Universite Libre de Bruxelles
Universite d'Europe
Faculté des Sciences Appliquées
IRIDIA, Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

Division of Labour in Groups of Robots

Ing. Thomas Halva Labella

Promoteur:
Prof. Marco Dorigo

Thése présentée en vue de l'obtention du
titre de Docteur en Sciences Appliquées

Année académique 2006–2007

Division of Labour in Groups of Robots

Ing. Thomas Halva Labella

Doctor
Communitatis
Europeæ

Universite Libre de Bruxelles

"I try all things; I achieve what I can."
(Herman Melville, Moby Dick.)

"Vedere le cose che non si vedono.
Guardare oltre l'orizzonte."
("To see the things that you can not see.
To look beyond the horizon."
Sensei Fugazza)

Audio. Video. Disco.
(Latin: I hear. I see. I learn.)

Abstract

In this thesis, we examine algorithms for the division of labour in a group of robot. The algorithms make no use of direct communication. Instead, they are based only on the interactions among the robots and between the group and the environment.

Division of labour is the mechanism that decides how many robots shall be used to perform a task. The efficiency of the group of robots depends in fact on the number of robots involved in a task. If too few robots are used to achieve a task, they might not be successful or might perform poorly. If too many robots are used, it might be a waste of resources. The number of robots to use might be decided *a priori* by the system designer. More interestingly, the group of robots might autonomously select how many and which robots to use. In this thesis, we study algorithms of the latter type.

The robotic literature offers already some solutions, but most of them use a form of direct communication between agents. Direct, or explicit, communication between the robots is usually considered a necessary condition for co-ordination. Recent studies have questioned this assumption. The claim is based on observations of animal colonies, e.g., ants and termites. They can effectively co-operate without directly communicating, but using indirect forms of communication like *stigmergy*. Because they do not rely on communication, such colonies show robust behaviours at group level, a condition that one wishes also for groups of robots. Algorithms for robot co-ordination without direct communication have been proposed in the last few years. They are interesting not only because they are a stimulating intellectual challenge, but also because they address a situation that might likely occur when using robots for real-world out-door applications. Unfortunately, they are still poorly studied.

This thesis helps the understanding and the development of such algorithms. We start from a specific case to learn its characteristics. Then we improve our understandings through comparisons with other solutions, and finally we port everything into another domain.

We first study an algorithm for division of labour that was inspired by ants' foraging. We test the algorithm in an application similar to ants' foraging: prey retrieval. We prove that the model used for ants' foraging can be effective also in real conditions. Our analysis allows us to understand the underlying mechanisms of the division of labour and to define some way of measuring it.

Using this knowledge, we continue by comparing the ant-inspired

algorithm with similar solutions that can be found in the literature and by assessing their differences. In performing these comparisons, we take care of using a formal methodology that allows us to spare resources. Namely, we use concepts of *experiment* design to reduce the number of experiments with real robots, without losing significance in the results.

Finally, we apply and port what we previously learnt into another application: Sensor/Actuator Networks (SANETs). We develop an architecture for division of labour that is based on the same mechanisms as the ants' foraging model. Although the individuals in the SANET can communicate, the communication channel might be overloaded. Therefore, the agents of a SANET shall be able to co-ordinate without accessing the communication channel.

Acknowledgements

The work presented in this thesis was financially supported by the SWARM-BOTS project, a project funded by the Future and Emerging Technologies programme (IST-FET) of the European Commission, under grant IST-2000-31010, and from the “ANTS” Project, an "Action de Recherche Concertée" funded by the Scientific Research Directorate of the French Community of Belgium. The final part of the work was funded by the Deutscher Akademischer Austauschdienst (DAAD) under the grant 331 4 03 003.

I would like to thank the persons that supervised my work: Dr. Marco Dorigo, Prof. Jean-Louis Deneubourg and Dr.-Ing. Falko Dreßler. Marco has been a severe supervisor, but I learnt a lot from him. I thank him for his criticisms (a lot) and his praises (a few). He was a good guide, as Virgilio was for Dante. (I do not pretend I have the qualities of Dante, but reaching the end of this thesis was surely like Hell). From Jean-Louis, I learnt how to look at problems from the outside (as a biologist) and not only from the inside (as an engineer). Falko introduced me to the vast word of communication networks and was the precious mentor that helped me to find the way.

I wish to thank the research directors of IRIDIA, Prof. Hugues Bersini and Marco Dorigo, for providing an extremely friendly and stimulating research environment. I thank all@iridia. In a purely random order, they are: Mauro & Carlotta, Max & Roberta, Elio, Ciro, Rodi, Campatzis, Anders, Rehan, Paola, Federico, Krzysztof (notwithstanding the number of consonants, he is a good guy), Francisco, Josh & Julia, Gianni, Bruno, Maria & Christian, Tom, Prasanna, Alex, Michael, Andrea, Michela, Fabiola, Davide, Roberto. A special mention goes to Vito, with whom I have shared the office for many years. Thank to him, we all in IRIDIA could remind everyday Catullus’ Poem LXIX. All my sympathy to Michela, who is now living with him. A special thank goes also to Marcello for the wonderful work that he did for Chap. 5. Thanks to my friends in Italy: Ginver, Il Billa, and the Pilgrim from Rome.

