explicitly mention it. The advantages of this methodology are threefold:

1. It allows the researcher to draw scientifically sounder conclusions.

2. It allows the researcher to spare resources, in terms of time to conduct the experiments and number of experiments.

3. It clearly states which are the practical differences between algorithms.

Sounder conclusions come from the wise application of the correct statistical tests for the analysis of the results. The right use of the tests and the right design of the experiments, can lead to statistically significant differences with fewer experiments. Especially when working with real robots, experiments are the most consuming part of a research. They not only require a lot of time, but they use other resources like people, free rooms for the experiments, and also money (think, for instance, about the maintenance of the robots after several experiments). Finally, the methodology can tell whether an algorithm is better than another, giving precious information to the designer of a swarm robotic system that wants to use it for actual applications.

We limit the application of our methodology to some specific cases, focusing on foraging as test field for our research and evaluating the effectiveness of only three algorithms. But the case study we present is just a possible application of the rigorous procedure that we define. This methodology, we hope, can be used in a wide range of other researches in the Swarm Robotics field, to compare new learning algorithms with the employ of different test fields.

1.4 Outline of the Thesis

In Chapter 2 we describe some important background information and the state of the art about Multi Robot Systems and, in particular, about Swarm Robotics. Chapter 3 introduces the task we chose as test field for our research, discussing also its importance both in nature and in Swarm Robotics studies. The experimental setup we prepared is then presented in Chapter 4, where we discuss about the hardware implementation of our agents and about the simulator we used to expand our research. The three learning algorithms we decided to compare are presented in Chapter 5, as well as the modifications we had to introduce to adapt their structure to a prey retrieval task. Chapter 6 shows the results we obtained both in simulation and with real robots. Statistical tests are used to compare the data we collected and to ensure the soundness of our simulator. Finally, in Chapter 7, we present some concluding remarks and some ideas for future works.

Chapter 2

Multi Robot Systems

During the last years, a number of researchers have considered the problems and the advantages in using a group of robots, instead of a single, and often much more complex unit, in order to achieve one or more specific tasks.

> For some specific robotic tasks, such as exploring an unknown planet, pushing objects or cleaning up toxic waste, it has been suggested that rather than sending one very complex robot to perform the task it would be more effective to send a number of smaller, simpler robots (Dudek et al., 1996).

The advantages of this kind of approach are evident: simpler robots are usually cheaper and the whole system may be more economical, scalable and it may also have a superior fault tolerance compared to more complex and larger robots. Of course, "it is essential that the collective have an overall behaviour or set of actions that accomplishes the same behaviour or action that was required of the single more complex robot" (Dudek et al., 1996).

It is evident that there are tasks that will not take advantages of a multi-agent approach (i.e., when the goal can be reached only through a series of actions that must be done in a specific order, one after the other), but there are many, even in our every day life, that are very suitable for this kind of method.

Tasks that require simultaneous actions in different places, often cannot be accomplished by a single robot, no matter how capable, because it is spatially limited (Cao et al., 1997).

Even when the presence of more than one robot is not strictly required, we can identify highly parallelized tasks that could be solved more efficiently by a Multi Robot System (MRS). This happens when we have an overall improvement if different actions of the task can be performed at the same time by different agents.

For instance, if we were planning to design an automatic system for garbage collection in a town, it would be pointless to create a single robot to achieve the task, while a *colony* or *swarm* of simpler robots could be a more reliable and efficient solution.

There are also other tasks that seem ideally suited to MRS. For instance, Gage (1992) identifies some military applications, such as mine deployment, mine sweeping, surveillance, sentry duty, maintenance inspection, ship hull cleaning and communications relaying, that are characterized by high potential for damage to each single agent involved in the operation. Using a large number of relatively simple, inexpensive, interchangeable, autonomous elements, rather than a complex, very expensive, highly sophisticated unit, the risk of total failure of the task is decreased and thus the reliability of the whole system is increased.

Another example could be the already mentioned planet exploring problem, discussed by Aylett and Barnes (1998). The idea is to use a number of rovers that cooperate with each other in an unknown and hazardous environment.

In MRS, robots usually cooperate to achieve one or more goals, but there are also systems where competition is required, among single elements or different teams. An example of this kind of behaviour can be found in the RoboCup competition, a world tournament where teams of robots play soccer games. This is only one of the many examples that could be found where robots are required to be be competitive, and the study of such field raises many interesting issues, both practical and scientific, as strategy acquisition (to overcome the opponent), real-time reasoning, multi-agent collaboration and sensor-fusion (Kitano et al., 1997).

2.1 A Taxonomy for Multi Robot Systems

The design of a MRS is a very arduous task, because of the size and complexity of the space of the possible designs. Dudek et al. (1996) suggest a taxonomy to classify MRS according to several natural dimensions, that address the characteristics of the collective rather than the architectural details of the single elements:

Size of the Collective The collective can be composed by one single robot, two robots ("the simplest group") or by a number n of robots. Increasing n until $n \gg 1$, we can assume the collective as composed by an infinite number of elements.

Communication Range There could be no communication at all among the elements of the collective or the communication range could be limited (each robots is supposed to communicate with those nearby).
If no bounds exist on the range, then we assume that robots can communicate with any other collective mate. This assumption becomes often impractical when the size of the collective is $\gg 1$.

Communication Topology Designing a collective of robots, four different communication strategies can be identified and used. The elements of the collective can broadcast their messages to all the robots at the same time or they can communicate with each single robot by name or address. If the communication network among the robots is implemented as a general graph or as a tree, then each agent has to respect the communication chain and will be allowed to transmit to those nearby in the chain.

Communication Bandwidth This classification dimension considers the cost of the communication. If the communication is free, we can design robots that behave as if there was a central intelligence, because there is a full information sharing (infinite bandwidth).
The independence of the robots grows when the communication costs increase. Communication may have the same cost of moving the robots

between locations or even a higher cost (for instance, the radio link can require a high amount of electrical power because the robots are far from each other). When robots are unable to sense each other, then the bandwidth is reduced to zero.

Collective Reconfigurability The spatial arrangement of the robots can be static (elements of the collective have fixed positions and they move without changing their relative placement), coordinated using communication or dynamic with arbitrary changes.

Processing Ability This dimension considers the model of computation of each collective unit. It is useful to note that the entire collective can have an overall computational ability more powerful that the ones of each single component.
The controller of each robot can be modeled as a simple neural network, a finite state automaton, a push-down automaton or as a Turing machine equivalent.

Collective Composition The elements of the collective can be identical, which means they have the same physical and behavioral characteristics, homogeneous (or physically uniform) or heterogeneous.

The described taxonomy "serves the dual functions of allowing concise description of the key characteristics of different collectives, and describing the extent of the space of possible designs" (Dudek et al., 1996, p. 26).

In our work, the collective is composed by identical robots and it has a limited size, even if the final aim should be the use of a *swarm*. No *direct* communication will be allowed among elements, so communication topology and bandwidth will not be considered. The topological reconfigurability is dynamic and not coordinated. The behaviour of each robot can be modeled as a finite state automaton.

We focus our attention on *cooperative robotics*, i.e., where there is no competition among robots, and more specifically on the task of prey retrieval, also known in literature as *foraging*. This is considered one of the canonical domains for Multi Robot Systems (Cao et al., 1997) and "it consists of searching for objects in the environment and bringing them to a region called *home* or *nest*" (Labella, 2003, p. 13).

2.2 An Overview on Multi Robot Systems

In order to analyze the issues related to MRS, we can start studying the problems that arise in a single agent approach, characterizing also the environment where our robots will act.

The main objective of a robot is to perform one or more tasks. That means the necessity for its controller to lead it to a goal state, given a starting condition of both the robot and the environment where the robot has to operate. The controller will perceive its environment through sensors and will act upon that environment through actuators, changing it if necessary.

We can describe the environment that surrounds the robot using six independent dimensions, as proposed by Russel and Norvig (2003):

Fully observable *vs.* partially observable If the agent is able to perceive every relevant aspect of the environment at each point in time, then we can consider the environment as fully observable. A partial observability might come from noisy sensors or from sensors' limitations.

Deterministic *vs.* stochastic We label an environment as deterministic, when the next state is completely determined by the current state and the action executed by the agent.

Episodic *vs.* sequential If the environment is episodic, each action of the agent does not depend on the previous ones.

Static *vs.* dynamic An environment is dynamic for an agent, if it changes while the agent is deliberating.

Discrete *vs.* continuous This dimension can be applied to the state of the environment, to the perceptions and actions of the agent and to the way time is handled.

Single agent *vs.* multi-agent This dimension is defined by the number of agents acting in the environment.

Our robots must move in a non deterministic world and their sensors are subject to error and noise. This means that the exact behaviour of one single agent is difficult to foretell, even when the controller is made implementing the simplest of algorithms (Matarić, 1995).

When we have more than one robot, then, the complexity of the system increases, so every accurate prediction of future states is impossible.

From point of view of each robot, the actions of the other agents make the environment even more dynamic, changing it continuously. Even when communication among robots is allowed, noisy or limited sensors lead to an important lack of information. Then, the controller of a single agent has to take decisions with a partial knowledge of the overall situation.

We can deduce that, when we consider Multi Robot Systems, we have to deal with a partially observable, sequential, continuous, multi agent, dynamic and usually stochastic environment.

We can broadly distinguish two approaches to the decision process (Labella, 2003). A controller can be *reactive*, when it considers only the current state and binds it to a specific action, or based on a *planning* approach. In the second case, the best action is selected according to predictions of future states, evaluated using a model of the environment.

The planning approach, even if sometimes more efficient in simple environments or for simple tasks, becomes harder to implement when we have to deal with complex situations, with more than one task to be performed in a specific order by different agents and with noisy information. Interactions among robots, the possibility of failure of both the agents and the communication channel (if any) contribute to create possible inconsistent predictions of the future states.

We can find some drawbacks also in a fully reactive control system. For instance, when we have to deal with a small group of agents and with a complex task in a known environment, it can be inefficient and even inadequate in accomplishing some specific goals that require some planning or cooperation.

Both approaches are still widely used in mobile robotics. In our work we will mainly focus on the reactive one, but good examples of the planning strategy can be found in a number of real applications, such as the robot navigation system proposed by Bruce and Veloso (2002). It uses rapidly-exploring random trees (RRTs) to elaborate paths for the robots and the results show a good performance of the algorithm, making it suitable for real time and constrained applications, such as the already mentioned RoboCup.

Matarić (1995) identifies *interference* as a problem directly related to MRS. When the agents share the same goals, then the interference arises as competition for shared resources (*resource competition*). On the other hand, when robots' goals differ, more complex conflicts can arise, such as deadlocks (*goal competition*).

At the present time, there is a vast literature about problems that could be solved in a more efficient way with MRS rather than with a single robot. The work done in the field of cooperative mobile robotics can be essentially categorized in two groups: *swarm-type* approach, that will be widely described in the next section, and robots that use intentional cooperation (Parker, 1998).

While both of them deal with achieving a strict synergy among robots in order to perform some (sometimes sequentially related) tasks, they take very different approaches to reach their goals.

In Swarm Robotics, the actions of the agents are triggered by robot-robot and robot-environment interactions, exploiting only local information. On the other hand, traditional robotics usually uses approaches borrowed from operations research, such as planning, and the overall coordination if often obtained thanks to a direct communication channel among the agents.

Cao et al. (1997) identify three specific tasks that are now traditionally associated with the research related to MRS: *traffic control*, *box pushing* (or *cooperative manipulation*) and *foraging*. Such tasks are particularly useful in this field because they consider problems that do not arise with a single robot and we can identify a number of real tasks strictly related to the testing ones (for instance, an algorithm elaborated to solve a foraging problem can be adapted for the garbage collection in a town).

With a growing number of robots working on the same field and the requirement of a strong coordination, the issue of task allocation has become the focus of many interesting works. Gerkey and Matarić (2004), in a recent publication, elaborate a deep analysis of the problem in Multi Robot Systems, using theory from operations research and combinatorial optimization both in the understanding of existing approaches and in the elaboration of new ones.

Another important issue on which the research has focused is learning. Learning is a major topic with single robots, but it is more challenging when there are more agents acting at the same time, that need to know if their behaviour is leading the group to the solution of the overall problem, having only a partial knowledge of the situation.

When a centralized store of complete world knowledge is not available, what we have to solve is a (often confounding) credit assignment problem, that is "the problem of properly assigning credit or blame for overall performance changes (increase and decrease) to each of the system activities that contributed to that changes" (Weiß, 1995).

Matarić (1997) addresses the issue in a foraging problem, where the objects to be collected are pucks deployed in a squared arena. The decision process uses behaviours, activated by the arising of specific conditions, as the basic representation level for control and learning. The learning is then obtained in two ways: heterogeneous reward functions and progress estimators. The reward functions, in particular, are composed by three main sub functions. The first one considers internal events triggered by the accomplishment of a behaviour (i.e., a puck is dropped at home), the second one takes care of the distance from the other robots and the third one measures the progress in the retrieving of a puck after it has been grasped. This approach only uses local information available to the robot thanks to its sensors and no communication is implemented to check the situation of other agents.

A totally different approach to the learning problem can be taken when communication is allowed among the robots. In this case, the behaviour of the agents on the field is equally determined by the internal status of each robot and by the information it gets from the teammates around it. A more deep knowledge of the overall situation is then obtained, even if we cannot exclude possible errors and sensor noise. The architecture ALLIANCE (Parker, 1998) and its extension, L-ALLIANCE (Parker, 1997), use this kind of approach to better coordinate the heterogeneous robots deployed on the field and to let them learn by comparing their results in accomplishing some specific tasks with the ones of the teammates.

For years, the simultaneous use of more than one robot and the related problem of cooperation among agents has been faced as an extension of the single robot approach. Matarić (1998) suggests that this is not the best way to address the issue, proposing a new way to design the robots' control systems, with which the overall desired behaviour is obtained as a result of the interaction dynamics between the robots and their environment.

In the next section we discuss the part of the work done in multi-robotics where the interactions among agents are not regarded as problems to face, but they are instead exploited to solve the proposed task.

2.3 Swarm Robotics

Swarm Robotics takes a different approach to MRS. It receives deep inspiration from biological examples, especially from social insects like ants, bees and termites (Bonabeau et al., 1999) and it can "be loosely defined as the study of how collectively intelligent behaviours can emerge from local interactions of a large number of relatively simple physically embodied agents" (Dorigo and Şahin, 2004).

Nature is rich of examples of relatively simple animals that cooperate efficiently to complete tasks impossible for the single unit. Ants are probably the most studied and cited one: they can carry big prey thanks to coordinate efforts, find the shortest path to the nearest food source or build huge nests. Also bees show great aptitude in exploiting the best food source, while the ability of the termites to build complex nests is well known to the biologists.

It can be observed that those animals use very little direct communication (if any), even if they carry on works in huge groups. They use only locally available information, exploit features present in the environment and use

Figure 2.1: Honey bees are one of the best examples in nature of social insects with high self-organization capabilities.

indirect communication.

A particular form of indirect communication, used by social insect to coordinate their work, is *stigmergy*. This term was first introduced by Grassé (1959) in a study that focused on two species of termites, *Bellicositermites natalensis* and *Cubitermes*, and their behaviours during the complex activity of nest building. It indicates the form of indirect communication obtained by modifications of the environment. The use of the term was later extended also to other social insects (Theraulaz and Bonabeau, 1999), such as ants and wasps.

Using this natural strategy, social insects can accomplish tasks that transcend the abilities of individuals, even when the environment is noisy and no global communication is allowed or possible.

The goal of the swarm robotics research is to design controllers inspired by the collective behaviour of such social insect colonies, in order to obtain robust MRS, capable of operating even in a noisy environment and exploiting only local information. Swarm Robotics does not assume each robot as

Figure 2.2: A swarm of leaf-cutting ants retrieving pieces of leaves back to their nest.

a standalone independent unit, but, on the contrary, it assumes that the mission is the result of a joint action of simple agents.

The challenge introduced by this approach is about finding simple behaviours based on local knowledge that can produce complex global patterns. Answering to this question for a specific problem can lead to a better understanding of the natural phenomena and, consequently, to an improvement of the artificial system.

We can find three main advantages in using swarm of robots instead of classic MRS. First of all, scalability. Since our agents exploit only local information in their decision process and they are independent, the control architecture of each single robot is kept the same, no matter how many units we intend to use. Second, the system will be flexible. Changing the size of the swarm during the task execution should not affect the behaviour of other units. Third, robustness. The design of the robots will be more minimalistic than in complex MRS and, even more important, the redundancy of the abilities of each unit will lead to a greater fault-tolerance.

Control algorithms in Swarm Robotics must be kept as simple as possible. Complex representations of the environment are usually not needed, because the achievement of a task is based on simple interactions triggered by reactive behaviours. The use of probabilistic decisions is also common and learning has proved to be a good way to increase the efficiency of the collaboration among units, as shown by Labella (2004, 2003).

Ijspeert et al. (2001) address a task that requires strict collaboration among robots. It consists in finding and pulling out from the ground of a circular arena a number of sticks. Because of the length of each stick, a single robot is not capable to accomplish the task working alone. The paper shows that even with limited sensor capabilities and without any direct communication among the agents, the task can be accomplished, if the parameters that lead the robots' behaviours are rightly tuned. For this experiment also a simulator and a probabilistic model have been used. The first is a physical, sensor-based simulation of realistic robots (Webots), while the second is a model that is intended to represent the dynamics of a group of robots as a series of stochastic events. The results obtained with those two models have been validated with real robot experiments and it has been showed that even simple simulations can provide very good predictions on the real behaviour of the swarm.

In his Master Thesis, Li (2002) continues the analysis on the strictly collaborative problem of the stick pulling experiment, introducing distributed learning to better tune the parameters that trigger the behaviours of each robot. Li et al. (2004) study the emergent specialization in a swarm of robots where each unit starts with homogeneous parameters, using a distributed learning system and measuring the improvement in task accomplishment as a consequence of the specialization.

Rybski et al. (2003) evaluate the performance of a swarm of robots on foraging experiments. In the paper, the efficacy of using localization instead of random walk is tested, as well as the use of communication versus random walk without communication. Only real robots are employed, because the authors deny the usefulness of a simulated environment. This approach differs from ours, as we show in the next chapters.

2.3.1 The Swarm-Bots Project

To better understand the issues and the goals of Swarm Robotics, in this section we briefly describe one of the best projects developed in this field.

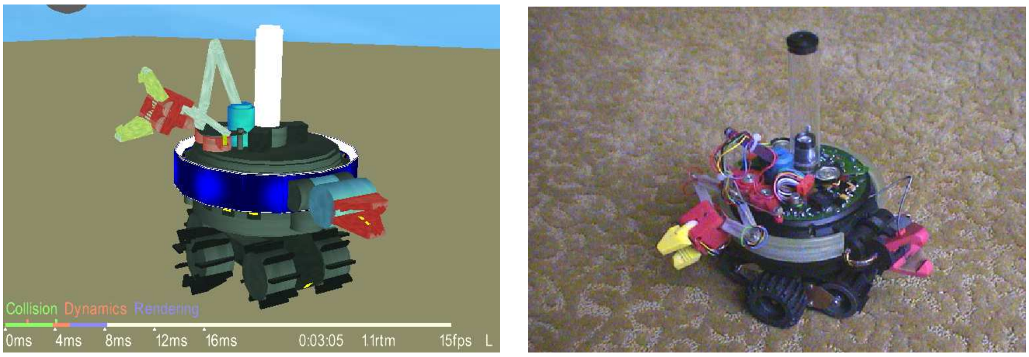
Swarm-Bots was a project, sponsored by the Future and Emerging Technologies program of the European Community and coordinated by Dr. Marco Dorigo, that lasted 42 months and was successfully completed on March 31, 2005. At the time we write, the project is considered the state of the art in the field of self-assembling, self-organizing autonomous robotics.

The aim of the project was the study of a new approach to the design and the implementation of self-organising and self-assembling artifacts, inspired by recent researches on *swarm intelligence* shown by social insects.

The goal, fully reached at the end of the project, was the design and the hardware implementation of a *swarm-bot*, a metamorphic robotic system composed by a number of smaller devices, the *s-bots*. Each *s-bot* is a self-contained module, capable of independent movements and with its own control system. The behaviour of the *swarm-bot* is the result of the interactions among the simple independent entities. Collaboration is achieved by mean of indirect and non-symbolic communication.

The prototype of such robotic system had then to accomplish a number of tasks and to show some particular features, such as the ability to self-assemble into different geometric configurations, the possibility to move on rough terrain and the capability to adapt to match environmental variability.

Mondada et al. (2002b) introduce the principal ideas behind the project, illustrating the *swarm-bot* main concept, the inspiration from self-organizing capabilities of social animals and a first mechanical concept of each *s-bot*.


Collision Dynamics Rendering
0ms 4ms 8ms 12ms 16ms 0:03:05 1.1rtm 15fps L

(a)

(b)

Figure 2.3: Figure 2.3(a) shows a detailed simulated version of an *s-bot*. Figure 2.3(b) is a picture of the hardware implementation of an *s-bot* prototype.

The authors focus also on the need of a simulated environment, important to make possible the first studies on the behaviour of the robotic system.

Şahin et al. (2002) describe a first imaginary scenario where the *s-bots* are challenged with a task that requires dynamic shape formation, navigation on rough terrain and the capability of pushing/pulling heavy objects. Preliminary results on the formation of patterns are also presented, obtained with a newly developed simulator.

Mondada et al. (2002a) delineate more in detail the mechanical concept of the future *s-bots*, illustrating sensors, interconnections and control electronics needed to build each single unit of the *swarm*.

The evolution of the project has been accompanied by a number of publications that explained the results reached.

Mondada et al. (2003), for instance, present the first *s-bot* working prototypes from a mechatronic prospective. The result is an autonomous small unit, equipped with a drive system obtained by the combination of tracks and wheels and with two grippers (one rigid, the second placed at the end of a semi-flexible arm) for physical interconnections. Each *s-bot* has been also loaded with a number of sensors, like an omnidirectional camera, four microphones, proximity sensors, both around and at the bottom of the robot, humidity and temperature sensors, optical barriers on grippers and a three axis accelerometer.

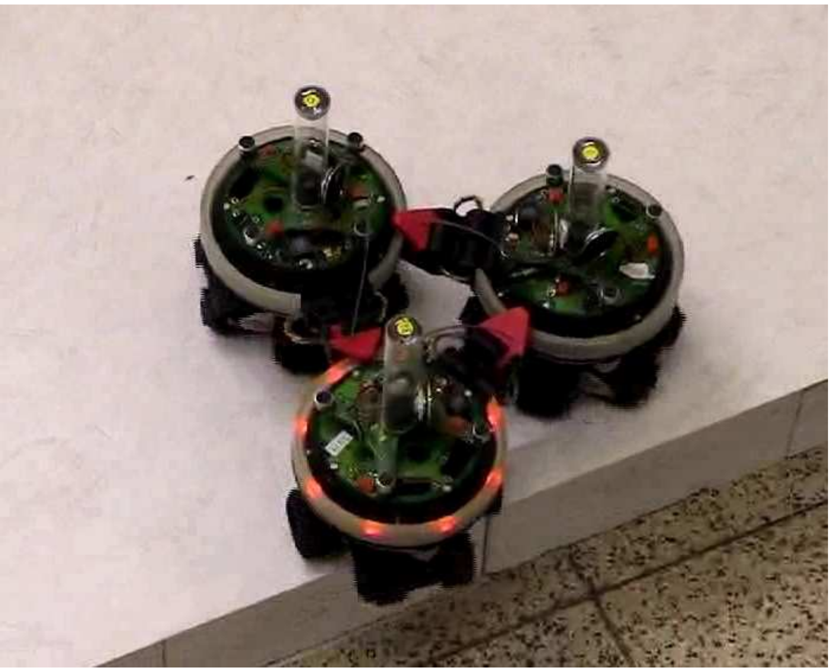


Figure 2.4: Three *s-bots* navigating in an arena with open borders. The red lights around one of the robots indicate the emission of a tone to communicate the presence of a hole.

Trianni et al. (2004) focus on the software development, and in particular on the progresses made with artificial evolution to synthesize the controller for the single agents in order to obtain coordinated motion and an efficient solution to the hole avoidance task.

Also the prey retrieval task is used as a test field for the new robotic system. Labella et al. (2004a) address this issue, studying a simple adaptation mechanism to increase the group efficiency. The algorithm, inspired by ants' behaviour and based only on local information available to each robot, is designed to minimize the interferences between agents during the execution of the task.

Our work is strictly related to the research done on prey retrieval in the frame of the *Swarm-Bots* project. The robots used to perform our experiments are less sophisticated than the *s-bots*, but this is uninfluential for our purposes, as we see in the next chapters.

Chapter 3

Prey Retrieval

The aim of our work, as we already said, is to provide a methodology to compare the performances of different learning algorithms presented in distinct robotic researches. To the best of our knowledge, a comparison has not been possible so far, because each algorithm has been studied in a particular environment with some specific goals.

Looking for a task suitable for our experiments, our choice falls on *prey retrieval*, a traditional test field for Multi Robot Systems, deep inspired from biological systems and possibly useful for a number of real applications.

3.1 Prey Retrieval in Biology

Prey retrieval, also known as foraging, is a task common to a wide range of different species of social insects. An efficient division of labor among the nest-mates is essential to guarantee to the colony the necessary amount of food without waisting time and precious resources. The mechanisms that govern the emergence of some of the behaviours observed during prey scavenging or the search for building material remain unclear.

Nevertheless, a number of studies have been conducted over different species, with a particular interest for ants. The general behaviour of ant foragers is efficient but rather simple. When they find a prey, after a random search in the environment, they try to carry it to the nest. If the prey is too heavy, then they can either recruit nest mates with chemical signals, or cut the prey and retrieve smaller parts of it.

The use of chemical trails, as well as recruitment in the nest, have been observed in many species. Some of the problems about coordination of the ants and their decision process are often poorly understood. What seems evident is that the global cooperation in a colony emerges from simple interactions among individuals and between individuals and the prey. For instance, coordination in collective transport seems to occur also through the item transported. The actions of the ants performing prey retrieval seem to generate new stimuli perceived by other team mates that react continuing the task and generating information to the other group members (Labella, 2003). We can suppose that the simple fact that the ants perceive the movement of the prey indicates that the force applied is sufficient and that no other foragers are needed. This is an example of the already mentioned *stigmergy.*

Some ant species, such as *Pheidole pallidula* or *Lasius niger*, provide good models for the understanding of prey retrieval. In particular, the studies carried on the *Lasius niger* (Mailleux et al., 2003; Portha et al., 2002; Devigne and Detrain, 2002; Mailleux et al., 2000) investigate how those ants perform environmental exploration and how works the process of trail recruitment, according to the amount and quantity of food sources around the nest.

Ant colonies show also an effective division of labor among workers and a good adaptation to changes in the environment.

Deneubourg et al. (1987) developed a simple mathematical model of learning. The model is capable to reproduce the individual foraging pattern observed in ants (*Pachycondyla apicalis*) or in bumblebees. The model was developed observing that a *Pachycondyla apicalis* forager that returns to the nest retrieving a prey rests in the nest less time than other foragers that return without a prey. The authors proposed a learning mechanism as a way to get social organization. In total absence of communication among workers, a simple learning strategy can lead to an efficient distribution of foragers in the environment. Moreover, it can also lead to a balanced division of initially identical potential foragers into highly active and largely inactive ones.

3.2 Swarm Robotics and Prey Retrieval

Foraging is widely considered a canonical test domain for collective robotics (Cao et al., 1997). It involves exploration of an environment, detection of objects of interest and retrieval of those objects to a specific location (the nest). Moreover, it is a good prototype for actual useful tasks, such as locating and marking land mines, distributed mapping of the area, collective surveillance and many more.

As the social insects from which they take inspiration, the robots composing a swarm have only local sensing and communication capabilities. Then, the adaptation of the agents must be left to a control system capable to adapt the simple behaviours of each agent according only to local available information.

Foraging is a good application domain for testing the interactions among the agents on the field and their behaviour in unpredictable situations. It has already been used in many swarm robotic research works as test field for adaptation processes (Groß, 2003; Labella, 2003).

Since our goal is to describe a formal method to compare learning algorithms for swarm of robots, foraging is the most suitable task to be employed in our experiments.

3.2.1 The Division of Labor

Swarm Robotics, in order to improve the scalability of the system, do not use a central knowledge unit where agents can find global information about the environment and the amount of prey available to be retrieved. Therefore, there is the need of a mechanism that allows the agents to autonomously adapt to the overall situation, to exploit in the most efficient way the environmental resources and to reduce negative interferences among them.

One of the most challenging issues that arise in such situation is how to tune the division of labor among the agents. Task allocation is effective in exploiting mechanical differences among the robots, introducing specialization in the robot activities (Labella et al., 2004b). This way, the elements of the swarm can autonomously try to determine the optimal size of the group that cooperate in a foraging application.

What Robots Can Learn: The *Time in Nest* Parameter

In order to properly tune the number of *foragers* (that is, the robots that actively search for prey in the environment), we use a method inspired by biology that exploits positive and negative feed-backs as usually done by self-organized systems (Labella et al., 2004b; Camazine et al., 2001).

This method does not require direct communication among the agents or any kind of human intervention. The learning process is focused on one single parameter, that is, the *Time in Nest*.

According to the success or failure of each prey retrieval trial, each robot adjusts autonomously its *Time in Nest* parameter. In other words, the elements of the colony decide how long they should stay in the nest before trying again to search for prey. A robot with a short resting time is part of the group of the *foragers*, while the agents that rest in the nest for longer periods are considered as *loafers* (that is, elements not actively involved in the retrieval task).

If the dynamics of the method are quite simple, it is not obvious how the robots learn and according to which rules they change their behaviour. For instance, Labella (2003) proposed a probabilistic method to improve the task allocation among agents, as we see in Section 5.1. Other learning algorithms could directly fix the time the elements of the swarm must wait before starting another trial.

We intend to compare different learning strategies and to evaluate the effectiveness of different ways to tune the *Time in Nest* parameter.

Chapter 4

The Experimental Setup

We decided to use both real and simulated robots. This choice is motivated by the constraints we have performing experiments with real agents. For instance, the number of robots we can use in a single experiment is limited to the ones we have, while in simulation we can extend the colony beyond these constraints. A reliable simulator, as it is explained in Sections 4.3.3 and 4.3.4, allows us to explore a wider range of different setups and to conduct our research in more various situations. This chapter does not detail how we validate the results obtained with the simulator. This topic is discussed in Section 6.3.

In the following, we first introduce the methodology we set up to have a fair comparison among algorithms, then the hardware we chose for our experiments with real robots. We explain also the procedure we use to perform the experiments. Finally, we describe the simulator we coded to complete our work.

4.1 The Research Methodology

The focus of our research is to compare learning algorithms, evaluating their performances on a common test field we already chose (see Chapter 3). Therefore, we must test all the strategies under the same conditions and minimize all the possible random fluctuations. This allows us to directly correlate the results of our research to the effectiveness of each learning algorithm, because it is the only aspect that varies among experiments.

It is important to notice that we have no need of a particularly advanced hardware technology. If all the strategies are tested under the same conditions, using the same robots, the results and the adaptation process of the agents are likely to be dependent only on the learning algorithm used. In fact, any problem related to the robots affects all the experiments and disturbs equally all the strategies tested.

To classify simulated and real experiments, we define two different variables, *size* of the swarm and *prey probability*. The *size* of the swarm is a self-explaining variable, that indicates the number of robots involved in a single experiment. The *prey probability*, on the other hand, is the probability of a prey to appear at each second of the experiment in a random position of the arena. Modifying this variable, we change the rate at which the environment is filled by prey and, consequently, the number of *foragers* needed to collect as many prey as possible. Each combination of these two variables identifies a different experimental setup.

In order to make the comparison among the learning algorithms as fair as possible, we decide to apply another important procedure, both in real and in simulated experiments. This procedure is called *blocking design* (Montgomery, 2000). It consists in arranging the experimental units in groups (blocks) which are similar one to another. It leads to a reduction of the variance of the results.

To implement the blocking design, we coded a small program that accepts as parameter the *prey probability* and randomly generates a file (an *instance*) where there is indicated the number of prey that have to appear during the experiment, their timing and the exact location where they have to be placed. Using every instance only once for each algorithm and employing the same instances we use with real robots also in simulation, we reduce the influence of random fluctuations in prey generation on the results of our experiments.

Before preparing the setting for our research with both real and simulated robots, we must decide exactly how to measure the goodness of each algorithm we compare. We want to know how many prey the colony captures during each experiment and we intend to compare this score with the total number of prey appeared.

We want also to determine if the agents of the colony become specialized and what is the cost for the whole colony of the prey captured during the experiment. The cost can be expressed by the *colony duty time*, that means the sum of the times each agent spends outside the nest, searching for prey.

A good measure for the income-cost ratio can be found in the work of Labella (2003). We adopt the *efficiency* value defined in the mentioned paper as the main parameter to evaluate our algorithms.

$$\eta = \frac{\text{number of retrieved prey}}{\text{colony duty time}} \tag{4.1}$$

We wrote a program to parse the log files generated both in simulation and with real robots, to summarize the results and to calculate the *efficiency* value.

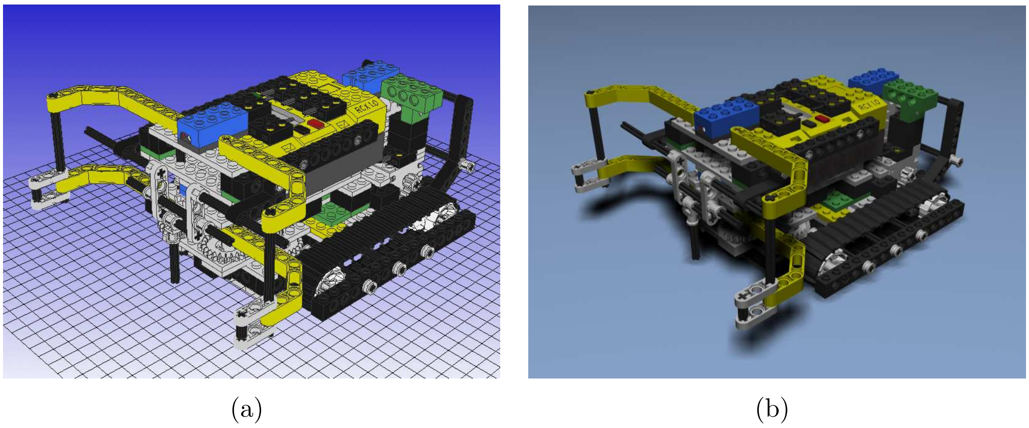
4.2 The Real Robot Environment

This section introduces the MindS-bots, the robots we chose to employ in our work, as well as the environment where they act.

4.2.1 The MindS-Bots

The MindS-bots have been built using the Lego Mindstorm^{TM}, a line of Lego products that provide a standard set of Lego Technic^{TM} bricks and some more special pieces, such as motors, light sensors and bumpers, useful to build simple but properly working robots.

The main block of a Mindstorm^{TM} box is the RCX, that contains an Hitachi H8300-HMS 1 MHz microprocessor, 32 KB of RAM, a Read Only Memory, 6 slots for AA batteries and a beeper. On the upper side of the RCX there are six special brick slots, used to connect the main robot body to actuators and sensors, four buttons and a Liquid Crystals Display, that we used for debugging purposes. In the front side of the block is positioned



(a)

(b)

Figure 4.1: Figure 4.1(a) shows the model of a MindS-bot obtained with Leocad, a CAD specifically designed to reproduce Lego bricks. Figure 4.1(b) is the rendered version of the same model.

an infrared receiver/transmitter that allows the robot to transmit data to a special Lego USB Tower and then to a PC.

The operating system provided with each Mindstorm^{TM} box has been made for simple entertainment and it is not useful for our purposes. Instead of the original one, we chose to use BrickOS.[1] This software embodies a cross-compiler for the standard C language and supports priority-based preemptive multitasking, a programming paradigm widely used in our code.

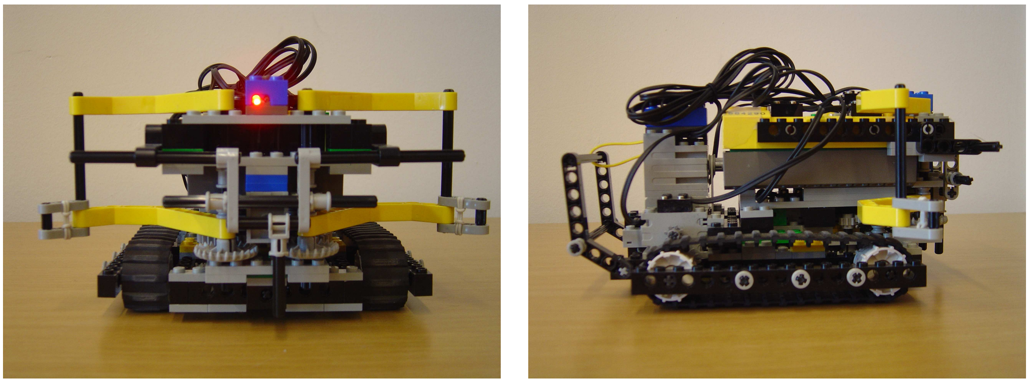
When the RCX is turned on, the default program in the Read Only Memory is executed. It loads the operating system using the infrared interface. Then, once BrickOS is loaded and executed on the RCX, it is possible to upload to the robot the program we want to run.

At the end of each experiment, using the infrared interface and the Lego USB Tower, it is then possible to download from the robot to a common PC the data we need for our analysis.

The MindS-bots we use in our work have been designed by Labella (2003) and employed in his research. In Figure 4.1(a) is shown the model of a MindS-bot, generated with Leocad,[2] a CAD specifically designed to reproduce Lego bricks.

[1] http://brickos.sourceforge.net

[2] http://www.leocad.org



(a) (b)

Figure 4.2: Pictures of a MindS-bot. Figure 4.2(a) shows a front view of the robot. The light sensor is active, as it is used during experiments. In the side view (Figure 4.2(b)) we can see the two motors that move the tracks and, on top of them, the third motor, linked to the gripper's arms by an elastic ring and a Lego axle.

Each robot has three motors and four sensors. The motors are activated independently and they control the two tracks, used for the movement of the robot, and the gripper. The latter is composed by two arms, placed in the front part of the main body, that have a synchronized movement.

The sensor set of each robot consists of two bumpers and two light sensors. The bumpers are situated in the front and in the back of the main body. Each of them is connected to a structure assembled with Lego axles that enhances the obstacle detection.

The front light sensor, placed on top of the gripper, is employed to detect the presence of a prey and to estimate its proximity. The back light sensor helps the robot to find its way back to the nest (the nest is marked by a light source in the middle of the arena, as we see later on in the chapter).

The front and side view of a real MindS-bot are shown in Figure 4.2.

4.2.2 Arena and Prey

The environment in which our robots move is a circular arena with a diameter of 240 cm. For the geometry of the arena, we chose a circle instead of a square to guarantee a perfect symmetry. A symmetric shape of the test ground should eliminate any possible “privileged zone” during experiments. The existence of such places could influence the results of the research, affecting the fairness of the comparison among the learning algorithms.

The inner part of the arena represents the nest, where the robots are placed at the beginning of each experiment and where they have to bring the captured prey. No prey can be placed inside the nest and when a robot completes its task and deposits a prey inside the drop area, the retrieved object is immediately removed.

A light source is positioned in the exact center of the arena. At the end of a trial, successful or not, the MindS-bots can then find the nest using their back light sensors.

To reduce the erroneous readings of the light sensors of the robots, the floor of the arena, as well as the walls that surround it, have been painted in white.

The real representations of the prey are plastic cylinders covered by black curly cotton and partially filled with water. Their color, contrasting with the one of the arena, allows the front light sensors of the MindS-bots to properly identify the prey.

4.2.3 The Experiments

For each experimental setup (that is, a combination of the two variables, *size* of the colony and *prey probability*), we perform a fixed number of experiments, repeating every one of them with the three different algorithms we consider.

Since we want to highlight any difference among the learning strategies, we have to perform a number of experiments sufficient to show statistically relevant differences. We found out that five runs were sufficient for our purposes, thanks to the care we took in reducing external influences.

We fixed the length of each experiment at 3600 seconds (one hour). At the end of that period, the robots reach a steady state and we can then analyze the results using the log recorded in each agent. This log keeps trace of the main events occurred to the robot during the experiment (exit from the nest, prey successfully retrieved, etc.) and it can be downloaded to a

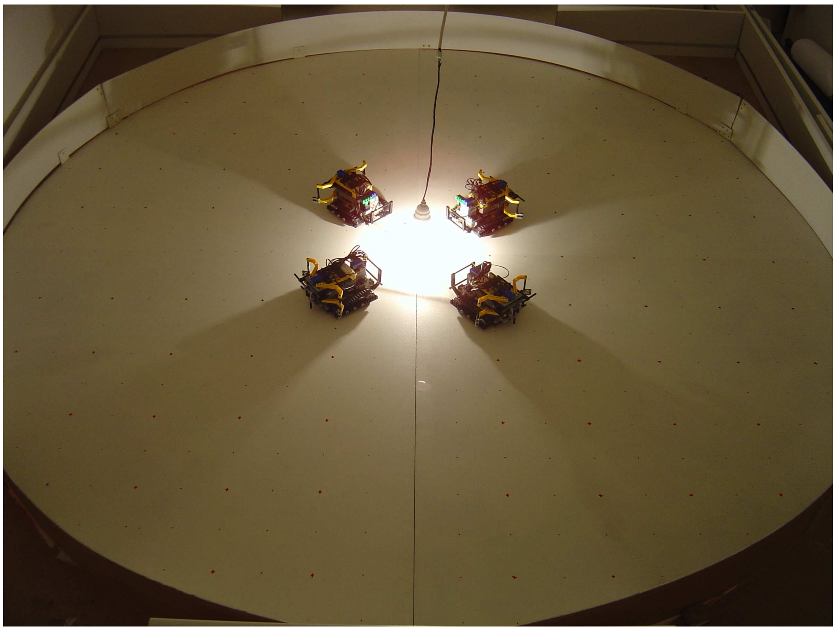


Figure 4.3: At the beginning of each experiment, the MindS-bots are placed on the inner border of the nest. They have a symmetric disposition and the light source marking the center of the arena is behind them.

common PC using a specific software and the Lego USB Tower we already mentioned.

The robots start each experiment in a fixed position, as shown in Figure 4.3. They are placed in the nest, in a symmetric disposition, giving their backs to the light source that marks the center of the arena.

When the starting signal is given (it is necessary to activate each robot by means of a *run* button placed on their upper side), another program running on a common PC is launched as well. This small program starts a counter and parse the file describing the prey generation (instance) for the running experiment. When a prey has to be placed, a sound alert is played and its coordinates are displayed. The prey is then placed by hand inside the arena, as indicated. Should be noted that there might be some delays and small misplacements of the prey due to human intervention. The effects of these errors are however counterbalanced by having always the same experimenter for all the instances. In this way, her/his errors influence equally all the algorithms.

Prey removal is performed as well. A prey is taken away from the arena when it has been dropped inside the nest by a robot after a successful retrieval.

The MindS-bots communicate their failure (when they give up), their success and the act of leaving the nest in two ways. The first one is evident during the experiments. They play three different music pieces according to the event they have to signal. The second way is to record the event in their logs, that are analyzed only at the end of each experiment and then compared to the ones of the other robots.

Not all the experiments are considered as successful and then included in the analysis. When a major failure of a robot occurs, the experiment is halted and then started again from the beginning. We adopt this procedure because we focus on the efficiency of the learning strategies and not on the hardware implementation of the robots and their reliability. A major hardware failure of an agent could influence the results of the experiment, even if the mechanical factor is not related to our topic of research.

The experiments we perform are time consuming. Each experiment requires at least one hour of work and, after this period, additional time must be spent to download the log files from the robots to a PC, to recharge the batteries and to prepare the environment for a new start up. Moreover, the failed experiments must be discarded and repeated.

Considering the amount of time needed to accomplish one single successful experiment, we limit the study on the real robots to two different setups. The choice of the two setups has been made considering many factors, such as the number of agents at our disposal and, even more important, the first results obtained in simulation. We recreate in the real environment the situations where the differences among the algorithms are emphasized.

In the first setup, we set the *size* of the colony to four agents and the *prey probability* to 0.005 (it means that a prey has probability 0.005 to appear each second of the experiment and that the average value of prey appeared in one hour is eighteen).

In the second one, we set the *prey probability* to 0.01 (that means an average value of prey appeared in one hour equal to thirty-six) and we hold the value of the *size* of the swarm to four agents. An increment of the *size* to six elements would lead to a huge number of unacceptable experiments, because of the higher risk of hardware major failures.

As already said, we perform five successful experiments for each learning algorithm. Such amount is sufficient to have statistically significant differences, that are used to validate the results obtained with the simulator.

4.3 The MindS-miss Simulator

It is our opinion that simulations can be a useful tool to investigate the dynamics of a swarm, if the simulator is reliable and its results are validated with a sufficient number of real robot experiments.

With a simulator, we can investigate interesting setups we cannot analyze in laboratory. As in biology, in a swarm-type approach to a problem, the number of agents involved can be really huge. The number of hardware-implemented agents at our disposal is limited as well as their robustness, while in a virtual environment we can increase the amount of agents in the colony without reducing its reliability.

Another important issue that led us to the use of simulations is that they are a fast way to conduct experiments, because they do not require the continuous attention of an operator and they can be run in parallel on different machines.

In the past years, many critiques have been appointed to the use of simulators in the robotic research field. According to some authors, the use of a simulator was dangerous, because it does not reproduce the world where the real robots are supposed to act, but it reflects the programmer's beliefs about the world itself. Moreover, control programs that work in a simulation probably will not work when applied to real robots, because of the differences between the simulated actuators and sensors and the real ones (Brooks, 1992).

A new concept of simulator was then introduced by Jakobi et al. (1995). The authors focused on the development of control systems for autonomous robots through the use of artificial evolution in simulation, that would be later employed on hardware-implemented agents.

Some important guidelines for the realization of a reliable simulator emerge from their work. The authors assert that a simulation must be designed using empirical information measured in real world. More important, it has to be regularly validated by data produced by real experiments. Artificial noise should be used at each level to emulate erroneous sensor readings and environmental disturbs, in order to obtain sufficiently robust programs.

Another key issue discussed in the same paper is which part of the reality must be simulated. According to the authors, only the important features have to be included in the model. These features are chosen intuitively and they vary according to the kind of simulation or controller we want to build. The validation of the model will then assure the correctness of the choice.

A simulator that includes only the strictly necessary features and that introduces noise (to emulate environmental disturbs) can be defined as a *minimal* one.

One of the most tough task we had to accomplish during our research work was the design and the implementation of the *MindS-miss* simulator. To validate the results of the virtual environment, we use the ones obtained in real robot experiments (to be more precise, we control that the differences among the three strategies detected in simulation are the same that we find in real robot experiments).

The *MindS-miss* is not a fully minimal simulator. It introduces noise to robot readings, but we decided to use an accurate physical engine, because we estimated that physical interactions among agents and between agents and prey could be very important in our case study.

The UML schema of the simulator can be found in Appendix A.

4.3.1 Simulated Physics

In order to obtain an accurate reproduction of the reality, we built our simulator employing a physical simulator engine. We began to work with the Vortex^{TM} libraries but then we switched to the Open Dynamics Engine,[3] a

[3] http://ode.org

free, high-quality library for simulating articulated rigid body dynamics.

Another important side tool we used in the development of our simulator is KODEX.[4] This wrapper has been coded by Kovan Laboratory[5] and it is intended to provide model loading facilities similar to Vortex^{TM}. Moreover, it translates the function calls to the Vortex^{TM} libraries into the ODE equivalent, creating, when necessary, additional data structures.

Unfortunately, the KODEK version we used[6] still contained a huge number of bugs. To avoid dangerous problems, the final version of our simulator included a significant number of direct calls to the ODE library and several bug reports were sent to the Kovan Laboratory in order to improve the development of their wrapper.

All the relevant elements of the experiments, as well as the interactions among them, are modeled and reproduced in *MindS-miss*. The circular arena is represented as a solid floor surrounded by a finite number of squared walls. In the center of the experimental field, a sphere indicates the nest of the colony. The prey are reproduced as solid black cylinders that appear and disappear from the environment.

Each robot has been modeled with a limited number of polygons. Main body, front and back bumpers, six wheels (three at each side of the main body, that replace the tracks of the real robots) and an additional led to indicate the gripper activity are introduced in the virtual environment as independent bodies and then linked with special joints.

The movement of the robot is decided by its controller applying a torque force to the six wheels. The simulation of the gripper is somewhat simplified because it is obtained creating a temporary joint between the robot main body and the cylinder representing the grasped prey. Gripping time, needed by the arms of a real robot to open and close, is modeled as a delay introduced when the joint is created and deleted.

The core of *MindS-miss* is represented by the independent controllers used by each robot involved in a specific simulation. They have the same

[4] Kovan ODE eXtensions

[5] http://kovan.ceng.metu.edu.tr

[6] KODEX version 0.5.2

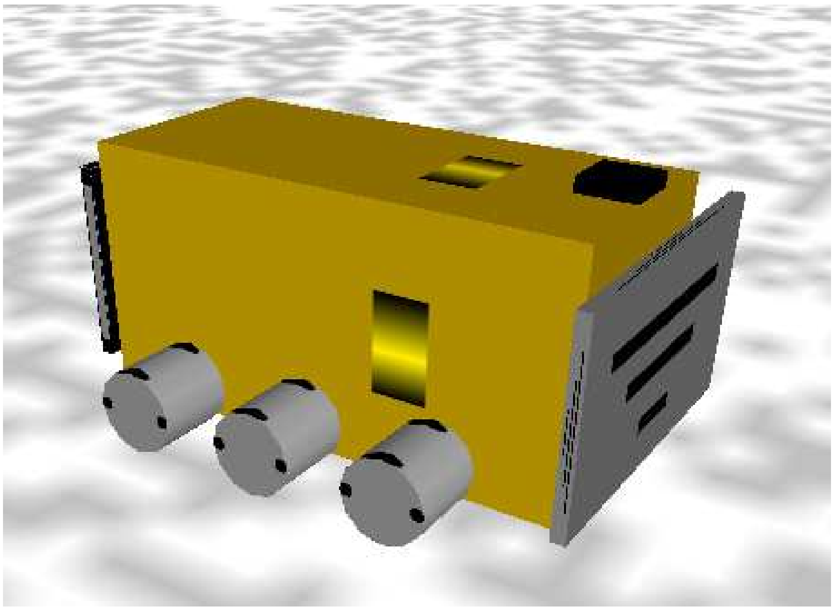


Figure 4.4: Each simulated MindS-bot is a simplified but physically accurate version of the original ones. In fact, the behaviour of the virtual robots is the same that we can observe on the hardware-implemented agents.

structure of the ones we coded for the real robots and they are invoked every 100 ms.

At each simulation step,[7] the Open Dynamics Engine checks the contacts among the objects present in the simulation and applies the forces that are necessary to resolve physical contacts and constraints. When a bumper touches another body, a special callback function we wrote in *MindS-miss* passes the information to the control system of the concerned robot, simulating a touch sensor.

In order to speed up the simulation, the central sphere and the walls are declared as *frozen*. This way, the physics engine does not need to calculate the forces applied to those objects. Another trick we use to improve the performance is to disable every possible check over the collisions among objects that will never touch each other (i.e., the wheels on the right side of a robot will never collide with the ones placed on the other side).

[7] The simulation step is invoked with a sensibly higher frequency than the control step because it has to ensure the coherence of the physics in the simulated world.

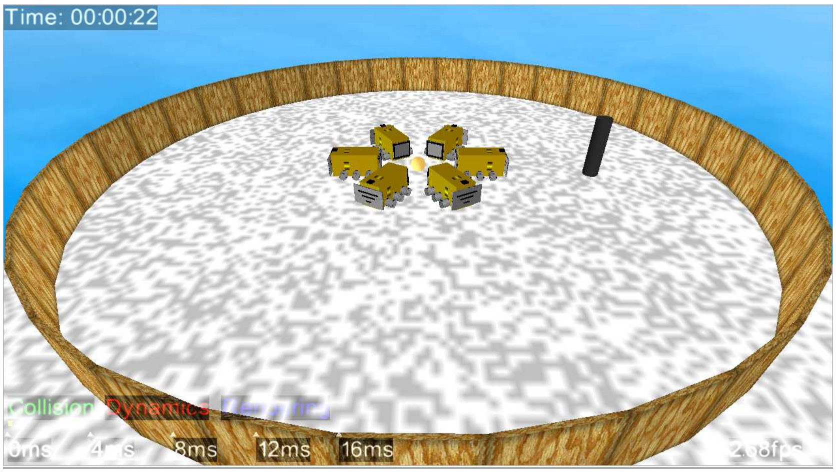


Figure 4.5: Colony composed by six simulated MindS-bots in their starting positions, at the beginning of an experiment. The black cylinder is the representation of a prey.

All the objects in the simulation and their properties are described in XML files parsed recursively. The main program loads the root file (`World.me`) and then all the other ones referenced by it.

All the bodies are created at start-up but the prey, that need to be created and destroyed according to a special timing file (`prey_timing.sam`), which is also loaded at the beginning of the simulation.

Even if usually no rendering is used during the simulated experiments, a renderer is available and it was used to make a first visual control over the behaviour of the colony in the virtual environment. The renderer is based on OpenGL libraries and a picture of the simulated world where our robots act is shown in Figure 4.5.

4.3.2 Tuning the Parameters

As we said in section 4.3, one of the main critiques appointed to the simulators was the lack of coherence with the real world. ODE has proved to be a robust and reliable library, but in order to obtain a valid simulated environment we had to tune all the parameters that describe each object represented in the virtual world.

As suggested by Jakobi et al. (1995), we based our work on empirical observations. Weight and physical dimensions of every real object were carefully measured, scaled and included in the XML description of the correspondent simulated body.

Parameters that were not easily measurable, such as the coefficient of dynamic friction between two objects or the exact speed of the tracks of the MindS-bots, were reasonably guessed and then hand tuned, comparing the real behaviour and the simulated one.

To obtain a reliable model of the two light sensors placed on top of each robot, we used the sampling technique. We sampled sensor readings in the real environment (according to the distance and the angle of visibility among the objects). With the obtained values, we compiled a table (look-up table) for each sensor. At each control step, the main simulation engine computes the position of the objects in the arena and, for the sensors of each robot, finds in the respective look-up tables the readings to pass to the specific controller. A probabilistic error that simulates the environmental noise is then added to the values found and the final results are what the controller really reads from the simulated sensors.

4.3.3 The Simulated Experiments

The simulated experiments reflect the real ones in every possible detail. As we said in the previous sections, the controller of the virtual MindS-bots has been ported with no major modifications from the one running on the real robots. The use of a virtual class `Controller` and of three other derived classes (one for each learning algorithm we want to compare) allows a fast and easy change in the learning strategy of the robots.

The length of each experiment is set to 3600 seconds and no visual interface is employed to speed up the simulation. Accurate log files are saved during each run. The logs collect the same data recorded by real robots and some additional information.

Since we have at our disposal a high performing, multi processor cluster, we decide to run more than five experiments for each setup and we set the

		prey probability		
		0.005	0.01	0.02
size	2	40	40	40
	4	40	40	40
	6	40	40	40

Table 4.1: This Table shows the different setups we use in the simulated environment. They are classified by the values of two independent variables. *Size* of the colony indicates the number of robots employed in the experiment, while *prey probability* is the probability at each second for a prey to appear in the arena. Forty runs are performed for each experimental setup.

number of runs to forty.

As for the real experiments, we want a comparison among the learning algorithms as fair as possible. We use the same instances for prey timing already employed in experiments with real robots. Additional ones are also generated to fulfill our needs in simulation.

The random number generator of the simulator, used by robot controllers, accepts an initialization value, called seed. Passing twice the same seed to the simulator at the beginning of a run ensures that the two sequences of generated random numbers during the experiments are exactly the same. Since we intend to simulate forty runs for each experimental setup, we prepare also a list of forty seeds that are paired with the instances of prey appearance already mentioned.

4.3.4 Better than Reality

The possibilities of the simulated environment are far beyond the ones we have with the real robots. We can change more freely the two variables that define the experimental setup, the *size* of the colony and the *prey probability.* The use of a higher number of robots does not increase the probability of a major hardware failure that may invalidate the experiment. Moreover, the simulator can introduce in the arena an arbitrary number of prey. An example of setups run only in simulation can be observed in Figure 4.6, where a colony of six MindS-bots is searching for prey in the virtual arena.

We intend to test the learning strategies under as many different condi-

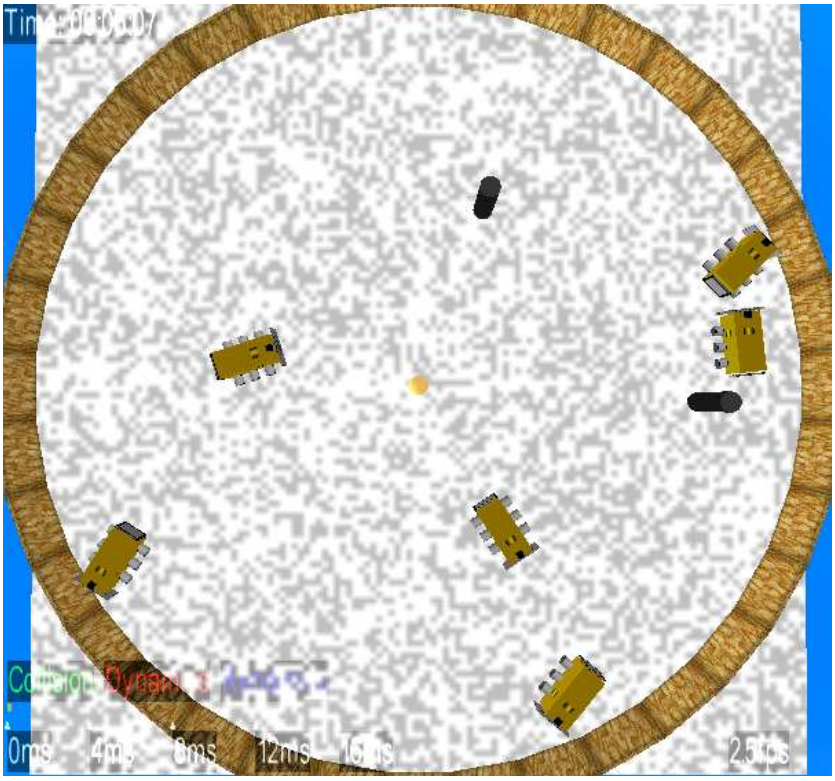


Figure 4.6: The simulator allows us to perform experiments with a higher number of robots. In this example, a colony of six MindS-bots is searching for prey in the arena.

tions as possible. Consequently, we decide to try nine setups. The setups are characterized by three different values of both variables, *size* and *prey probability*, as illustrated in Table 4.1. The setups corresponding to {*size*=4, *prey probability*=0.005} and {*size*=4, *prey probability*=0.01} are the ones we test also with real experiments.

The total number of runs we perform for each learning strategy is then equal to 360. Among these simulations, we discard the ones that present run time problems. ODE and KODEX are still under development and, testing our simulator, we noticed that a low percentage of trials terminated their execution prematurely.

As we verified, the problems do not arise from our code, but they are linked to infrequent run-time errors that emerge in the mentioned libraries and that simply cause the immediate end of the program. The detection of these occurrences is very easy and the correspondent log files can be then excluded from the final data analysis.

Chapter 5

Three Different Strategies

In this chapter we present the original version of the three algorithms we chose for our research, as well as the reasons why we chose them and the modifications we introduced to let them fit our experimental setup.

We also show how the learning strategy of the algorithms is applied to the parameter we need to adapt, the *Time in Nest*.

5.1 A Specific Prey Retrieval Algorithm

The first algorithm on which we focus was designed by Labella (2003). The aim of the original work was to prove that swarm intelligence techniques can be used in a prey retrieval task to improve the performance of the system.

Communication among agents is not allowed and the dynamic adaptation of the agents in the environment is obtained by exploiting only local information. A learning strategy, based on probabilistic parameters, is the core of the adaptation process.

The work was later extended by the author, that focused on the importance of task allocation in prey retrieval (Labella et al., 2004a) and on the efficiency improvements obtained through collaboration among agents (Labella, 2004).

Since our work originated from the research made by Labella, this algorithm was the natural starting point for our comparison.

In order to extend the original work described in the mentioned papers, we decide also to use as test field for our real robots the experimental setup realized for Labella's tests, as already seen in previous chapters.

This thesis focus on the comparison among different learning strategies applied to a specific parameter that controls the time agents spend in the nest. Therefore, we want to keep as constant as possible the control architecture of the robots, with the only exception of the learning algorithm used to update the parameter of interest.

We made this choice in order to directly link the performance of the swarm to the efficiency of each learning strategy. It is up to them to create the right level of specialization among the agents. If we do not vary the main structure of the robot architecture, we eliminate the interference of any other variable, not related to the core of our research, that could influence the experimental results.

In the next sections, we analyze in detail the phases in which the prey retrieval task has been decomposed to implement the main control system.

5.1.1 States and Behaviours

The control program of the agents can be summarized with a finite state machine, as represented in Figure 5.1.

The main states described represent the different phases that compose the prey retrieval task. The transitions are executed under some specific self-explicative conditions highlighted in the schema.

Each state can be considered as an independent sub-task:

Rest The MindS-bot stays in the nest, waiting until the predicate *can_go* changes its value to *true*. When it happens, the agent becomes active and starts searching. The time it takes to *can_go* to become true is controlled by the learning algorithms.

Search The robot explores the environment, searching for prey. The moving direction is chosen randomly and obstacles are avoided. When a prey is found and grasped (that is, *have_prey* is *true*), the agent switches its

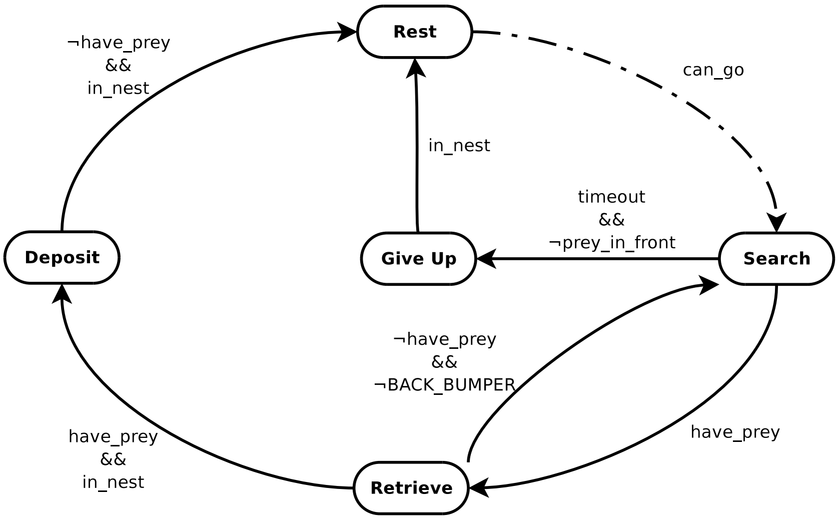


Figure 5.1: This diagram is a high level representation of a slightly modified version of the control program realized for the MindS-bots by Labella (2003).

state to *Retrieve*. If the searching time expired and no prey is spotted in front of the robot, then it has to give up.

Retrieve The MindS-bot carries the prey to the nest, avoiding obstacles if needed.

Deposit When the robot arrives in the nest, it drops the prey. As soon as the operation is completed and the retrieving has been successfully completed, it starts resting and updates the parameter that determines how long it will stay in the nest (this parameter expresses a probability value in Labella's algorithm).

Give up The robot gives up. It comes back to the nest after a failure and updates its parameter for nest resting.

The reaction to external stimuli is achieved with a number of behaviours. These behaviours allow the execution of the sub-tasks and are triggered by some *activation conditions* directly related to sensor readings.

Each state uses only a restricted number of behaviours, as it can be seen in Table 5.1.

State	**Behaviour**	**Conditions**
Search	gripping	hit_prey ∧ gripper_open
	avoiding	(front_bumper ∧ gripper_open ∧ ¬prey_in_gripper) ∨ back_bumper
	releasing	gripper_closed ∧ ¬prey_in_gripper
	exploring	gripper_open ∧ ¬front_bumper ∧ ¬back_bumper
Retrieve	avoiding	back_bumper
	back_light_following	¬back_bumper
Give Up	avoiding	back_bumper
	back_light_following	¬back_bumper
Deposit	depositing	have_prey
	back_light_following	¬have_prey ∧ ¬in_nest
Rest	nest_exiting	

Table 5.1: List of behaviours used by every possible state in the control system of a *MindS-bot*. Each behaviour is triggered by the *activation conditions* listed in the last column of the table. The execution of a behaviour inhibits the activation of all others listed below.

5.1.2 How Robots Learn

The aim of this algorithm is easily explained: given a colony (or *swarm*) of homogeneous robots employed in a foraging task, they should self-organize to improve the overall efficiency with a dynamic task allocation. The goal, fully achieved in the original research, was to insure the creation of two classes of agents, *foragers* (agents with a low latency time in the nest) and *loafers* (robots that rest for longer periods), according to the prey availability in the environment.

The difference among the robots lies in the amount of time they spend in the nest before leaving to search for prey. In this algorithm, this is not a fixed time, but it is determined by a probabilistic value, P_l.

A specific algorithm, called *Variable Delta Rule*, was created by the author to adjust the probability to leave the nest of each agent. This system, specified in Algorithm 1, rewards the robot when it successfully retrieves a prey, raising its P_l value. An agent reporting a failure, on the other hand,

Algorithm 1 Variable Delta Rule.
P_l is the probability for the agent to leave the nest

```
initialization:
successes ← 0
failures ← 0
$P_l$ ← INITIAL VALUE

if prey retrieved then
    successes ← successes + 1
    failures ← 0
    $P_l$ ← $P_l$ + success * $\Delta$
    if $P_l > P_{max}$ then
        $P_l \leftarrow P_{max}$
    end if
else if timeout then
    failures ← failures + 1
    successes ← 0
    $P_l$ ← $P_l$ - failures * $\Delta$
    if $P_l < P_{min}$ then
        $P_l \leftarrow P_{min}$
    end if
end if
```

will see its probability reduced.

The amount to be added or subtracted to P_l is not constant, but it varies according to the consecutive failures or successes reported by the agent. P_l is also bounded. If it were not bounded, an agent could reach a P_l value too high or, on the opposite side, equal to 0. In the second case, the agent would be never again allowed (during the current experiment) to leave the nest, stopping the process of dynamic adaptation chased by the author.

In the original version of the algorithm, as well as in ours, P_l=[0.0015, 0.05] and the initialization value of P_l is fixed to 0.033.

When an agent rests in the nest, its controller casts a random generated number between 0 and 1 at each second. If the value is less or equal to the value of P_l, then the robot is allowed to leave the nest and the *can_go* flag displayed in Figure 5.1 is set to *true*.

The search time for each agent is fixed to 228 seconds. After that period, if no prey is in sight, the agent comes back to the nest, reporting a failure. The timeout value was decided by Labella after having analysed the mean time a MindS-bot need to find a prey.

5.2 ALLIANCE

The second control architecture we chose for our research is ALLIANCE.

Designed by Parker (1998), it is a fully distributed, behaviour-based software architecture, developed to obtain fault tolerant cooperative control of teams of mobile robots. We chose this algorithm because of the similarities it shows with Labella's one (robots learn and adapt their behaviour in an automatic way, even when a centralized knowledge is absent) and it is a well known and widely used learning strategy in mainstream robotics.

It is interesting that ALLIANCE was not specifically designed for swarm robotics. We want to investigate if the modifications introduced to the algorithm to let it fit our experimental setup do not lead to a loss of efficiency. We can see if an algorithm designed for multi-robot cooperation can be employed using the more restrictive constraints of swarm robotics. It is also worth to note that the missions for which Parker's algorithm has been designed are much more complex than a simple prey retrieval task.

ALLIANCE allows reliable cooperation among small or medium-sized teams of heterogeneous mobile robots. The teams are employed in missions composed by different tasks that can have ordering dependencies.

An important assumption in ALLIANCE is that not all tasks can be performed by all team members. Moreover, even if more than one robot can accomplish a specific task, they can perform it with different efficiency. This is not the case in our experimental setup, because in prey retrieval all robots can perform the single task required.

The environment where the agents act is considered dynamic and it is assumed that the robots of the team have a probability greater than 0 to detect the effect of their own actions through their sensors.

The control architecture has been designed for intentional cooperation. It means that there is communication among the agents and that each agent does not lie to other team mates. Nevertheless, the availability of the communication medium is not guaranteed, as well as the reliability of each agent. The failure of an agent could be not communicated to team mates.

The goal of ALLIANCE is to allow each robot of the team to select appropriate actions to perform during a mission. The choice is made considering the requirements of the mission, the activities of other robots (this information is available through a broadcasted communication), the current environmental conditions and the robot's own internal status.

The whole mechanism is implemented using *impatience* and *acquiescence*, two mathematically-modeled motivations that trigger the execution of high-level behaviour sets. While impatience incentives a robot to perform tasks in which other robots fail, acquiescence enables a robot to handle situations when the robot itself fails to accomplish its task.

Each behaviour set has a parameter, the threshold of activation (θ), that determines the level of motivation beyond which the behaviour set will become active.

Parker (1997) extended the original architecture in L-ALLIANCE. It implements an adaptation of the parameters controlling motivational rates based on learning. This extension is not considered in our work, since it is not useful when agents cannot communicate among them.

5.2.1 ALLIANCE on Prey Retrieval

ALLIANCE has not been designed to be implemented on swarms of robots and the author supposed that a communication channel is available to the agents. Moreover, in the original version of the architecture, the robots are not homogeneous and the missions they have to accomplish are more complex than prey retrieval, where there is only one high level behaviour set.

In order to adapt ALLIANCE to our purposes, we had to introduce some changes in its original design.[1]

[1] We tried to stick as much as possible to the original schema, but we hat to modify it to have sound results. We contacted the author asking for some comments, but we received no answer. When we refer to ALLIANCE in the following chapters, the reader should keep in mind that we refer to our version of the algorithm.

We start from the assumptions described by Parker (1998):

1. The robots of the team can detect the effect of their own actions, with some probability greater than 0.
2. Robot r_i can detect the actions of other team members for which r_i has redundant capabilities, with some probability greater than 0; these actions may be detected through any available means, including explicit broadcast communication.
3. Robots of the team do not lie and are not intentionally adversarial.
4. The communication medium is not guaranteed to be available.
5. The robots do not possess perfect sensors and effectors.
6. Any of the robot subsystems can fail, with some probability greater than 0.
7. If a robot fails, it cannot necessarily communicate its failure to its team mates.
8. A centralized store of complete world knowledge is not available.

As we said, in our work the robots are homogeneous, even if run time problems or not intentional physical diversities could lead to different levels of efficiency, and they cannot directly communicate.

In our experiments, there is only one high level behaviour set. It leads the robots to search for prey in the environment and to retrieve what it has been found to the nest or to come back after the expiration of a searching time limit. We decide to set the time limit value to 228 seconds, as it is in Labella's algorithm.

To adapt the control architecture to our needs, we modify some of the original assumptions in the following way:

1. The robots of the team can detect the effect of their own actions, with some probability greater than 0.

2. Robot r_i **cannot** directly detect the actions of other team members (MindS-bots do not have sensor capability to distinguish team mates and their actions).

3. Robots of the team are not intentionally adversarial.

4. The direct communications medium is **never available** (in swarm robotics direct communication among agents is not allowed. The only kind of communication the robots may employ is the indirect one, as insects use *stigmergy* to organize their collective behaviour).

5. The robots do not possess perfect sensors and effectors.

6. Any of the robot subsystems can fail, with some probability greater than 0.

7. If a robot fails, it **cannot directly communicate** its failure to its teammates.

8. A centralized store of complete world knowledge is not available.

Another important aspect that needs to be modified is the way with which impatience and acquiescence are handled.

In the original algorithm a motivational behaviour works as follows. A robot's motivation to activate any behaviour set is initialized to 0. Then, the robot's motivation to perform a specific behaviour set is incremented at a fast rate of impatience, as long as the task corresponding to that behaviour set is not being performed by any other robot, or at a slow rate of impatience, if some other robot is performing it. Impatience is modeled as a specific mathematical function.

If the motivation level of a behaviour set exceeds the value of its threshold of activation (θ), the behaviour set becomes active.

If a robot is performing a specific task, it may still give up if it perceives that the task is not being accomplished in an acceptable period of time. The giving up mechanism is handled with the acquiescence characteristic.

We do not present here the full formal model described by Parker (1998). In the following, we detail the modified version of ALLIANCE we used in our work, its main parameters and mathematical formulas.

Motivation Motivation value is modeled as a mathematical function:

$$m(t) = [m(t-1) + impatience(t)] * acquiescence(t) \tag{5.1}$$

When the value of the function is greater than the threshold of activation[2] θ, the robot is forced to exit from the nest and to look for prey.

This function is calculated only when the robot is resting in the nest and its increment is time dependent. The value of motivation can be set to 0 only when $acquiescence(t) = 0$, that means the robot is giving up and coming back to the nest to start resting.

We eliminated three other terms present in the original version of this function. They modified the final value of motivation under circumstances that are not valid in our experimental setup, such as the communication among team mates and the presence of more than one single behaviour set.

Acquiescence This mathematically-modeled motivation is originally used by the robot controller to evaluate when it is time to give up and to come back to the nest.

The value of this function becomes really important when the behaviour set is active (that is, when the robot is searching for prey). When the time spent by the robot searching exceeds the timeout limit of 228 seconds, the function value is set to 0.

$$acquiescence(t) = \begin{cases} 0 & if \ \textbf{timeout} \ expired \\ 1 & else \end{cases} \tag{5.2}$$

[2] In our prey retrieval experiments, we have only one behaviour set that can be executed by the robots. Therefore, when we generally refer to θ, we intend the threshold associated to that behaviour set.

After giving up, the robot returns to the nest and it starts resting. Motivation value is first reset and then it begins to increase again. Once the resting period starts, acquiescence remains constantly equal to 1.

As we did for motivation, we had to simplify the original acquiescence function, to eliminate all the parameters dependent to direct communication among robots and to the presence of more than one high level behaviour set.

Impatience Considering that in our case robots are not aware of the state of the other agents, their impatience value grows independently, as it is specified by the following functions:

$$impatience(t) = \delta\_fast(t) \tag{5.3}$$

$$\delta\_fast(t) = \frac{\theta}{(min\text{-}delay\ +\ (task\text{-}time(t)\ -\ low)\ *\ scale\ factor)} \tag{5.4}$$

Where:

task-time(t) = (average time over robot's trials on prey retrieval task) + (one standard deviation of the times of these attempts)

We set the value of some parameters needed to calculate the impatience function. The constant *max-delay* is set equal to 228 seconds (the same value of *timeout*) and we experimentally estimate *min-delay* = 45 seconds. The latter represents the minimum possible delay employed by a robot to exit the nest, to grab a prey and to retrieve it (to measure the minimum value, we placed a prey just outside the nest's perimeter and in front of the robot).

We also decide the values for two additional parameters, *high* and *low*, used in the original version of the algorithm:

low = *min-delay*

$high = max\text{-}delay$

Our choice is motivated by the fact that in the modified version of the algorithm it is not possible for an agent to record the performance of its team mates. Then, we assign to the two parameters the minimum allowed value. Finally, we calculate the *scale factor*

$$scale\ factor = \frac{max\text{-}delay\ -\ min\text{-}delay}{high\ -\ low} = 1 \qquad (5.5)$$

At the beginning of each experiment (both simulated and real), to avoid that the robot starves in the nest for a long period before starting to look for prey, we initialize its motivation value to $0.87 * \theta$. With this correction, the first exit from the nest is after about 30 seconds from the beginning of the experiment, as it happens in average in Labella's algorithm and at each start up in the one described in the next section.

The low level behaviours, that compose the main behaviour set, are the same we use in the first algorithm. What changes among the two strategies is the time the robots spend in the nest and the way this time varies over time. In the case of the modified version of ALLIANCE, it is the *task-time* variable to lead the adaptation. It modifies the growing rate of the *motivation* considering the performance of the robot over the past trials made.

5.3 Distributed Learning in Swarm Systems

The third candidate for comparison is an adaptive line-search algorithm presented by Li (2002) and Li et al. (2004). The goal of the algorithm is to offer a method to improve the performance of an artificial swarm on a given task. Learning has been chosen as an automatic way to adjust the control parameters without a priori assuming the degree of heterogeneity needed by the swarm.

The main idea behind the work of the authors is that, starting from an homogeneous swarm of agents, the adaptation obtained with the learning algorithm can lead to a diversification among the robots. This diversity can bring significant advantages to the swarm overall performance, since the system becomes specialized.

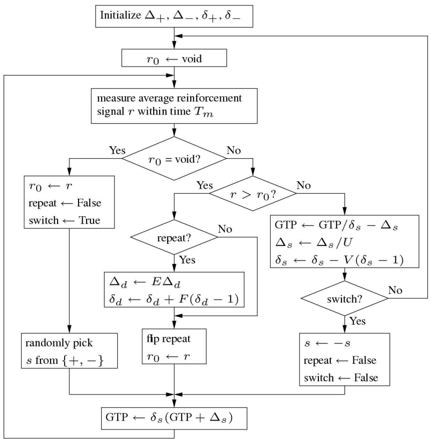
Li et al. (2004) identify also a method to measure specialization in a system, as a function of both diversity and overall performance of the swarm. Specialization can be defined as "the part of diversity that is demanded for better performance".

This learning strategy has been tested in simulation over two generalized versions of the stick-pulling experiment and over the original one as well.

The original version of that experiment was used by Martinoli and Mondada (1995) and by Ijspeert et al. (2001) to investigate collaboration in non-communicating robot swarms. In the first set-up, robots equipped with a gripper and proximity sensors search for sticks in a circular arena to pull them out of the ground. The length of each stick has been chosen in a way that a single robot cannot succeed on its own in the task, but only a collaboration between two agents with two successive grips can lead to the accomplishment of the task. In the original version of the experiment, the control system of the agents was quite simple. Each robot randomly moves in the arena. When it identifies a stick, it tries to pull it out of the ground. The agent can recognize by the speed of the elevation arm whether another team mate is already holding the same stick or not. In the first case, the task is completed; the robots leave the stick out of the ground and start searching for new ones. In the second case, the robot holds the stick for a given time, waiting for another agent to come and to complete the task.

The maximum delay that a robot waits holding the stick is called the *gripping time parameter* (GTP). The experiments explained in the already mentioned papers show the variation of the performance of the swarm when both the GTP of each robot and the size of the swarm change.

The two generalized versions of the stick pulling experiment have been introduced by Li et al. (2004) and tested only in simulation. They have the same structure of the original experiment, but they extend its purposes focusing on different aspects of the collaboration. Issues related to sequential collaboration and to parallel collaboration are explored. The goal is achieved using longer or heavier sticks. The former require the successful sequential


Initialize $\Delta_+, \Delta_-, \delta_+, \delta_-$
$r_0 \leftarrow$ void
measure average reinforcement signal r within time T_m
r_0 = void?
Yes
No
$r_0 \leftarrow r$
repeat ← False
switch ← True
$r > r_0$?
Yes
No
repeat?
No
Yes
$\Delta_d \leftarrow E\Delta_d$
$\delta_d \leftarrow \delta_d + F(\delta_d - 1)$
GTP ← GTP/$\delta_s - \Delta_s$
$\Delta_s \leftarrow \Delta_s/U$
$\delta_s \leftarrow \delta_s - V(\delta_s - 1)$
switch?
No
Yes
randomly pick s from $\{+, -\}$
flip repeat
$r_0 \leftarrow r$
$s \leftarrow -s$
repeat ← False
switch ← False
GTP ← δ_s(GTP + Δ_s)

Algorithmic variables:

	Range	Description
s	$\{+, -\}$	search direction
Δ_+	$[2, 60]$	GTP offset (sec)
Δ_-	$[-60, -2]$	GTP offset (sec)
δ_+	$[1.1, 5]$	GTP factor
δ_-	$[0.2, 0.9]$	GTP factor
r_0	$\{\text{void}\} \cup \mathbb{R}$	previous performance
r	$\mathbb{R}$	current performance
repeat	Boolean	reinforcement flag
switch	Boolean	direction flag

Algorithmic parameters:

	Value	Description
T_m	2400	averaging period for reinforcement signal (sec)
E	1.9	GTP offset enlarge factor
F	0.3	GTP factor enlarge ratio
U	2	GTP offset shrink divider
V	0.5	GTP factor shrink ratio

Figure 5.2: The original learning algorithm designed to improve the swarm's performance by means of an adaptive change of the GTP parameter, as presented by Li et al. (2004).

collaboration of a number k of robots, while the latter needs a kind of parallel collaboration because k agents must act at the same moment to pull them out of the ground.

The learning algorithm described by Li (2002) and Li et al. (2004) focuses on the automatic and dynamic adaptation of the GTP in a swarm of independent robots. The agents adapt their parameter in an independent way to maximize the overall performance of the colony.

What makes this learning strategy particularly suitable as our third choice is that, as Labella's one, it was specifically designed for swarm robotics. Moreover, it presents only one parameter that is adapted with learning.

In the stick pulling experiment, as well as in prey retrieval, every agent has the same capabilities of the others. In the first case, they have to grip sticks out of the floor, while in the second one the task is accomplished when a robot retrieves a prey found in the environment. Finally, also the shape of the arena used in the original stick pulling experiments is circular, as the one we prepared for our experimental setup.

With the original learning algorithm, two different types of reinforcement signal are used. A local one rewards the agent when it achieves a successful collaboration, pulling a stick out of the ground. The second one, a global reinforcement signal, is broadcasted to all the agents and it specifies the performance of the swarm. The global reinforcement signal is particularly important in the generalized version of the stick pulling experiment that requires sequential collaboration, because in this case only the the robot that makes the final grip knows if the collaboration has been successful.

The schema of the original version of the learning algorithm is shown in Figure 5.2. After the initialization of the variables that hold the offset and the multiplication factor of the GTP (δ_-, δ_+, Δ_- and Δ_+), the main cycle starts. Each agent first updates its GTP in a random chosen direction s. Then, after a period T_m, the robot checks its performance using the two reinforcement signals. If the performance is better than before, then the agent continues in the same direction. Otherwise, it changes the direction and it also undoes the last update made to the GTP. Offset and multiplication factor of the GTP are also changed during learning in order to speed up the convergence of the parameter to the optimal value. They are increased when the same direction is chosen twice, while they are decreased when the performance oscillates.

5.3.1 Learning on Prey Retrieval

As we did for ALLIANCE, we describe now how the original algorithm designed for the stick pulling experiment has been modified to fit our experimental setup.

As already said, we do not use any communication media among the robots or between the robots and a possible central unit. In the original version of the algorithm both a local and a global reinforcement signal were used. The second one allowed the robots to estimate the performance of their behaviour even in situations where an immediate feedback was not available, i.e., when sequential collaboration was required. In that case, only the last robot, the one who pulled the stick out of the ground, knew that the collaboration was successful. We limit our reinforcement signals to the local one. This should not lead to information loss, because in prey retrieval experiments each agent knows when its task has been successful (a prey retrieved to the nest is obviously a success, while the expiration of the searching time without any finding must be considered as a failure). The main loop of the algorithm is executed each time the robot comes back to the nest.

In the original algorithm, the search direction s is chosen in a random way, because it is not obvious if the GTP must be increased or decreased to improve the performance of the system. In our case, we need to adapt the *Time in Nest* parameter, that should be changed according to the amount of prey found by the agents in the environment. Analyzing the task our robot must achieve, we know that in case of success a robot will most likely find another prey in the environment, while if it comes back to the nest after a failure, the execution of another search task should be delayed.

We deduce that it would be pointless to change our parameter in a random way. The reinforcement signal we use can be only positive or negative (prey found or failure) and we associate the positive one to a reduction of the *Time in Nest* parameter, while a failure would lead to an increase of the value associated to the variable.

In order to make the three algorithms more homogeneous, we bound the variable that holds the *Time in Nest* parameter:

$$WINT \text{ (Waiting In Nest Time)} = \{1, 667\} \tag{5.6}$$

The values of the variable are expressed in seconds. We chose 1 as lower bound because in the first algorithm we presented each robot could leave the nest in every moment since it starts its sleeping time. The upper bound is equal to the average value of the time spent in the nest by a robot using the first algorithm, when the value associated to its probability to leave the nest is equal to 0.0015, that is, the minimum allowed (notice that in Labella's probabilistic algorithm the robots could theoretically wait an infinite amount of time).

The initialization value for the WINT variable is 30 seconds The robots leave the nest for the first time after a period equal to the one we employed in the modified version of ALLIANCE. In Labella's algorithm, the average

Algorithmic variables:

	Range	Description
Δ_+	[2, 30]	
Δ_-	[-30,-2]	
δ_+	[1.1, 5]	
δ_-	[0.2,0.9]	
r_0	$\{$success, failure$\} \cup \{$void$\}$	previous performance
r	$\{$success, failure$\}$	current performance
repeat	boolean	reinforcement flag
switch	boolean	direction flag

Table 5.2: Values and bounds of the variables in the modified version of the algorithm designed by Li et al. (2004).

Variable	Init value
Δ_+	5
Δ_-	-5
δ_+	2.4
δ_-	0.5

Table 5.3: Initialization values of the WINT enlarge and shrink factors and offsets.

delay the agents spend in time at the beginning of the experiment is 30 seconds as well.

The other variables used by Li et al. (2004) are also employed in our modified version. Nevertheless, we change some of the bounds of the original GTP offsets and factors, as shown in Table 5.2.

We do these modifications in order to reduce the dependency of the results on other factors than the learning algorithm. As in the original prey retrieval algorithm and in the adapted version of ALLIANCE, four consecutive successful retrieval bring the WINT variable from its lower value to the higher allowed value and vice versa.

Algorithm 2 details the structure of our modified version of the original Li's learning strategy.

In Tables 5.3 and 5.4 we present the initialization values of the variables and also the constants we used in our adaptation of the algorithm.

The T_m parameter (average period for reinforcement signal) is useless for

Algorithmic parameters:

	Value	Description
E	1.9	WINT offset enlarge factor
F	0.3	WINT factor enlarge ratio
U	2	WINT offset shrink divider
V	0.5	WINT factor shrink ratio

Table 5.4: In our modified version of the algorithm, we do not change the values associated to the parameters. Therefore, these values are the same employed by Li et al. (2004).

our purposes, because the reinforcement is assigned to each robot when it comes back to the nest after a trial (successful or unsuccessful).

Algorithm 2 Li's Algorithm on Prey Retrieval.
WINT is the time an agent has to stay in the nest, expressed in seconds.

```
initialization:
Initialize $\Delta_+$, $\Delta_-$, $\delta_+$, $\delta_-$
$r$ ← no_value    ;    repeat ← 0    ;    WINT ← 30

if prey retrieved then
    if r == no_value then
        repeat ← 0
    else if r == success then
        if repeat == 1 then
            $\Delta_-$ ← E * $\Delta_-$
            $\delta_-$ ← $\delta_-$ + F * ( $\delta_-$ - 1 )
            check if $\Delta_-$ and $\delta_-$ are out of bounds
        end if
        flip repeat
    else if r == failure then
        WINT ← ( WINT * $\frac{1}{\delta_+}$ ) - $\Delta_+$
        check if WINT is out of bounds
        $\Delta_+$ ← $\frac{\Delta_+}{U}$
        $\delta_+$ ← $\delta_+$ - V * ( $\delta_+$ - 1 )
        check if $\Delta_+$ and $\delta_+$ are out of bounds
        repeat ← 0
    end if
    WINT ← $\delta_-$ * ( WINT + $\Delta_-$ )
    check if WINT is out of bounds
    r ← success
else if timeout then
    if r == no_value then
        repeat ← 0
    else if r == failure then
        if repeat == 1 then
            $\Delta_+$ ← E * $\Delta_+$
            $\delta_+$ ← $\delta_+$ + F * ( $\delta_+$ - 1 )
            check if $\Delta_+$ and $\delta_+$ are out of bounds
        end if
        flip repeat
    else if r == success then
        WINT ← ( WINT * $\frac{1}{\delta_-}$ ) - $\Delta_-$
        check if WINT is out of bounds
        $\Delta_-$ ← $\frac{\Delta_-}{U}$
        $\delta_-$ ← $\delta_-$ - V * ( $\delta_-$ - 1 )
        check if $\Delta_-$ and $\delta_-$ are out of bounds
        repeat ← 0
    end if
    WINT ← $\delta_+$ * ( WINT + $\Delta_+$ )
    check if WINT is out of bounds
    r ← failure
end if
```

Chapter 6

Results

In the following, we present the results we obtained applying our methodology. We detail the procedure we employed to prove the soundness of our simulator and its efficiency as an instrument in the research.

We explain which are the parameters we are interested in comparing and what results we obtained for each algorithm.[1]

Also, we show how our procedure allows us to draw final conclusions on the performance of the algorithms we considered.

6.1 Parameters of Interest

Our research focuses on the individual, autonomous learning on one important variable, that is, the time each robot spends in the nest before starting a new prey retrieval task.

ALLIANCE and the learning algorithm designed by Li directly fix the time the agent will wait after each successful or unsuccessful trial, while in Labella's *Variable Delta Rule* the adaptation is made indirectly by means of the changes of a probabilistic value.

The main parameter we use to compare results is the already defined efficiency (see Equation 4.1):

[1] The data analysis has been performed using GNU R, a language and environment for statistical computing and graphics. It provides a wide variety of statistical and graphical techniques.

$$\eta = \frac{\text{number of retrieved prey}}{\text{colony duty time}}$$

It allows to evaluate the income-cost ratio of the captured prey for the whole colony, in terms of time spent searching outside the nest.

Another aspect we are interested in rating is the percentage of prey captured by the robots, for each setup and for each learning algorithm. It is defined as:

$$\frac{\sum_{i=1}^{N} \text{prey captured in experiment } i}{\sum_{i=1}^{N} \text{prey appeared in experiment } i} \tag{6.1}$$

Finally, the last parameter we intend to compare is the degree of specialization we reach with each strategy. We want to evaluate if, at the end of the experiments, our colony has been splitted in two different main categories, *foragers* (robots that frequently leave the nest to retrieve prey) and *loafers* (robots that stay in the nest for longer periods), and if the size of those groups changes according to the setup we are evaluating. We expect that a good adaptation leads to an enlargement of the group of the *foragers* when the *prey probability* increases. On the contrary, if we increment the *size* of the colony (holding the *prey probability*) we expect a different kind of distribution among the agents.

6.2 Simulation Results

We start our analysis explaining the results obtained in simulation for each learning strategy. Later, in Section 6.2.4, we group the results together and analyze the differences among algorithms.

6.2.1 Labella's Algorithm

In this section we elaborate the data we obtained using the learning algorithm designed by Labella (2003), the *Variable Delta Rule.*

Figure 6.1 summarizes the effects on efficiency coming from changes of the two variables which define each experimental setup, *prey probability* and

		prey probability		
		0.005	0.01	0.02
size	2	79.20	70.66	41.92
	4	81.46	86.49	70.83
	6	86.98	86.92	77.07

Table 6.1: Percentage of captured prey over the total number of prey appeared for each experimental setup, when we use Labella's *Variable Delta Rule* (see Equation 6.1).

size of the colony.

As we could expect, efficiency increases when *prey probability* is increased as well. A higher amount of prey in the environment causes fewer failed trials and, in average, an inferior amount of time spent searching before coming back to the nest when the retrieval is successful. We can also notice that the smallest colony (two robots) is the most efficient one in every condition and it reaches its top efficiency in the richest environment.

The reduction in efficiency that we remark when the *size* of the swarm is increased can be motivated by two different reasons. The *Variable Delta Rule* could be not able to eliminate from the group of the *foragers* the robots that are not strictly necessary. Another explanation could come from the way we measure efficiency. Since the P_l value is always greater than 0, even *loafers* leave the nest in average once every eleven minutes. Each trial they do, when unsuccessful, increases the colony duty time.

The percentage of captured prey in each experimental setup is shown in Table 6.1. These percentages are calculated using Equation 6.1 over all the runs done in simulation for each setup.

Evidently, the percentage increases when we consider bigger colonies. We can also see that, holding the *size* of the swarm, the lowest percentage always corresponds to the higher value of *prey probability*. The explanation of this result is straightforward: the colony is never big enough to totally exploit the richest environment. In particular, we notice that the lowest value in the whole table is obtained with the setup that displayed the best average efficiency (*size* $= 2$ and *prey probability* $= 0.02$).

In a very rich environment, having a small colony, we would expect the

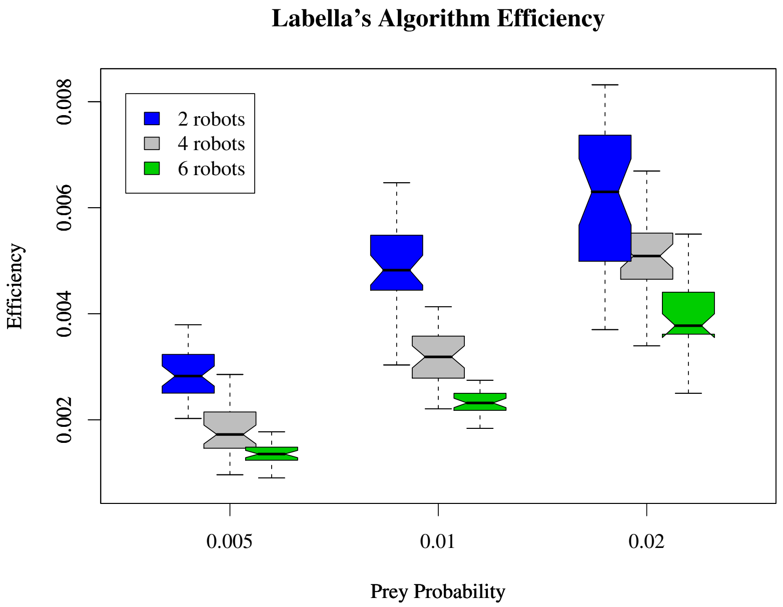


Figure 6.1: Effects of *prey probability* and *size* of the colony on the efficiency of prey retrieval in the simulation, using the algorithm designed by Labella (2003). The bars report the results of forty experiments for each value of *prey probability* (on the x axis) and for each *size* of the colony (different colours of the boxes). The top and bottom limits of the boxes extend from the first to the third inter-quartile of the distribution of the results. The line drawn inside the boxes represents the median value of the distribution. The whiskers extend to the most extreme data point, which is no more than 1.5 times the inter-quartile range from the box.

adaptation process to lead to a situation where all the agents involved in the retrieval task are *foragers*. The learning strategy designed by Labella works properly in this case, as we can see in Figure 6.2, on the right hand side of the first row. This specific histogram holds the final probability values P_l of all the simulated robots employed for the experiments of the setup where $size$ = 2 and *prey probability* = 0.02. As it can be seen, with only a few exceptions, all the agents show a very high P_l value. It means that the colony changed itself into a group of *foragers*.

On the last row of Figure 6.2, in the middle position, we can observe another example of the behaviour of the learning process with this algorithm.

Here we find the final P_l values of the robots involved in the setup where the *size* of the swarm is set to six elements and the *prey probability* is equal to 0.01 (that is, a prey appears in average every 100 seconds). In this case the final situation changes. Not all the robots need to become *foragers* and their final P_l value varies in all the allowed interval. Nevertheless, we can distinguish two main groups in the diagram, near to the highest allowed values of P_l and to the lowest ones. These groups identify the two classes of *foragers* and *loafers*.

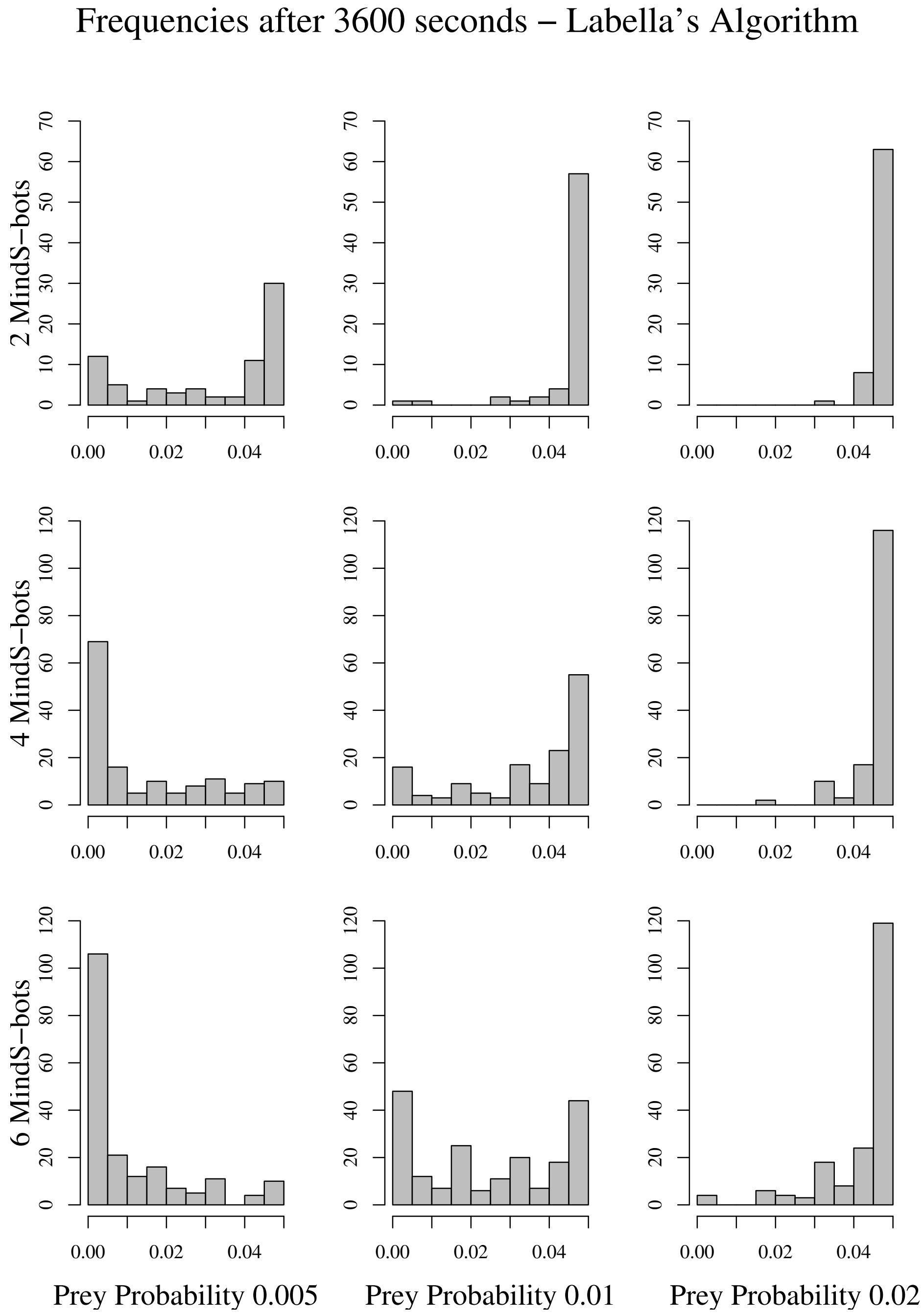


Figure 6.2: Results of the specialization process in the nine different setups we evaluated in our research. On the same row, we find the data concerning a specific colony *size*, while each column corresponds to a different value of *prey probability*. In the diagrams, we represent the P_l values of the robots at the end of the experiments.

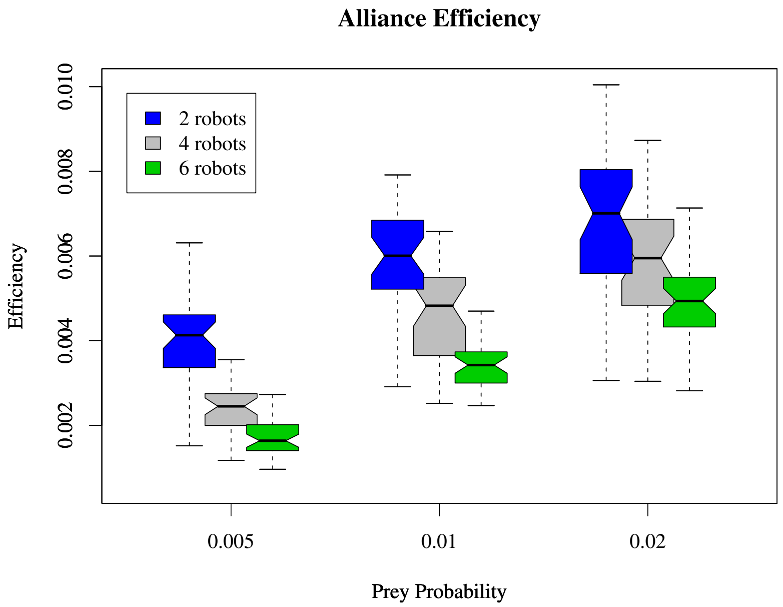


Figure 6.3: Variation of efficiency with different values of *prey probability* and *size* of the colony, using the ALLIANCE learning algorithm. For a detailed explanation of the meaning of the symbols plotted, see Figure 6.1

6.2.2 ALLIANCE

We performed the same set of experiments we prepared for Labella's learning strategy using the ALLIANCE algorithm.

The first parameter of interest, efficiency, is analysed in detail in Figure 6.3. As we can see in the plot, increasing the *prey probability* value and the *size* of the colony, we notice an increase also in the average value of efficiency; but it is not the only change we observe. We can also see an enlargement of the spread between the highest and the lowest value of efficiency for each specific setup, as it can be deduced by the progressive elongation of the whiskers in the diagram.

The percentages of captured prey are presented in Table 6.2. As it was with Labella's strategy, the lowest value in the table corresponds to the experimental setup where we have the richest environment and the smallest colony.

		prey probability		
		0.005	0.01	0.02
size	2	63.52	42.66	23.50
	4	78.97	64.48	39.16
	6	84.47	77.42	51.90

Table 6.2: Percentages of captured prey over the total number of prey appeared in the environment in every one of the nine setups using ALLIANCE (see Equation 6.1).

In this situation, we would expect that also the ALLIANCE adaptation process would lead the agents of the colony to specialize as *foragers*, that is, their final *Task Time* value should be low. This parameter is expressed in seconds and represents the time the robots spend in the nest before starting to search for prey, as it can be inferred by Equations 5.1, 5.3 and 5.4. But if we look at the histogram on the right hand side of the first row, in Figure 6.4, we notice that the bars are grouped in a central position, rather than on the left hand side of the diagram.

We find a similar situation when we consider the experiments done using this learning strategy, a colony composed of six elements and a *prey probability* value of 0.01 (Figure 6.4, last row, central position). Also in this case, we see that the specialization process has led to a situation where most of the robots are gathered in a small portion of the diagram, but there are no classes that emerge from this distribution.

We discuss more about this in Section 6.2.4, where we compare the results.

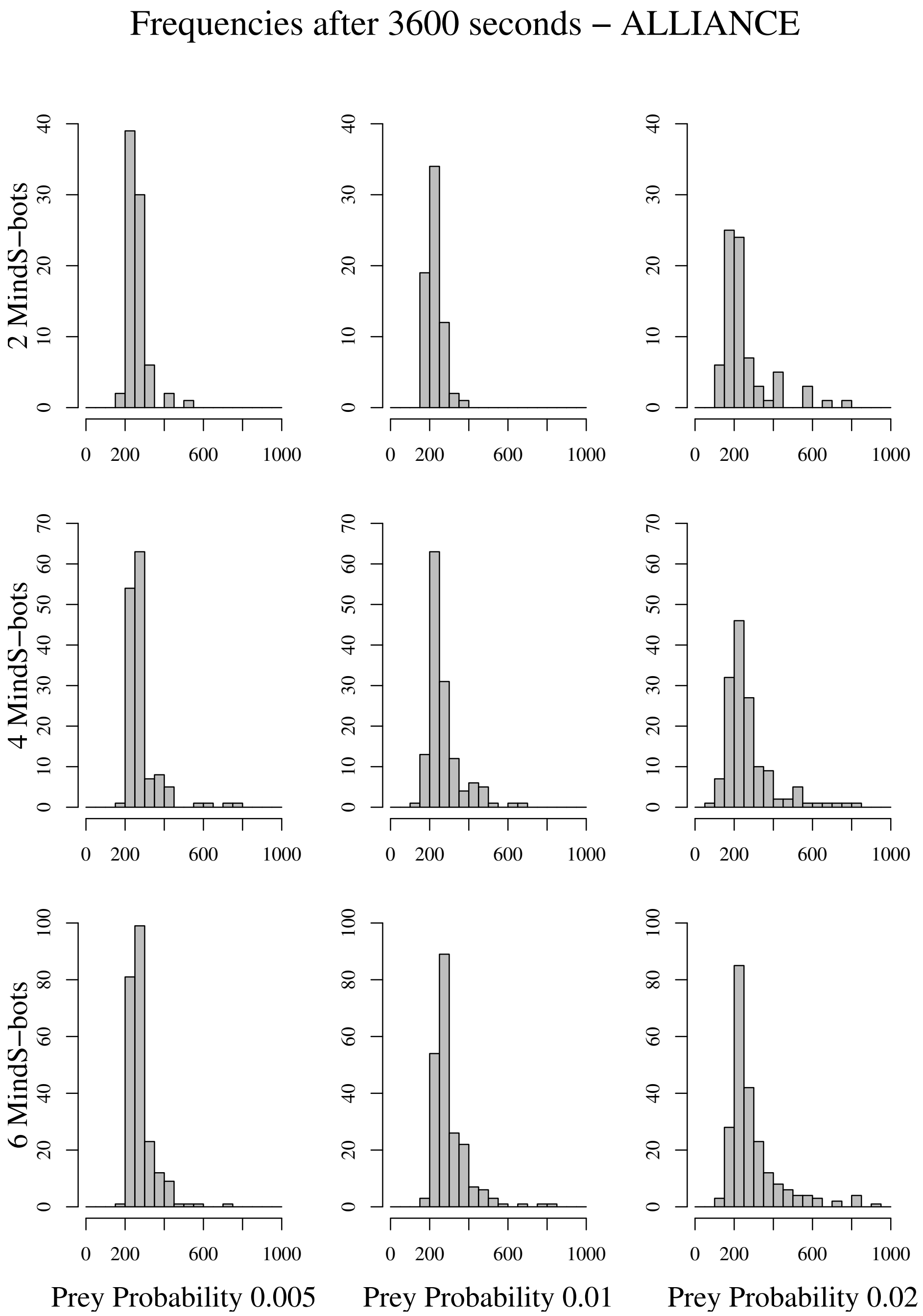


Figure 6.4: Distribution of the *Task Time* parameter final values (expressed in seconds). These values have been obtained at the end of the learning process with the ALLIANCE algorithm in each experimental setup. The bars in the left hand side of each histogram represent the robots that rest less time in the nest (*foragers*), while on the right hand side we find the agents with a higher final value of the *Task Time* parameter (*loafers*).

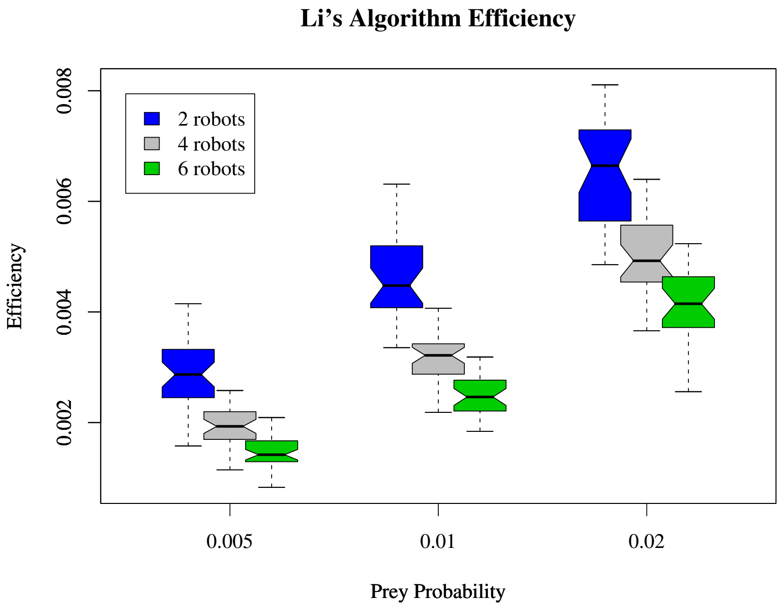


Figure 6.5: Statistical analysis of the efficiency values measured in the simulated environment, using Li's learning algorithm. Each box and its whiskers represent about forty experiments performed on a specific setup. For the explanation of the meaning of each symbol plotted in the diagram, see Figure 6.1

6.2.3 Li's Algorithm

The overall situation of efficiency can be evaluated looking at Figure 6.5. Also in this case, efficiency values increase when the colony reduces its size and the environment becomes richer.

We can already notice that the spread between the highest and the lowest value of efficiency for each experimental setup is less wide than the correspondent we had for ALLIANCE. This means that the performance of this learning algorithm is more constant among different runs and it is less influenced by statistical fluctuations.

Table 6.3 summarises the percentage of captured prey in each setup. A first, qualitative analysis of the data indicates that the distribution of the values is the same we found with the other two learning algorithms. As usual, they increase when we hold the *prey probability* and we expand the

		prey probability		
		0.005	0.01	0.02
size	2	77.42	68.56	47.88
	4	85.22	85.71	73.53
	6	85.63	85.57	83.16

Table 6.3: Percentages of captured prey obtained with the learning strategy originally designed by Li (2002). Each cell holds the percentage of prey retrieved to the nest by the robots over the total amount of prey appeared in the environment during the runs of a specific setup (see Equation 6.1).

colony, with the only exception of the setup where we use six robots in an environment with *prey probability* = 0.01. We can remark that this value is very close to the one just above, on the same column.

Another common point among the three algorithms is that the lowest value in the table is always associated with the setup where we observe the highest efficiency, that is, when the colony is composed of two agents that are exploiting the richest environment.

It is not surprising that in this case almost all the robots are in the group of the *foragers* at the end of the experiments, as we show in Figure 6.6 (first row, on the right). There are only very few exceptions, represented by the small bar in the middle of the histogram, that, in this situation, can be considered as *loafers*.

As we did for the other two algorithms, we analyze also the same kind of plot for a different setup, where we have six robots in the colony and an environment with a value of *prey probability* equal to 0.01 (Figure 6.6, last row, middle position). While most of the agents have a final WINT value that classifies them as *foragers*, we can notice also a small, but significant group of *loafers*. The wide spread between the WINT values of the robots that are likely to retrieve prey and the ones that are supposed to stay in the nest indicates some level of specialization among agents that were homogeneous at the beginning of the experiments.

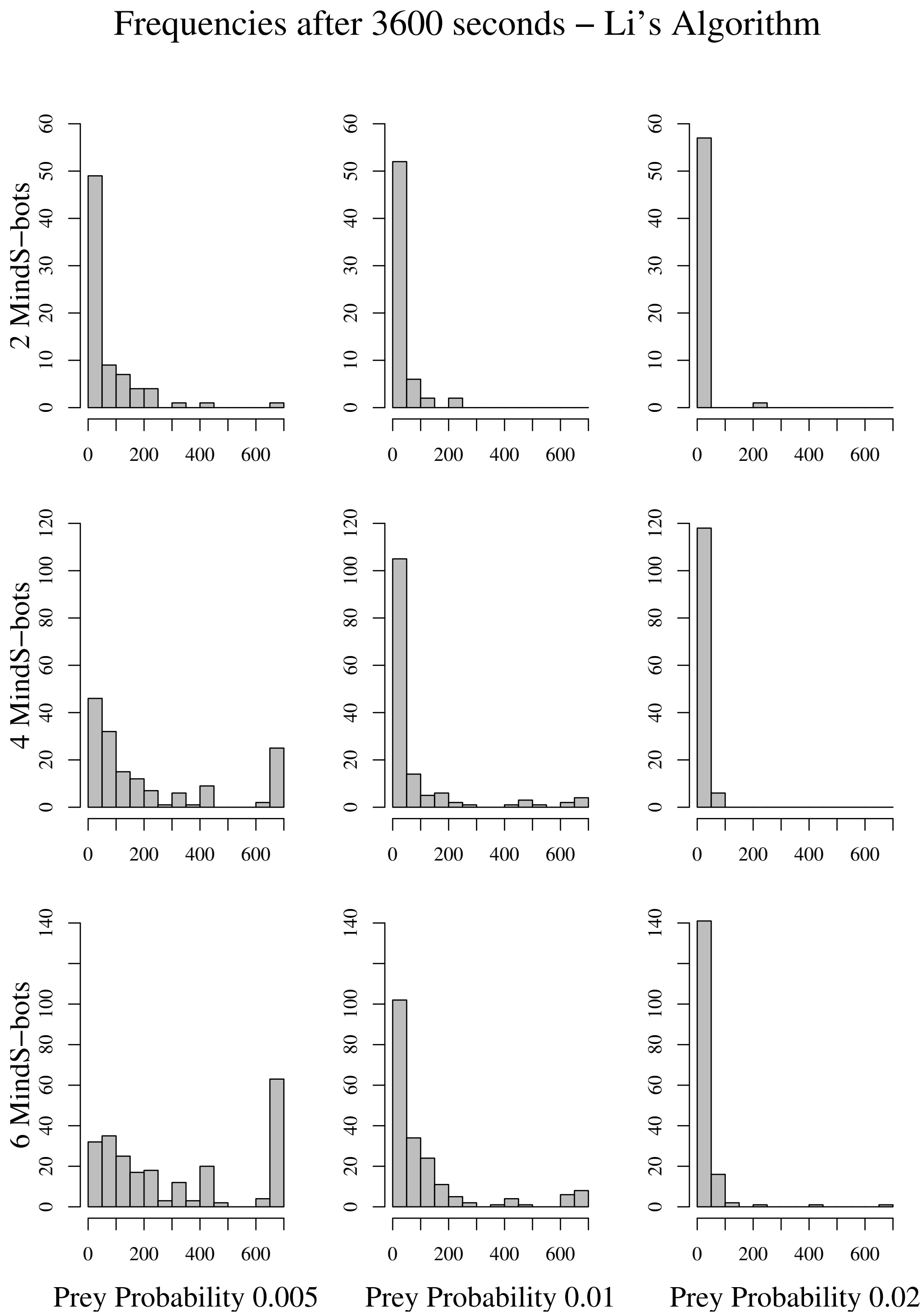


Figure 6.6: Distribution of the *Waiting In Nest Time* (WINT) final parameter values in each experimental setup. These values express the time the agents spend in the nest at the end of the learning process and are recorded when the experiments terminate. A low WINT value means that the agent has become a *forager*, while the bars on the right hand side of each histogram represent the robots that spend more time in the nest (*loafers*).

6.2.4 The Algorithms Compared

While in the previous sections we analyzed in detail each single algorithm, now we want to draw our conclusions about their performances.

Efficiency

We start our analysis from the results obtained in terms of efficiency, the first parameter of interest defined in the previous sections.

Figure 6.7 shows the plots of the effects on efficiency of different learning strategies, in environments where *prey probability* changes.

We see that, in average, ALLIANCE always performs better than the algorithms designed by Labella and Li. Nevertheless, it shows the biggest variance. It is evident, indeed, that the performance of this algorithm is less constant over different runs of experiments prepared for the same setup.

In order to prove that there is a difference among the algorithms, we decide to perform two nonparametric statistical tests (Siegel and Castellan Jr., 1988). We must state a null hypothesis H_0 and its alternative H_1. The former is formulated with the express purpose of being rejected. If it is the case, then the alternative hypothesis is supported. In our case, obviously, H_0 is that the three samples have been drawn from the same population or identical populations.

We take as sample size the whole set of experiments we performed, under all the explored experimental setups, and we set our significance level α equal to 0.05. If the value that comes from the the test statistic falls into the region of rejection,[2] then our decision is to reject H_0 and to support H_1 (that is, there is a difference among the samples coming from the different algorithms).

We start our statistical tests with the Friedman two-way analysis of variance by ranks. This method is intended for testing the null hypothesis that three or more samples have been drawn from identical populations and it

[2] The sampling distribution includes all possible values that a test statistic can take on. The region of rejection consists of a subset of these possible values, and it is chosen so that the probability under H_0 of the occurrence of a test statistic having a value which is in that subset is α (Siegel and Castellan Jr., 1988).

indicates if there is an overall difference among the samples. This is necessary before picking up any pair of samples to test the significance of the difference between them. If we began our analysis by comparing each of the three samples with the other ones, using a two sample test, we would have three chances, rather than only one, to reject the null hypothesis when it is true (this is called type I error). We would take the risk to make a type I error three times, instead of only one. Only when an overall test allows us to reject the null hypothesis, it is justified to employ a procedure for testing for differences between any two of the three samples.

More details about the Friedman test can be found in Appendix B.

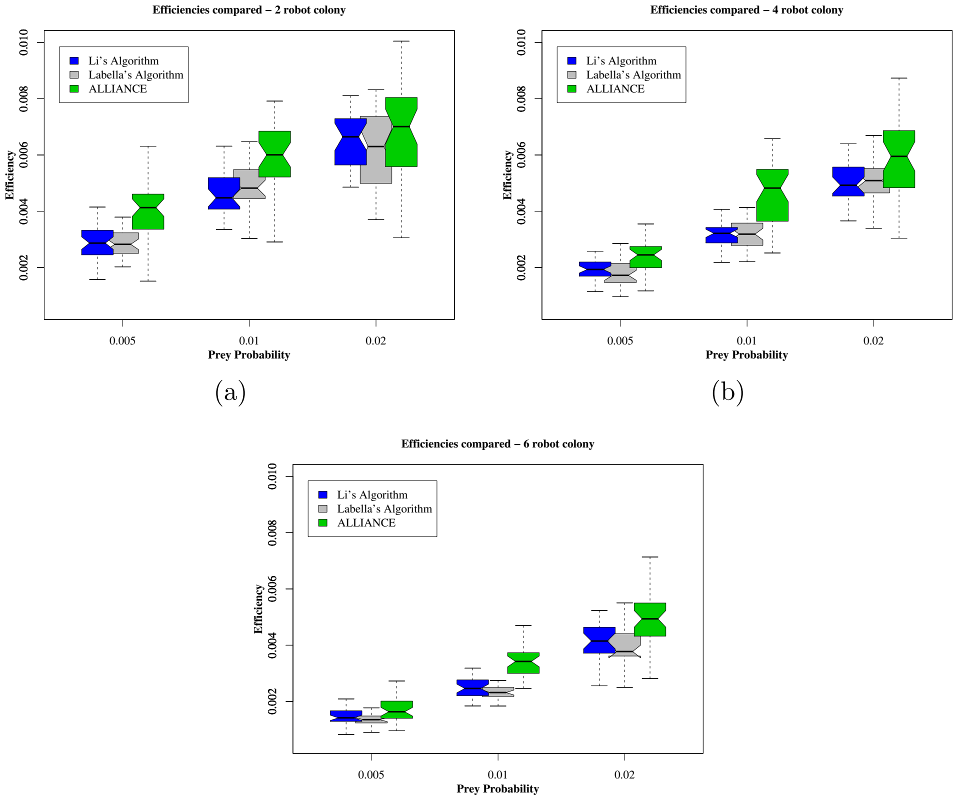
As we said, we set the significance level α for our test equal to 0.05. The Friedman test on the efficiencies returns $F = 183.5161$ and a p-value $\ll 0.001$, that is, we can *reject* the null hypothesis that the three samples have been drawn from identical populations and then there is a significant difference among the efficiency shown by the three algorithms.

We are now justified in employing a procedure for testing for differences between any two of the three samples.

For this purpose, we choose to perform the random permutation test for paired replicates. This test involves paired replicates and is is intended to establish whether two algorithms are different or if one is better that the other. A description of rationale and method of this test can be found in Appendix B.

As we expected, ALLIANCE is the best performing algorithm when we consider their efficiency. When compared both to Li's learning strategy and to Labella's *Variable Delta Rule*, the p-values we obtain are largely inferior to 0.001. It means that we can reject the null hypothesis of no difference and we can state that ALLIANCE shows the best efficiency values.

The comparison between the *Variable Delta Rule* and Li's algorithm gives us a p-value of 0.6426. Therefore there is no significant difference.



(a) (b)

(c)

Figure 6.7: Direct comparison among the efficiencies of the learning algorithms. The results are grouped by the *size* of the colony. Figure 6.7(a) refers to a two robot swarm. The boxes are categorized by *prey probability* and the learning strategy to which they refer. The same kind of plot is presented also in Figures 6.7(b) and 6.7(c), that refer, respectively, to a four and to a six robot colony. For a complete explanation of the meaning of each symbol plotted in the diagrams, see Figure 6.1. It is evident that, in average, ALLIANCE performs better than the other two algorithms, even if the results are spread on a wider range of values.

Labella's Algorithm

		prey probability		
		0.005	0.01	0.02
size	2	**79.20**	**70.66**	41.92
	4	81.46	**86.49**	70.83
	6	**86.98**	**86.92**	77.07

(a)

ALLIANCE

		prey probability		
		0.005	0.01	0.02
size	2	63.52	42.66	23.50
	4	78.97	64.48	39.16
	6	84.47	77.42	51.90

(b)

Li's Algorithm

		prey probability		
		0.005	0.01	0.02
size	2	77.42	68.56	**47.88**
	4	**85.22**	85.71	**73.53**
	6	85.63	85.57	**83.16**

(c)

Table 6.4: The three tables summarize the percentages of captured prey in the nine different setups. Each table groups the results of a learning strategy. The best performance in a specific setup is indicated in bold characters. ALLIANCE is always dominated by the algorithms designed by Li (2002) and Labella (2003), that show a better performance, respectively, in four and five setups.

Percentage of Captured Prey

The second parameter of interest is the percentage of captured prey over the total number of prey appeared during experiments, when different learning strategies are employed. The data, presented in the previous sections, are now grouped and compared in Table 6.4.

Considering the methodology we employed in our research[3] (see Chapter 4), we can directly compare the results obtained in each particular setup.

For a two agent colony in an environment where *prey probability* is equal to 0.005, for instance, we see that the best performing algorithm is the one designed by Labella, with a percentage of captured prey of 79.20. The second best score belongs to Li's learning strategy, while the third place is for

[3] For each value of *prey probability* that we examine in our research, we prepared forty instances of prey timing, as well as forty seeds to pass to the random number generator of the simulator. We used those pairs (instance - seed) to run the set of experiments three times, each time with a different learning algorithm.

ALLIANCE.

The progress of the values in each table, when we consider a single learning algorithm, is almost the same. In all the cases we examined, the lowest percentage is placed in the cell corresponding to a two robot colony that exploits a rich environment. The highest values, on the contrary, refer to bigger swarms, when the *prey probability* is equal to 0.005 or 0.01.

We marked in bold characters the best performance for each experimental setup. In most cases, we see that the results of ALLIANCE are strongly inferior to the ones obtained by the two other algorithms. The best scores are quite equally splitted between the two other learning strategies, that show similar values almost in every experimental setup.

Also in this case, we perform both the Friedman test and the random permutation test for paired replicates to have a statistical evidence of the difference among the algorithms (see Section 6.2.4). The sample we consider is composed of all the experiments we performed with each learning strategy, and the values in each triplet are the number of captured prey during each experiment.

The Friedman test returns $F = 164.6674$ and a p-value $\ll 0.001$. It allows us to reject the null hypothesis that the three samples have been drawn from identical populations and then there is a significant difference among the values of captured prey shown by the three algorithms.

We continue our analysis with the random permutation test for paired replicates. As we expected, in this case both Li's algorithm and Labella's *Variable Delta Rule* show a better performance than ALLIANCE. Both the p-values we obtain from the comparisons are largely inferior to 0.001. Then, since we are performing a one-tailed test, we can state that the robots using ALLIANCE as learning strategy captured an inferior number of prey than the two other algorithms.

The p-value we get from the comparison between Labella's algorithm and the one designed by Li is 0.8858. In this case, we cannot reject the null hypothesis that the former has a better performance than the latter when we consider the number of captured prey.

Specialization

Finally, we analyze the degree of specialization that the colony has reached at the end of the experiments.

In order to have a fair comparison, we show the final distribution of $\frac{1}{P_l}$ [4] (Labella), *Waiting In Nest Time* (Li) and *Task Time* (ALLIANCE) values in three different setups, one for each value we considered for *prey probability* and *size* of the swarm.

The first case we examine corresponds to a two robot colony in an environment where *prey probability* is set to 0.005 (Figure 6.8.)

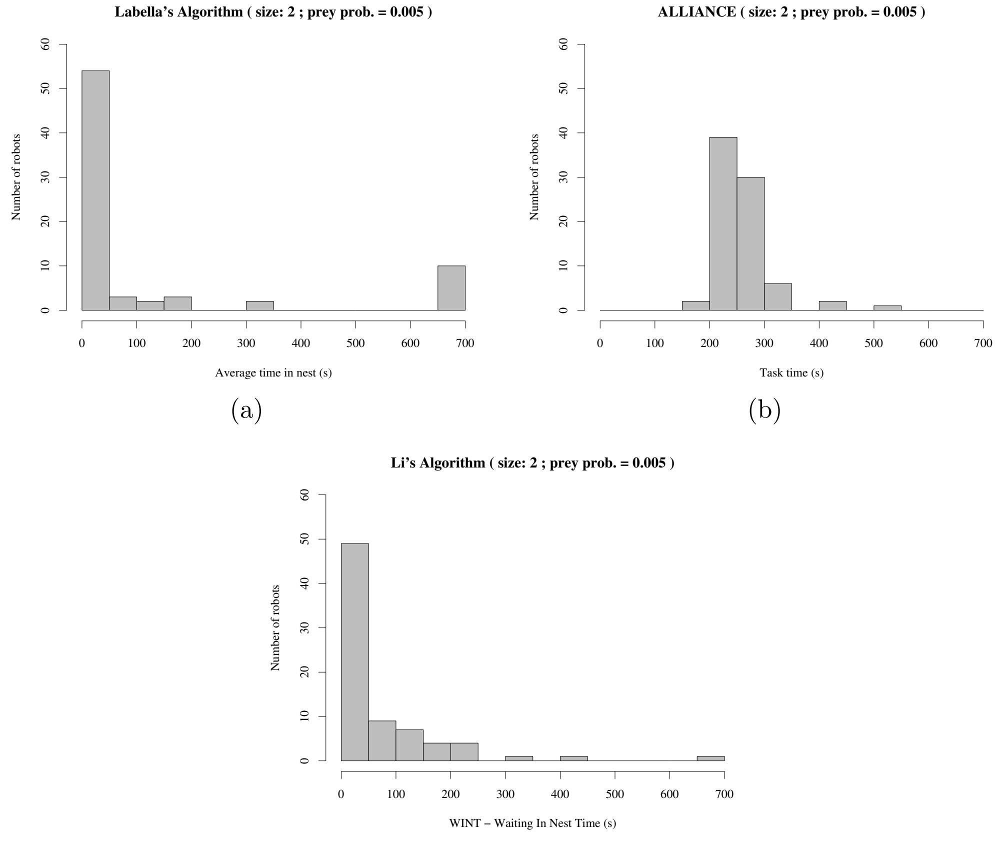
As we can see, Labella's *Variable Delta Rule* leads to a final situation where the agents are splitted in two macro categories, that we already defined as *foragers* (high P_l, then low $\frac{1}{P_l}$) and *loafers* (right hand side of the graph). The final values of the *Waiting In Nest Time* shown by the robots using Li's algorithm are concentrated mostly in the left hand side of the histogram. This means that the agents specialized themselves as *foragers*, even if small groups among them present a significant higher value of WINT. Finally, the situation is rather different when we look at the ALLIANCE final distribution. In this case we notice that the values of the *Task Time* parameter are concentrated in the middle part of the graph, with very small groups both on the left hand side and on the right hand side. We insist that, in the case of Labella's algorithm, we did not plot the P_l values (P_l is the parameter on which the learning is focused), but their inverses. This way, we obtain the average time spent in the nest by each agent at the end of the learning process and we can directly compare the results of Labella's *Variable Delta Rule* with the ones obtained with ALLIANCE and Li's algorithm, because all of them are expressed in seconds.

We find a similar situation analyzing another experimental setup. Figure 6.9 shows the final specialization of the agents using the three different learning strategies when the *size* of the swarm is equal to 4 and the *prey*

[4] P_l is the probability for a robot to leave the nest at each second. We decide to plot the $\frac{1}{P_l}$ value instead of P_l. It means that we plot the average time the agent spend in the nest before leaving, rather than its probability to leave. This way, the measures of all the three algorithm are expressed in seconds and, thus, the comparison is more evident.

probability is set to 0.01. Both Li's algorithm and Labella's *Variable Delta Rule* allow a categorization of their agents in the two mentioned categories, even if, in the first case, we notice that the group of *loafers* is limited to only a few robots. The histogram where we find the results for ALLIANCE (Figure 6.9(b)) presents a totally different shape when compared to the other two. As in the previous experimental setup, there is no real specialization, because most of the *Task Time* values are grouped in the middle of the graph.

Finally, we examine the experimental setup where a colony of six agents has to deal with the richest environment we considered. As we could expect, both Li's algorithm (Figure 6.10(c)) and the *Variable Delta Rule* (Figure 6.10(a)) group their final results in a range where most of the agents can be considered as *foragers*. Despite the rich environment, ALLIANCE does not allow its robots to lower their *Task Time* values as much as the environment could require. Indeed, most of the agents spend from 200 to 400 seconds resting, before starting a new prey retrieval task.

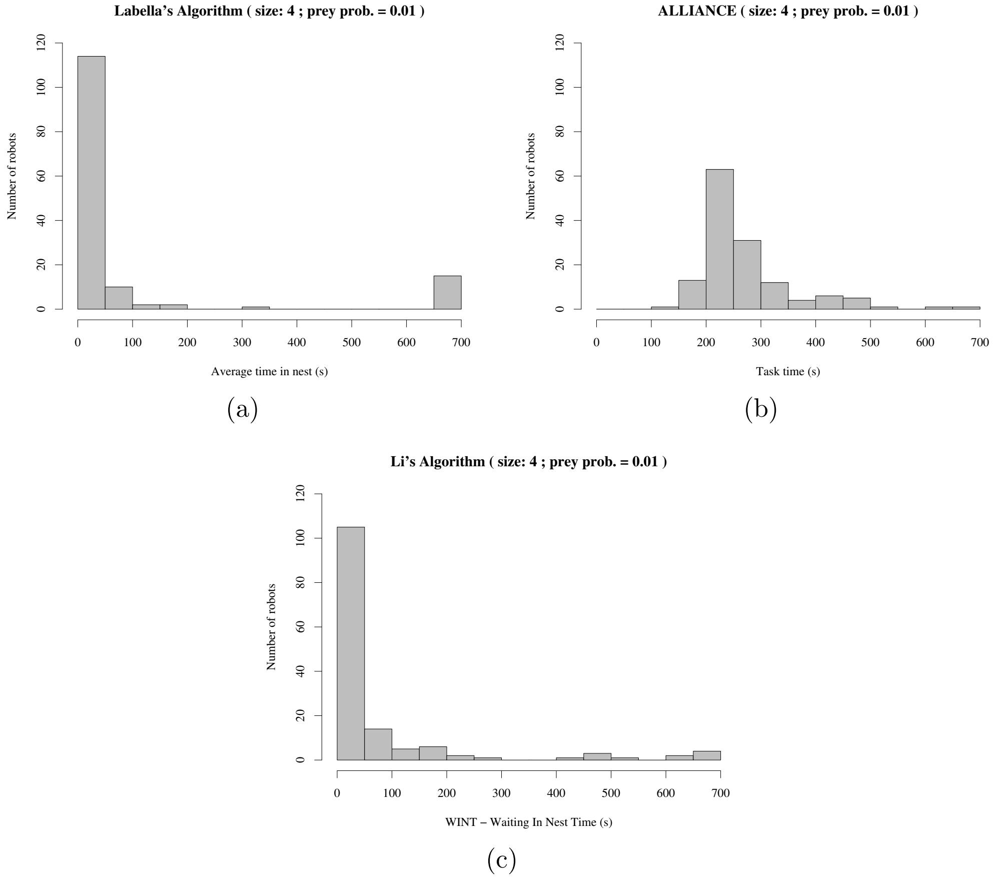


(a)

(b)

(c)

Figure 6.8: Degree of specialization reached at the end of the experiments by a two robot colony in an environment where *prey probability* is equal to 0.005. As it can be seen, Labella's learning strategy (Figure 6.8(a)) splits the robots in two different groups, *foragers* (on the left hand side of the histogram) and *loafers*. Li's algorithm (Figure 6.8(c)) offers a different situation, where almost all the robots have become *foragers*, except for a very small group. In ALLIANCE (Figure 6.8(b)), the final *Task Time* values of the robots are concentrated in the middle of the plot. It is worth to note that, showing $\frac{1}{P_l}$ instead of P_l for the robots that used Labella's *Variable Delta Rule*, the values on the x axis of the three histograms can be directly compared. While *Task Time* and *Waiting In Nest Time* express the exact time the robot spends in the nest before starting the retrieval task again, $\frac{1}{P_l}$ represents the average time it rests before leaving the nest.

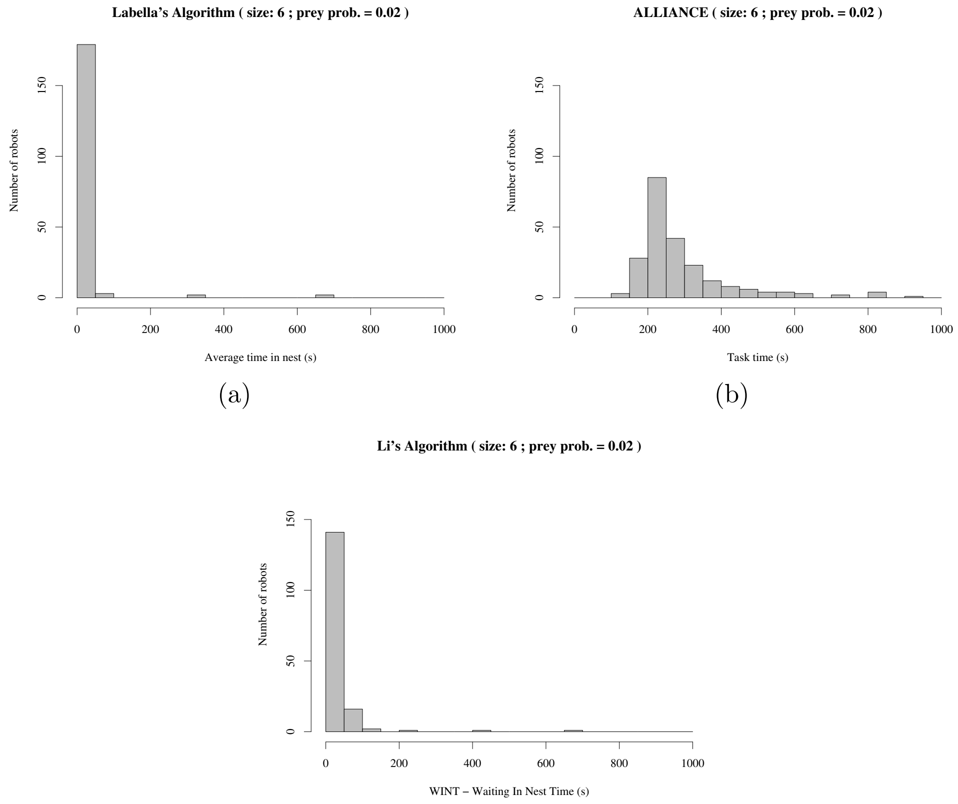


(a)

(b)

(c)

Figure 6.9: Final specialization for the three algorithms at the end of the experiments of the setup where *size* = 4 and *prey probability* = 0.01. The agents that learn by means of the *Variable Delta Rule* can be categorized in two big groups, even if a very small amount of robots have their final value distributed in the middle of the histogram (Figure 6.9(a)). In this situation, most of the agents have become *foragers*. Also Li's learning strategy leads to a situation where most of the robots belong to the group of the *foragers* (Figure 6.9(c)). Only a small group presents their final WINT value in the right hand side of the graph. ALLIANCE shows a peculiar distribution where almost all the final *Task Time* values are grouped in the middle of the histogram (Figure 6.9(b)).



(a)

(b)

(c)

Figure 6.10: The last experimental setup for which we present the specialization histograms involves a six robot colony acting in the richest environment we consider. As we could expect, both Li's algorithm (Figure 6.10(c)) and the *Variable Delta Rule* (Figure6.10(a)) bring most of the agents to be specialized as *foragers*. As in the previous situations, ALLIANCE's *Task Time* values are grouped between 200 and 400 seconds (Figure 6.10(b)).

6.3 Real Robot Results

This section is intended to validate the results we obtained in simulation using the ones we found with real robots. It is not possible to have a full validation of the simulator. That is, it is not possible to have any proof that the results of the simulations will always match those of the real robots for every possible setup. This comes from the fact that a simulator implements a model of the reality. The researcher chooses and develop the model according to her/his understanding of the phenomena and her/his needs.

The process of validation shall not aim at proving the match between simulation and reality. On the contrary, it shall aim at proving that the simulation is not valid. A failure of such attempt increases the confidence in the validity of the simulator and the correctness of its results.

We focused our attention on those values of *prey probability* and group *size* that give in simulation either big or low differences among the algorithms. Figure 6.7 shows that, when we consider efficiency, there are big differences for *prey probability* values of 0.01 or 0.02 and colonies composed by four or six elements.

It is difficult to perform experiments with real robots using colonies of six agents, because of the risk of major hardware failures. Additionally, *prey probability* equal to 0.02 leads to a so high rate of prey appearance that it is often difficult for the experimenter to place them at the right time. Therefore, we used four robots with a value of *prey probability* of 0.01, with the purpose to show that in such conditions the big difference among algorithms does not occur.

We tested five instances and performed the Friedman test to verify whether there is a statistical difference among the three learning strategies in terms of efficiency. The significance level α is set to 0.05, as it was for the simulated experiments. Calculating this statistic, we obtain $F = 7.6$, with p-value $= 0.02237$. It means that, using only five triplets of efficiency values, we can reject the null hypothesis that the three samples come from the same population or from populations with the same median.

This result allows us to perform the pairwise comparison between algorithms using the one-tailed random permutation test. As we did before, we use the Holm correction for multiple testing to adjust the final p-values. The results confirm what we found with the simulator. The direct comparisons between ALLIANCE and Labella's *Variable Delta Rule*, and between ALLIANCE and Li's algorithm lead to the final p-values of 0.0255 and 0.026, respectively. Both of them are smaller than 0.05, our significance level, and, as a result, we can reject H_0. Since we perform one-tailed tests, we can also deduce that, when we consider efficiency, ALLIANCE is the best performing algorithm of the three.

The direct comparison between the learning strategy designed by Li and the *Variable Delta Rule* shows a p-value equal to 0.2201, that is not sufficient to reject the null hypothesis.

The results obtained with the two statistical tests confirm the presence of wide differences among the algorithms in this setup. Five real robot experiments for each learning strategy allowed us to find the same kind of differences we found in simulation. We are therefore not able to invalidate the simulator in this case.

We focus then our attention on those parameters that show in the simulation a small difference. Figure 6.7 suggests three experimental setups: $\{\textit{prey probability} = 0.02, \textit{size} = 2\}$, $\{\textit{prey probability} = 0.005, \textit{size} = 4\}$ and $\{\textit{prey probability} = 0.005, \textit{size} = 6\}$. For the same reasons as above, we chose to use the setup where *prey probability* is equal to 0.005 and the colony is composed by four robots. For this setup, our purpose is to show that there is a relevant difference among the algorithms. As above, such difference should appear with a few instances, if it exists in the real robots.

Performing the Friedman test over five instances, we obtain a p-value greater than 0.05. It means that we cannot reject the null hypothesis that the samples come from populations with the same median. This is a less strong, but important evidence of the soundness of the simulator. Five runs in this experimental setup are not sufficient to emphasize the differences among the efficiency values. In order to find such a difference, we should perform more experiments with real robots. Additional experiments would be however a wast of resources. We are not interested in showing that there is

	Prey Probability	
	0.005	0.01
Labella's Algorithm	81.72	87.13
ALLIANCE	68.89	74.85
Li's Algorithm	84.94	91.81

Table 6.5: Percentages of captured prey in the two different setups we used for real robot experiments. The colony is always composed by four robots. The results are qualitatively close to the ones we showed for simulation. ALLIANCE never scores better than the other two learning strategies, while we get close values from Labella's *Variable Delta Rule* and from Li's algorithm.

a difference, rather in showing that such difference is of a different magnitude than in the simulator, which seems not to be the case with these parameters.

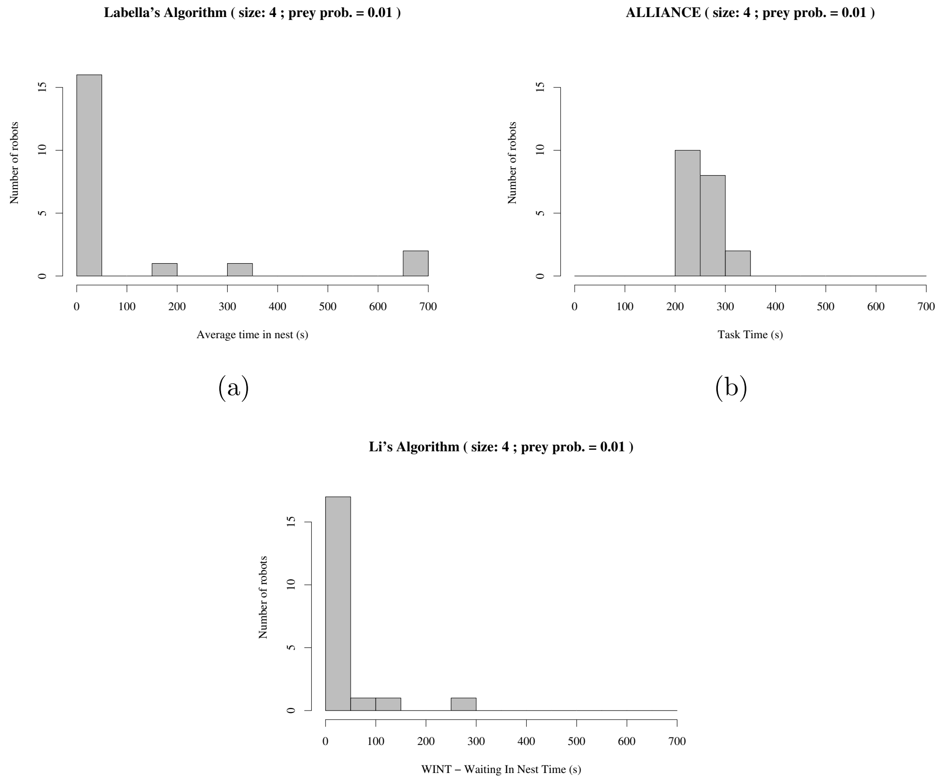
Table 6.5 shows the percentages of captured prey over all the prey appeared in the environment during the experiments with real robots. As we can see, the values are coherent with the ones we found in simulation. ALLIANCE displays scores sensibly lower than the other two algorithms. Moreover, Li's algorithm and Labella's *Variable Delta Rule* show close values in both the experimental setups. As we did for simulated runs, we perform a statistical analysis on the number of captured prey. We take all the experiments we performed as sample size (N = 10). The Friedman test returns $F = 12.4118$ and a p-value equal to 0.002018. This result, totally coherent with the one obtained in simulation, allows us to refuse that the three samples come from populations with the same median. The random permutation test, finally, confirms what found in simulation, that is, both Li's algorithm and Labella's *Variable Delta Rule* have a better performance than ALLIANCE on the number of captured prey (p-values equal to 0.0096 and 0.01, respectively).

Finally, we can see in Figure 6.11 that also a qualitative analysis of the third parameter of interest reflects what we observed in simulation. The histograms can be directly compared with the ones in Figure 6.9. The scales on the x and y axis are the same in the two figures and the overall arrangement of the bars is the same, as we could expect from a reliable simulation.

ALLIANCE evidently does not specialize its agents and their final *Task*

Time values are grouped in the middle of the diagram. On the other hand, we can characterize two different groups in the two histograms corresponding to Li's algorithm and to Labella's *Variable Delta Rule*.

All our attempts to invalidate the simulator failed. On the contrary, they show a good agreement between simulation and real robots. These results support the usefulness of the simulation as an instrument for our research.



(a)

(b)

(c)

Figure 6.11: Final specialization of the real robots used in experiments where the colony size was set to four agents and *prey probability* was equal to 0.01. It is evident the final result of the specialization process with Li's learning strategy (Figure 6.11(c)) and with Labella's *Variable Delta Rule* (Figure 6.11(c)). In both cases, we can notice that most of the robots that can be categorized as *foragers*. On the other hand, ALLIANCE does not specialize its agents and their final *Task Time* values are concentrated in the middle of the histogram. As we did analyzing the results from simulation, also in this case we show $\frac{1}{P_l}$ instead of P_l for the robots that used Labella's *Variable Delta Rule*. This way, the values on the x axis of the three histograms can be directly compared, because they are all expressed in seconds.

6.4 Conclusions

What we can infer from the above analysis is very interesting. It is true that ALLIANCE scores best on the first, and most important, parameter of interest (efficiency), but this result is achieved never truly specializing agents and, consequently, collecting a relatively low number of prey on the total appeared. The *Task Time* final values are grouped in the middle of the possible range in almost every experimental setup. It means that the robots using ALLIANCE as learning strategy do not really change their behaviour according to the environmental situation.

On the other hand, both Li's algorithm and the *Variable Delta Rule* designed by Labella (2003) show a better adaptation to the environmental situation and to the *size* of the colony. Their robots, in average, spend more time outside the nest (thus obtaining lower efficiency results) and collect a remarkably higher number of prey (as it is evident from Tables 6.4(a), 6.4(c) and 6.4(b)).

We present more practical conclusions about the results of the three algorithms in Section 7.2.

Chapter 7

Discussion

In this last chapter, we draw some final conclusions about what has been done in our research in terms of aims and major findings. We summarize here the methodology we developed and applied to the different learning algorithms studied and we evaluate the results that emerged from the data analysis. Finally, we describe briefly what we could expect as further research work.

7.1 An Overview of the Research

It was our intention to detail a methodology to evaluate the performance of different learning strategies applied to the Swarm Robotics field.

The procedure we explained in the previous chapters utilizes methods already accepted in many other scientific fields, that is, to bring all the learning algorithms to a common test field and to evaluate their performances regardless of different hardware implementations or environmental variations.

First, we defined the test field where we intended to perform our analysis. We chose prey retrieval as the task the robots must accomplish. Our choice is motivated by the importance that this simple task has both in robotic literature and in biology, which is the domain that mostly inspired Swarm Robotics. Prey retrieval does not require direct communication among the agents. The robots can exploit local information to evaluate their performance and to adapt their behaviour to the environmental conditions.

The second step we made was to define the experimental setups that describe the conditions under which the learning algorithms were tested. We delineated the setups by means of two variables: *size* of the colony and *prey probability*. The second variable specifies the probabilistic rate at which prey appear on the testing field.

Since we intended to compare learning strategies, regardless of hardware effectiveness, we chose to employ very simple robots. The choice fell on the MindS-bots, robots built with Lego MindStorm^{TM} bricks designed by Labella (2003) that have sufficient sensor capabilities to perform a prey retrieval task.

Considering that real experiments are time consuming and in order to expand our research to a wider number of different experimental setups, we decided to use also a simulator. We coded a specific software, *MindS-miss*, intended to emulate the MindS-bots and their interactions with the environment in a slightly simplified but realistic way.

Our goal, fully achieved at the end of the research, was to test the algorithms in simulation and then with real robots. The real experiments were intended to validate the soundness of the simulator and to prove that the results obtained in the virtual environment could be valid also in the real world.

We prepared nine different experimental setups. For each of them we decided to perform forty experiments in simulation. Each set of experiments had to be performed as many times as the number of algorithms we intended to examine. We wanted the comparison as fair as possible, thus we decided to prepare forty instances of random generated prey timing for each possible value of the *prey probability* variable. Each instance included the time at which each prey has to appear in the environment and its exact position. We coupled these instances with an equal number of seeds. The seeds are casually generated numbers that were passed to the random number generator of the simulator to have a fixed sequence of values produced at each simulation.

We chose two setups for the runs in the real world. Five experiments for each experimental setup were sufficient to demonstrate the soundness of the simulation and to validate the results obtained. For real robot trials we employed a subset of the instances generated for the virtual environment.

We decided to evaluate the performances of three learning algorithms. The choice of the first candidate was straightforward. Since our work is in some way the prosecution of Labella's research (Labella, 2003), we evaluated the performance of the *Variable Delta Rule* designed by the author.

The second algorithm we decided to include in our research is ALLIANCE. Designed by Parker (1998), it was created for Multi Robot Systems. It shows some similarities with Labella's *Variable Delta Rule* (robots adapt their behaviour in an automatic way even when a centralized knowledge is not present) and it is well known in mainstream robotics.

The last candidate was an algorithm designed by Li (2002). It was particularly suitable for our purposes, because it is intended to specialize the agents adapting only one parameter. Moreover, it was created to work with swarms of robots.

The final part of the work was the run of the experiments and the statistical analysis of the results.

7.2 Evaluation of the Results

The most important result we obtained in our work was to prove the robustness of our methodology. Using specific statistical tests, we demonstrated the soundness of the simulation with a very limited number of experiments performed in the real environment. This led to a significant saving of both resources and time. Moreover, we detailed guidelines to compare different learning strategies in Swarm Robotics.

Also the parameters we chose to rate the performances of the three algorithms led to interesting results. As we showed, ALLIANCE always scores better than the other two when we consider efficiency, but it never reaches the results of Labella's *Variable Delta Rule* and of Li's learning strategy in terms of captured prey.

Analyzing the histograms we presented in the previous chapter, we notice that ALLIANCE does not allow the agents to specialize according to different environmental situations. The final *Task Time* values, that represent the time each agent spends in the nest before starting a new prey retrieval trial, are almost constant in every experimental setup.

In conclusion, the choice of one of the three analyzed algorithms for real-world applications depends on the situation where the agents have to be employed. For instance, we could consider using ALLIANCE as learning strategy when efficiency is more important than the number of captured prey. It could be the case of a number of rovers that must retrieve samples on a distant planet. On the other hand, it is evident that Li's algorithm or Labella's *Variable Delta Rule* could give better results in terms of cleanliness, performing the already mentioned garbage collection task on the streets of a town.

7.3 Further Work

There are surely many ways to continue researching in the field we explored. The first, and most obvious, is to consider new algorithms for comparison, using the same test field we prepared for the three we examined in our work. We could also explore a larger number of experimental setups, increasing the number of robots in each colony or deploying them in environments with different values of *prey probability*.

The case of heterogeneous colonies could also be explored, that is, the use of robots having different specific capabilities or performances on a specific task. In such a situation, it could be interesting to analyze if the learning algorithms can lead to a different specialization degree among the agents.

Another interesting topic could emerge from the observation that, in our research, we did not find experimental evidence of the reason why efficiency, with all the strategies examined, decreases when the *size* of the swarm is incremented. A possible further research could focus on this issue, evaluating if this situation is caused by problems related to the learning strategies or, alternatively, by the way we decided to measure efficiency in our work.

Nevertheless, considering the huge number of different tasks to which Swarm Robotics is applied, we believe that the most interesting kind of prosecution of our work could be to expand our methodology to new test fields. Our procedure could allow swarm robotic system designers to improve their control algorithms sparing resources and time, drawing scientifically sound conclusions about their performances with a limited number of real experiments.

Bibliography

R.S. Aylett and D.P. Barnes. A multi-robot architecture for planetary rovers. In *Proceedings of 5th ESA Workshop on Space Robotics (ASTRA'98)*, 1998.

E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: from natural to artificial systems.* Oxford University Press, New York, USA, 1999.

R.A. Brooks. Artificial life and real robots. In F.J. Varela and P. Bourgine, editors, *Proceedings of the First European Conference on Artificial Life*, pages 3–10, Cambridge, MA, 1992. MIT Press/Bradford Books.

J. Bruce and M. Veloso. Real-time randomized path planning for robot navigation. In *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '02).* IEEE Press, New York, NY, USA, 2002.

S. Camazine, J.-L. Deneubourg, N. R. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau. *Self-Organisation in Biological Systems.* Princeton University Press, Princeton, NJ, USA, 2001.

Y.U. Cao, A.S. Fukunaga, and A.B. Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4(1):7–27, 1997.

J.-L. Deneubourg, S. Goss, J. M. Pasteels, D. Fresneau, and J.-P. Lachaud. Self-organization mechanisms in ant societies (II): Learning in foraging and division of labor. In J. M. Pasteels and J.-L. Deneubourg, editors, *From Individual to Collective Behavior in Social Insects*, volume 54 of *Experientia Supplementum*, pages 177–196. Birkhäuser Verlag, Basel, Switzerland, 1987.

C. Devigne and C. Detrain. Marquage exploratoire chez la fourmi lasius niger. *Actes colloques Insectes Sociaux*, 14:156–159, 2002.

M. Dorigo and E. Şahin. Guest editorial. *Autonomous Robots*, 17(2-3):111–113, 2004.

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

D.W. Gage. Command control for many-robot systems. In *Proceedings of AUVS-92, Technical symposium of the association for unmanned vehicle systems*, pages 22–24, Huntsville, AL, 1992.

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

P.P. Grassé. La reconstruction du nid et les coordinations inter-individuelles chez *Bellicositermes natalensis* et *Cubitermes*. La théorie de la stigmergie: essai d'interpretation des termites constructeurs. *Insectes Sociaux*, 6:41–83, 1959.

R. Groß. Swarm-intelligent robotics in prey retrieval tasks. Technical Report TR/IRIDIA/2003-27, IRIDIA, Universite Libre de Bruxelles, Brussels, Belgium, 2003. Thesis for the *Diplome d'Etudes Approfondies (DEA).*

S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, (6):65–70, 1979.

A.J. Ijspeert, A. Martinoli, A. Billard, and L.M Gambardella. Collaboration through the exploitation of local interactions in autonomous collective robotics: the stick pulling experiment. *Autonomous Robots*, 11(2):149–171, 2001.

N. Jakobi, P. Husbands, and I. Harvey. Noise and the reality gap: the use of simulation in evolutionary robotics. In F. Moran, A. Moreno, J. J. Merelo, and P. Chacon, editors, *Advances in Artificial Life: Proceedings* *of the Third European Conference on Artificial Life*, volume 929 of *Lecture Notes in Computer Science*, pages 704–720. Springer Verlag, Heidelberg, Germany, 1995.

H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. RoboCup: the robot world cup initiative. In W. Lewis Johnson and Barbara Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 340–347, New York, 1997. ACM Press.

T.H. Labella. *Prey retrieval by a swarm of robots.* Thesis for the Diplôme d'Études Approfondies (DEA). Technical Report TR/IRIDIA/2003-16, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2003.

T.H. Labella. Learning as a means of efficient collaboration and division of labour in a swarm of robots. Technical report, IRIDIA, Université Libre de Bruxelles, 2004.

T.H. Labella, M. Dorigo, and J.-L. Deneubourg. Efficiency and task allocation in prey retrieval. In A.J. Ijspeert, D. Mange, M. Murata, and S. Nishio, editors, *Proceedings of the First International Workshop on Biologically Inspired Approaches to Advanced Information Technology (Bio-ADIT2004)*, Lecture Notes in Computer Science, pages 32–47. Springer Verlag, Heidelberg, Germany, 2004a.

T.H. Labella, M. Dorigo, and J.L. Deneubourg. Self-organized task allocation in a group of robots. Technical Report TR/IRIDIA/2004-6, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2004b.

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

L. Li. *Distributed learning in swarm systems: a case study.* Unpublished Master's Thesis. Technical report, California Institute of Technology, Pasadena, California, 2002.

L. Li, A. Martinoli, and Y.S. Abu-Mostafa. Learning and measuring specialization in collaborative swarm systems. *Adaptive Behavior*, 12(3-4): 199–212, 2004.

A.-C. Mailleux, J.-L. Deneubourg, and C. Detrain. Regulation of ants' foraging to resource productivity. In *Proceedings of the Royal Society, 270*, pages 1609–1616, 2003.

A.C. Mailleux, J.-L. Deneubourg, and C. Detrain. How do ants assess food volume? *Animal behaviour*, 59:1061–1069, 2000.

A. Martinoli and F. Mondada. Collective and cooperative behaviours: Biologically inspired experiments in robotics. In *4th International Symposium on Experimental Robotics, ISER'95, Stanford, CA*, pages 2–7, 1995.

M.J. Matarić. Issues and approaches in the design of collective autonomous agents. *Robotics and Autonomous Systems*, 16:321–331, 1995.

M.J. Matarić. Reinforcement learning in the multi-robot domain. *Autonomous Robots*, 4(1):73–83, 1997.

M.J. Matarić. Coordination and learning in multi-robot systems. *IEEE Intelligent Systems*, 13(2):6–8, 1998.

F. Mondada, A. Guignard, M. Bonani, D. Bär, M. Lauria, and D. Floreano. Swarm-bot: from concept to implementation. In *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robot and Systems (IROS 2003)*, pages 1626–1631, Las Vegas, Nevada, US, 2003. IEEE Press.

F. Mondada, A. Guignard, A. Colot, D. Floreano, J.-L. Deneubourg, L. Gambardella, S. Nolfi, and M. Dorigo. Swarm-bot: a new concept of robust all-terrain mobile robotic system. Technical report, LSA2 - I2S - STI, Swiss Federal Institute of Technology, Lausanne, Switzerland, 2002a.

F. Mondada, G.C. Pettinaro, I. Kwee, A. Guignard, L.M. Gambardella, D. Floreano, S. Nolfi, J.-L. Deneubourg, and M. Dorigo. Swarm-bot: a swarm of autonomous mobile robots with self-assembling capabilities. In *Proceedings of the International Workshop on Self-organisation and Evolution of Social Behaviour*, Monte Verita, Switzerland, 2002b.

D.C. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons, New York, NY, USA, 5th edition, 2000.

L.E. Parker. L-ALLIANCE: Task-oriented multi-robot learning in behavior-based systems. *Journal of Advanced Robotics*, pages 305–322, 1997.

L.E. Parker. ALLIANCE: an architecture for fault tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, 1998.

S. Portha, J.-L. Deneubourg, and C. Detrain. Self-organized asymetries in ant foraging: a functional response to food type and colony needs. *Behavioral Ecology*, 13:776–781, 2002.

S. Russel and P. Norvig. *Artificial Intelligence - A Modern Approach, Second Edition*. Pearson Education, 2003.

P. Rybski, A. Larson, H Veeraraghavan, M. LaPoint, and M. Gini. Performance evaluation of a multi-robot search & retrieval system: experiences with MinDART. Technical Report 03-011, Department of Computer Science and Engineering, University of Minnesota, MN, USA, 2003.

E. Şahin, T.H. Labella, V. Trianni, J.-L. Deneubourg, P. Rasse, D. Floreano, L. Gambardella, F. Mondada, S. Nolfi, and M. Dorigo. SWARM-BOTS: pattern formation in a swarm of self-assembling mobile robots. In A. El Kamel, K. Mellouli, and P. Borne, editors, *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, Hammamet, Tunisia, 2002. Piscataway, NJ: IEEE Press.

S. Siegel and N. J. Castellan Jr. *Nonparametric Statistics for the Behavioral Sciences*. Statistics Series. McGraw-Hill, Singapore, second edition, 1988.

G. Theraulaz and E. Bonabeau. A brief history of stigmergy. *Artificial Life*, 5(2):97–116, 1999.

V. Trianni, S. Nolfi, and M. Dorigo. Hole avoidance: experiments in coordinated motion on rough terrain. In F. Groen, N. Amato, B. Bonarini, E. Yoshida, and B. Kröse, editors, *Intelligent Autonomous Systems 8*, pages 29–36. IOS Press, Amsterdam, The Netherlands, 2004.

G. Weiß. Adaptation and learning in multi-agent systems: some remarks and a bibliography. In G. Weiß and S. Sen, editors, *Adaption and Learning in Multi-Agent Systems*, volume 1042 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 1995.

Appendix A

Simulator UML

We present here four UML class diagrams of our simulator, MindS-miss. The diagrams do not include all the classes external to our program, such as KODEX or ODE.

After a first, global overview of the logical schema of the software, we focus on three important groups of classes that are analyzed in more detail.

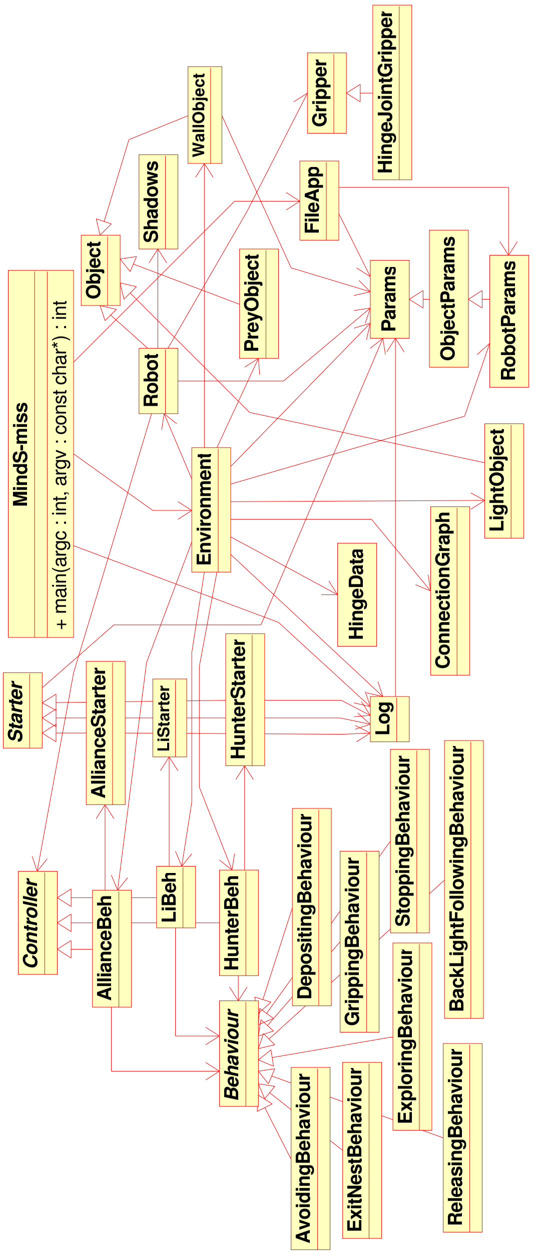


Figure A.1: This UML class diagram represents an overview over the simulator. We do not show here, for practical reasons, attributes and methods of each class.

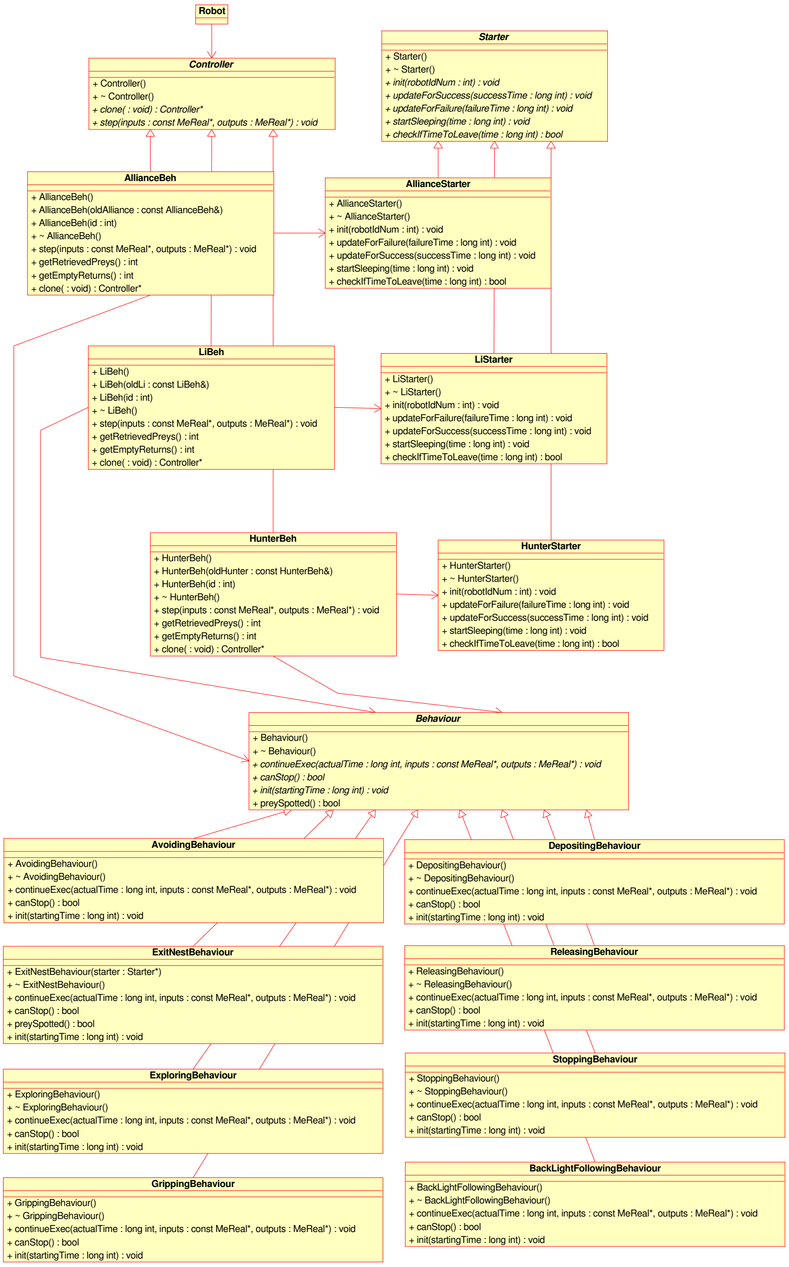
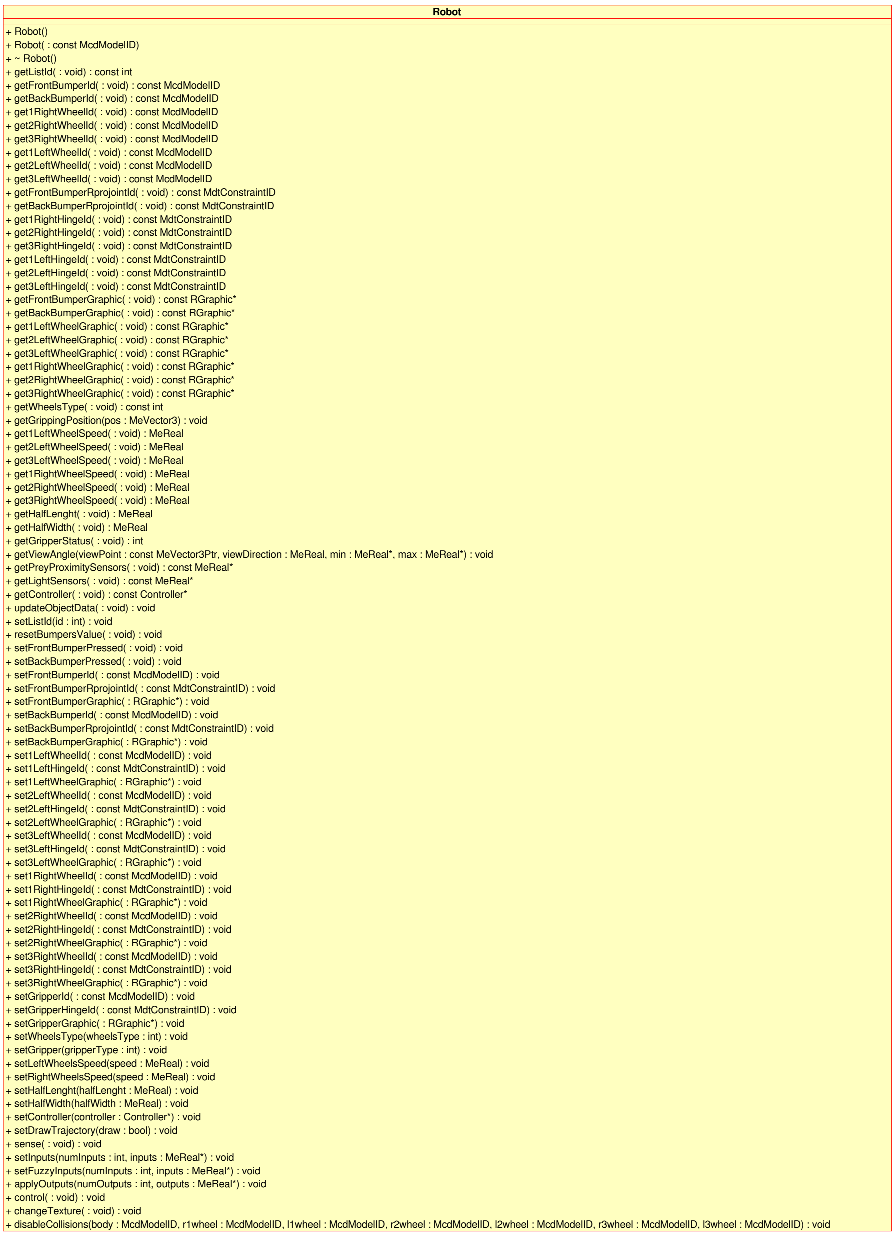

Robot
Controller
+ Controller()
+ ~ Controller()
+ clone(: void) : Controller*
+ step(inputs : const MeReal*, outputs : MeReal*) : void
Starter
+ Starter()
+ ~ Starter()
+ init(robotIdNum : int) : void
+ updateForSuccess(successTime : long int) : void
+ updateForFailure(failureTime : long int) : void
+ startSleeping(time : long int) : void
+ checkIfTimeToLeave(time : long int) : bool
AllianceBeh
+ AllianceBeh()
+ AllianceBeh(oldAlliance : const AllianceBeh&)
+ AllianceBeh(id : int)
+ ~ AllianceBeh()
+ step(inputs : const MeReal*, outputs : MeReal*) : void
+ getRetrievedPreys() : int
+ getEmptyReturns() : int
+ clone(: void) : Controller*
AllianceStarter
+ AllianceStarter()
+ ~ AllianceStarter()
+ init(robotIdNum : int) : void
+ updateForFailure(failureTime : long int) : void
+ updateForSuccess(successTime : long int) : void
+ startSleeping(time : long int) : void
+ checkIfTimeToLeave(time : long int) : bool
LiBeh
+ LiBeh()
+ LiBeh(oldLi : const LiBeh&)
+ LiBeh(id : int)
+ ~ LiBeh()
+ step(inputs : const MeReal*, outputs : MeReal*) : void
+ getRetrievedPreys() : int
+ getEmptyReturns() : int
+ clone(: void) : Controller*
LiStarter
+ LiStarter()
+ ~ LiStarter()
+ init(robotIdNum : int) : void
+ updateForFailure(failureTime : long int) : void
+ updateForSuccess(successTime : long int) : void
+ startSleeping(time : long int) : void
+ checkIfTimeToLeave(time : long int) : bool
HunterBeh
+ HunterBeh()
+ HunterBeh(oldHunter : const HunterBeh&)
+ HunterBeh(id : int)
+ ~ HunterBeh()
+ step(inputs : const MeReal*, outputs : MeReal*) : void
+ getRetrievedPreys() : int
+ getEmptyReturns() : int
+ clone(: void) : Controller*
HunterStarter
+ HunterStarter()
+ ~ HunterStarter()
+ init(robotIdNum : int) : void
+ updateForFailure(failureTime : long int) : void
+ updateForSuccess(successTime : long int) : void
+ startSleeping(time : long int) : void
+ checkIfTimeToLeave(time : long int) : bool
Behaviour
+ Behaviour()
+ ~ Behaviour()
+ continueExec(actualTime : long int, inputs : const MeReal*, outputs : MeReal*) : void
+ canStop() : bool
+ init(startingTime : long int) : void
+ preySpotted() : bool
AvoidingBehaviour
+ AvoidingBehaviour()
+ ~ AvoidingBehaviour()
+ continueExec(actualTime : long int, inputs : const MeReal*, outputs : MeReal*) : void
+ canStop() : bool
+ init(startingTime : long int) : void
DepositingBehaviour
+ DepositingBehaviour()
+ ~ DepositingBehaviour()
+ continueExec(actualTime : long int, inputs : const MeReal*, outputs : MeReal*) : void
+ canStop() : bool
+ init(startingTime : long int) : void
ExitNestBehaviour
+ ExitNestBehaviour(starter : Starter*)
+ ~ ExitNestBehaviour()
+ continueExec(actualTime : long int, inputs : const MeReal*, outputs : MeReal*) : void
+ canStop() : bool
+ preySpotted() : bool
+ init(startingTime : long int) : void
ReleasingBehaviour
+ ReleasingBehaviour()
+ ~ ReleasingBehaviour()
+ continueExec(actualTime : long int, inputs : const MeReal*, outputs : MeReal*) : void
+ canStop() : bool
+ init(startingTime : long int) : void
StoppingBehaviour
+ StoppingBehaviour()
+ ~ StoppingBehaviour()
+ continueExec(actualTime : long int, inputs : const MeReal*, outputs : MeReal*) : void
+ canStop() : bool
+ init(startingTime : long int) : void
ExploringBehaviour
+ ExploringBehaviour()
+ ~ ExploringBehaviour()
+ continueExec(actualTime : long int, inputs : const MeReal*, outputs : MeReal*) : void
+ canStop() : bool
+ init(startingTime : long int) : void
BackLightFollowingBehaviour
+ BackLightFollowingBehaviour()
+ ~ BackLightFollowingBehaviour()
+ continueExec(actualTime : long int, inputs : const MeReal*, outputs : MeReal*) : void
+ canStop() : bool
+ init(startingTime : long int) : void
GrippingBehaviour
+ GrippingBehaviour()
+ ~ GrippingBehaviour()
+ continueExec(actualTime : long int, inputs : const MeReal*, outputs : MeReal*) : void
+ canStop() : bool
+ init(startingTime : long int) : void

Figure A.2: We detail here the structure of the portion of our software that reproduces the control system of the robots. The classes that hold the mechanisms of the learning strategies we evaluated derive from two virtual classes, *Controller* and *Starter*. All the controllers employ the same basic behaviours, that are described in the lower part of the diagram.



```
Robot

+ Robot()
+ Robot( : const McdModelID)
+ ~ Robot()
+ getListId( : void) : const int
+ getFrontBumperId( : void) : const McdModelID
+ getBackBumperId( : void) : const McdModelID
+ get1RightWheelId( : void) : const McdModelID
+ get2RightWheelId( : void) : const McdModelID
+ get3RightWheelId( : void) : const McdModelID
+ get1LeftWheelId( : void) : const McdModelID
+ get2LeftWheelId( : void) : const McdModelID
+ get3LeftWheelId( : void) : const McdModelID
+ getFrontBumperRprojointId( : void) : const MdtConstraintID
+ getBackBumperRprojointId( : void) : const MdtConstraintID
+ get1RightHingeId( : void) : const MdtConstraintID
+ get2RightHingeId( : void) : const MdtConstraintID
+ get3RightHingeId( : void) : const MdtConstraintID
+ get1LeftHingeId( : void) : const MdtConstraintID
+ get2LeftHingeId( : void) : const MdtConstraintID
+ get3LeftHingeId( : void) : const MdtConstraintID
+ getFrontBumperGraphic( : void) : const RGraphic*
+ getBackBumperGraphic( : void) : const RGraphic*
+ get1LeftWheelGraphic( : void) : const RGraphic*
+ get2LeftWheelGraphic( : void) : const RGraphic*
+ get3LeftWheelGraphic( : void) : const RGraphic*
+ get1RightWheelGraphic( : void) : const RGraphic*
+ get2RightWheelGraphic( : void) : const RGraphic*
+ get3RightWheelGraphic( : void) : const RGraphic*
+ getWheelsType( : void) : const int
+ getGrippingPosition(pos : MeVector3) : void
+ get1LeftWheelSpeed( : void) : MeReal
+ get2LeftWheelSpeed( : void) : MeReal
+ get3LeftWheelSpeed( : void) : MeReal
+ get1RightWheelSpeed( : void) : MeReal
+ get2RightWheelSpeed( : void) : MeReal
+ get3RightWheelSpeed( : void) : MeReal
+ getHalfLenght( : void) : MeReal
+ getHalfWidth( : void) : MeReal
+ getGripperStatus( : void) : int
+ getViewAngle(viewPoint : const MeVector3Ptr, viewDirection : MeReal, min : MeReal*, max : MeReal*) : void
+ getPreyProximitySensors( : void) : const MeReal*
+ getLightSensors( : void) : const MeReal*
+ getController( : void) : const Controller*
+ updateObjectData( : void) : void
+ setListId(id : int) : void
+ resetBumpersValue( : void) : void
+ setFrontBumperPressed( : void) : void
+ setBackBumperPressed( : void) : void
+ setFrontBumperId( : const McdModelID) : void
+ setFrontBumperRprojointId( : const MdtConstraintID) : void
+ setFrontBumperGraphic( : RGraphic*) : void
+ setBackBumperId( : const McdModelID) : void
+ setBackBumperRprojointId( : const MdtConstraintID) : void
+ setBackBumperGraphic( : RGraphic*) : void
+ set1LeftWheelId( : const McdModelID) : void
+ set1LeftHingeId( : const MdtConstraintID) : void
+ set1LeftWheelGraphic( : RGraphic*) : void
+ set2LeftWheelId( : const McdModelID) : void
+ set2LeftHingeId( : const MdtConstraintID) : void
+ set2LeftWheelGraphic( : RGraphic*) : void
+ set3LeftWheelId( : const McdModelID) : void
+ set3LeftHingeId( : const MdtConstraintID) : void
+ set3LeftWheelGraphic( : RGraphic*) : void
+ set1RightWheelId( : const McdModelID) : void
+ set1RightHingeId( : const MdtConstraintID) : void
+ set1RightWheelGraphic( : RGraphic*) : void
+ set2RightWheelId( : const McdModelID) : void
+ set2RightHingeId( : const MdtConstraintID) : void
+ set2RightWheelGraphic( : RGraphic*) : void
+ set3RightWheelId( : const McdModelID) : void
+ set3RightHingeId( : const MdtConstraintID) : void
+ set3RightWheelGraphic( : RGraphic*) : void
+ setGripperId( : const McdModelID) : void
+ setGripperHingeId( : const MdtConstraintID) : void
+ setGripperGraphic( : RGraphic*) : void
+ setWheelsType(wheelsType : int) : void
+ setGripper(gripperType : int) : void
+ setLeftWheelsSpeed(speed : MeReal) : void
+ setRightWheelsSpeed(speed : MeReal) : void
+ setHalfLenght(halfLenght : MeReal) : void
+ setHalfWidth(halfWidth : MeReal) : void
+ setController(controller : Controller*) : void
+ setDrawTrajectory(draw : bool) : void
+ sense( : void) : void
+ setInputs(numInputs : int, inputs : MeReal*) : void
+ setFuzzyInputs(numInputs : int, inputs : MeReal*) : void
+ applyOutputs(numOutputs : int, outputs : MeReal*) : void
+ control( : void) : void
+ changeTexture( : void) : void
+ disableCollisions(body : McdModelID, r1wheel : McdModelID, l1wheel : McdModelID, r2wheel : McdModelID, l2wheel : McdModelID, r3wheel : McdModelID, l3wheel : McdModelID) : void
```

Figure A.3: The class *Robot* holds all the physical information about each agent acting in the simulation. It derives from the generic class *Object* and specifies also which is the controller that is coordinating the robot movement. In the class are present methods to specify the speed of the six wheels and to check the situation of the bumpers.

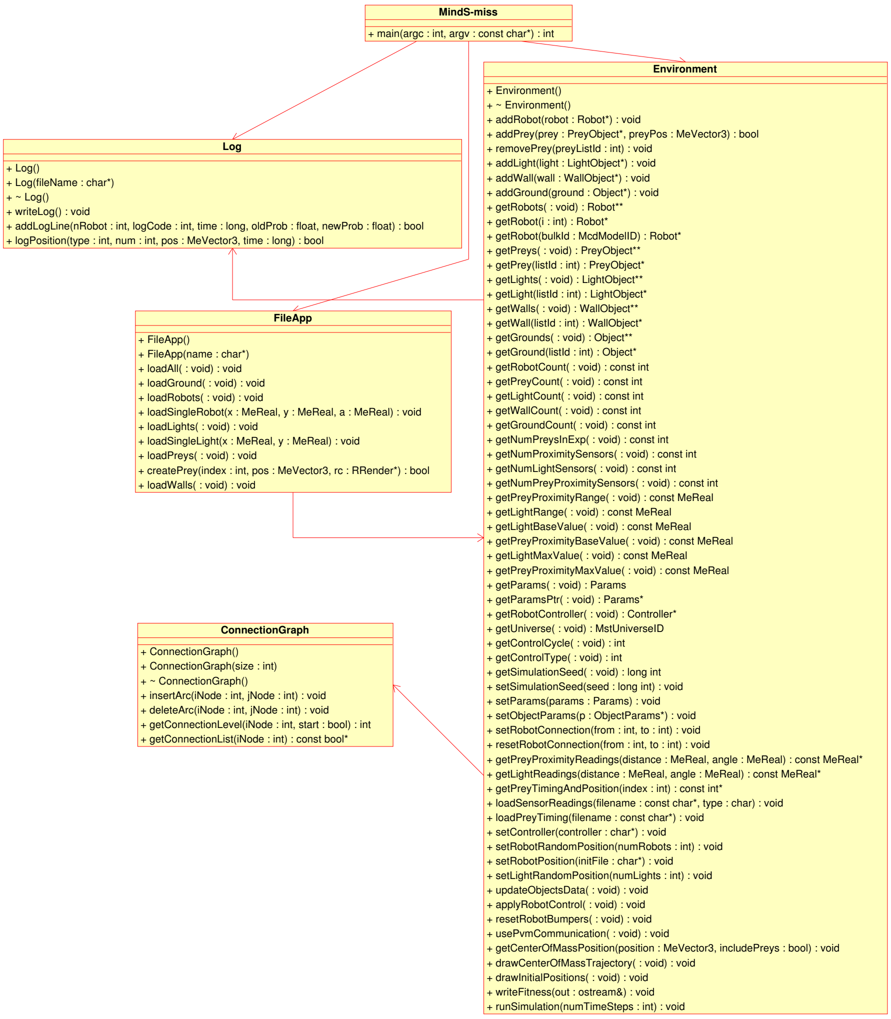

MindS-miss
+ main(argc : int, argv : const char*) : int
Environment
+ Environment()
+ ~ Environment()
+ addRobot(robot : Robot*) : void
+ addPrey(prey : PreyObject*, preyPos : MeVector3) : bool
+ removePrey(preyListId : int) : void
+ addLight(light : LightObject*) : void
+ addWall(wall : WallObject*) : void
+ addGround(ground : Object*) : void
+ getRobots(: void) : Robot**
+ getRobot(i : int) : Robot*
+ getRobot(bulkId : McdModelID) : Robot*
+ getPreys(: void) : PreyObject**
+ getPrey(listId : int) : PreyObject*
+ getLights(: void) : LightObject**
+ getLight(listId : int) : LightObject*
+ getWalls(: void) : WallObject**
+ getWall(listId : int) : WallObject*
+ getGrounds(: void) : Object**
+ getGround(listId : int) : Object*
+ getRobotCount(: void) : const int
+ getPreyCount(: void) : const int
+ getLightCount(: void) : const int
+ getWallCount(: void) : const int
+ getGroundCount(: void) : const int
+ getNumPreysInExp(: void) : const int
+ getNumProximitySensors(: void) : const int
+ getNumLightSensors(: void) : const int
+ getNumPreyProximitySensors(: void) : const int
+ getPreyProximityRange(: void) : const MeReal
+ getLightRange(: void) : const MeReal
+ getLightBaseValue(: void) : const MeReal
+ getPreyProximityBaseValue(: void) : const MeReal
+ getLightMaxValue(: void) : const MeReal
+ getPreyProximityMaxValue(: void) : const MeReal
+ getParams(: void) : Params
+ getParamsPtr(: void) : Params*
+ getRobotController(: void) : Controller*
+ getUniverse(: void) : MstUniverseID
+ getControlCycle(: void) : int
+ getControlType(: void) : int
+ getSimulationSeed(: void) : long int
+ setSimulationSeed(seed : long int) : void
+ setParams(params : Params) : void
+ setObjectParams(p : ObjectParams*) : void
+ setRobotConnection(from : int, to : int) : void
+ resetRobotConnection(from : int, to : int) : void
+ getPreyProximityReadings(distance : MeReal, angle : MeReal) : const MeReal*
+ getLightReadings(distance : MeReal, angle : MeReal) : const MeReal*
+ getPreyTimingAndPosition(index : int) : const int*
+ loadSensorReadings(filename : const char*, type : char) : void
+ loadPreyTiming(filename : const char*) : void
+ setController(controller : char*) : void
+ setRobotRandomPosition(numRobots : int) : void
+ setRobotPosition(initFile : char*) : void
+ setLightRandomPosition(numLights : int) : void
+ updateObjectsData(: void) : void
+ applyRobotControl(: void) : void
+ resetRobotBumpers(: void) : void
+ usePvmCommunication(: void) : void
+ getCenterOfMassPosition(position : MeVector3, includePreys : bool) : void
+ drawCenterOfMassTrajectory(: void) : void
+ drawInitialPositions(: void) : void
+ writeFitness(out : ostream&) : void
+ runSimulation(numTimeSteps : int) : void
Log
+ Log()
+ Log(fileName : char*)
+ ~ Log()
+ writeLog() : void
+ addLogLine(nRobot : int, logCode : int, time : long, oldProb : float, newProb : float) : bool
+ logPosition(type : int, num : int, pos : MeVector3, time : long) : bool
FileApp
+ FileApp()
+ FileApp(name : char*)
+ loadAll(: void) : void
+ loadGround(: void) : void
+ loadRobots(: void) : void
+ loadSingleRobot(x : MeReal, y : MeReal, a : MeReal) : void
+ loadLights(: void) : void
+ loadSingleLight(x : MeReal, y : MeReal) : void
+ loadPreys(: void) : void
+ createPrey(index : int, pos : MeVector3, rc : RRender*) : bool
+ loadWalls(: void) : void
ConnectionGraph
+ ConnectionGraph()
+ ConnectionGraph(size : int)
+ ~ ConnectionGraph()
+ insertArc(iNode : int, jNode : int) : void
+ deleteArc(iNode : int, jNode : int) : void
+ getConnectionLevel(iNode : int, start : bool) : int
+ getConnectionList(iNode : int) : const bool*

Figure A.4: The class *Environment* is a central part of the simulator. Its methods are also used to load all the objects in the arena. The properties and the number of such objects are read from XML files by the *FileApp* class. The outputs of MindS-miss are log files, generated by the class *Log*, where all the information about the running experiment is stored.

Appendix B

Statistical Tests

To validate the results of our simulation, we used two different statistical tests. In the following we summarize their procedures.

B.1 Friedman two-way analysis of variance by ranks

The Friedman two-way analysis of variance by ranks can be applied when the data from the matched samples[1] are in at least an ordinal scale.[2]

It tests the null hypothesis that the three (or, more in general, the k) repeated measures or matched groups come from the same population or from populations with the same median. In our case, for the Friedman test, the data are grouped in a table with three columns (k), that represent the different algorithms, and N rows (the number of experiments performed for each learning strategy). The scores in each row are ranked separately and the test determines the probability that the different columns of ranks come from the same population. If the null hypothesis were true, we would expect

[1] We recall that the experiments make use of the blocking design, which grants the matching of data coming from different algorithms.

[2] Ordinal scale: the objects in one category of a scale are not only different from the objects in other category of that scale, but also stand in some kind of relation to them. Typical relations among classes are: higher, more preferred, more difficult, etc. (Siegel and Castellan Jr., 1988).

the distribution of the ranks in each column to be a matter of chance and the sum of ranks in each column to be:

$$\frac{N(K+1)}{2} \tag{B.1}$$

If the opposite case (null hypothesis H_0 false), then the rank totals would vary from one column to another. The Friedman test determines whether the rank totals (R_j, where j is the column we consider) for each condition (or learning algorithm, in our case) differ significantly from the values which would be expected by chance. To do this test, we compute the value of the statistic which we denote as F (Siegel and Castellan Jr., 1988).

$$F = \left[\frac{12}{Nk(k+1)} \sum_{i=1}^{k} R_i^2\right] - 3N(k+1) \tag{B.2}$$

Once we have the value of F, we can calculate the probability associated with it.

For big sample sizes, a large-sample approximation can be used and the statistic F is distributed approximately as χ^2.

B.2 Random permutation test for paired replicates

Permutation tests, under certain conditions, are the most powerful of the nonparametric techniques and are appropriate whenever measurement is on an interval scale,[3] as it is in our case when considering efficiency values or the percentages of captured prey. It utilises both the direction of the differences within pairs and the relative magnitude, giving more weight to a pair which shows a large difference between the two conditions than to a pair which shows a small difference.

A permutation test assumes that the two scores observed in each pair

[3] Interval scale: a scale that has all the characteristics of an ordinal scale and where the distances or differences between any two members of the scale have meaning.

could be randomly assigned to the algorithms (say X and Y) that generated them. This is what we would expect if the null hypothesis of no difference between the algorithms is true.

We define $d_i = X_i - Y_i$ as the difference for the ith pair and it is a measure of the difference between algorithms, while $\sum_{i=1}^{N} d_i$ is the sum of the differences. If H_0 was true, then the sign of each d_i would be a matter of chance.

The sampling distribution consists of the permutation of the signs of the differences to include all possible 2^N (where N is the number of experiments) occurrences of $\sum_{i=1}^{N} d_i$. The region of rejection consists of those outcomes that have the most extreme $\sum_{i=1}^{N} d_i$'s and it is composed by $2^N * \alpha$ occurrences. If the measured $\sum_{i=1}^{N} d_i$ falls into the region of rejection, we can discard the null hypothesis that the algorithms are equivalent. Moreover, since we use a one-tailed test, we can also deduce which one is better than the other.

Since we are using a large sample, the number of permutations to calculate would be too large. We estimate the p-value by means of the Monte Carlo method. We perform the pairwise comparison of the algorithms, then we apply to the p-values the Holm correction for multiple test (Holm, 1979), in order to have a strong control over the errors.