Moving here and there in Europe, allowed me to know many people. They helped me to understand different cultures. I thank my trainer in Belgium, Marc, his wife, Fiorella, and all the people at the Kei Shin Kan Dojo in Brussels. Thanks also to my training-fellows in Italy: Paola (sorry for leaving you alone in a grey city), Fabio & Lidia, Fabio, Fabio, Alessandro, Sensei Fugazza, and all the others. Thanks also to the people I met in Germany: Gerhard, Ulrich, all members of Post-SV

Karate Abteilung.

My family has supported me for many years, and are still doing it although I think they have not really understood what I was doing (neither did I sometimes). Thank you Mom and Patty. I miss you.

I should thank another person, but I do not find the words to describe how helpful she was for me. She was next to me when I had bad times and she stood my mood changes. Without her, I would have been locked long time ago in a hospital for mental illnesses. With all my love, I want to thank you, Daniela.

Finally, I wish that my father was here to see also this moment occurring. He has missed as much of my and my family's, as we are missing him.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Robots have lived in the mind of mankind probably since the beginning of its history. Obviously not in the way we know them nowadays. They were sometimes made of living flesh (Frankenstein), sometimes of wood (Pinocchio), or also of mud (the Golem). The words "robot" and "robotics" are in fact quite recent,[1] but the idea of autonomous artifacts created by men was always there. The progress of technology in the last half-century allowed us to start seriously thinking to design and develop autonomous artifacts. This led to the birth of robotics as a scientific field.

Computer science, semi-conductor electronics and robotics were born nearly together, somewhere in the middle of the 20th century. Today, we are surrounded by computers and electronic devices, which entered into a nearly symbiotic relationship with mankind. Where are the robots? If we take aside those used for factory automation, where are the robots that used to populate our science fiction books and movies?

Robotics for industrial automation has enormously developed from its beginning. We are now able to build (nearly) fully automated factories, but we have not yet a robot that can prepare and bring us a decent coffee. Robotics outside industries in fact poses several problems that are hard to solve. Additionally, human beings are limited designer, because we can not fully understand and foresee the problems that might occur in open environments. It is our opinion that the lack of rigorous procedural methods is also responsible to the under-development of robotics.

This thesis does not offer a solution to all the problems, especially the coffee one. On the opposite, we study only one particular problem of autonomous robotics: division of labour. We show how our problem can be solved taking inspiration from biology. Differently to what commonly observed in the robotic literature, we base our study on a formal research methodology. We do think that if other researchers follow our example, it might in the long term help autonomous robotics to exit the universities and enter the real world.

[1]"Robot" comes from the Czech word "robotnik" and appeared in English in 1923. "Robotics" was used for the first time by Asimov [1991] in a 1941 science fiction story.

This thesis contributes to the research in robotics, and not only, under different aspects. We need to give the reader a brief overview of the background and the motivations of our work before pointing out our contributions. This is the content of the following section. Section 1.2 summarises the main contributions of our work. Section 1.3 describes the structure of this thesis and lists the publications of ours that are related to each chapter.

1.1 Background and Motivations

The main contributions of this thesis cover two aspects of robotics: communication-less division of labour in Multi Robot Systems (MRSs), and comparisons of different control algorithms. This is a rough summary and a gross categorisation of our work, but it is sufficient for the goals of this section. The following subsections introduce these aspects and clarify the background and the motivations of our research.

1.1.1 Multi Robot Systems

Robotics mainly focused on the development of single robots during its youth. Emphasis was on issues such as, for instance, how to sense and represent the environment, how to decide on the base of the representation and how to recognise and react to changes. The last decade has seen more and more research on groups of robots. These systems are also known as Multi Robot Systems. They share the same problems as single robots, but they pose additional issues. For example: how and what shall the robots communicate to each other or how to co-ordinate their actions to achieve a goal. Even more recently, researchers have studied heterogeneous systems like Sensor/Actuator Networks (SANETs). These systems comprise mobile robots and a network of fixed sensors. Robots and sensors can communicate usually through a wireless channel.

This thesis addresses one of the core issues of MRSs: what is the point of using more than one robot? Only one robot could be sufficient for most of the applications cited in the literature. Let us take the application used for most of our thesis: search and retrieval of objects (Sec. 4.1). More robots can retrieve more prey than one alone. More robots require however more power and more complex co-operation schemes. More robots also create more interferences among the group members. Are more robots also more efficient than a single one? How can we improve the efficiency of a group of robots?

There are many strategies to improve the efficiency. One can program the robots better, or build them better. Another option, the one we study in the next chapters, is to improve the efficiency by means of autonomous division of labour: the group selects, without human intervention, which and how many robots should be involved in a task.

More interestingly and following a recent research line, the mechanisms for division of labour in which we are interested shall not use direct communication or any representation of the environment. The reasons for this choice are explained below. In the following thesis we refer to communication as always direct communication. *Communication-less* algorithms are therefore algorithms that do not use direct communication. This definition is however not totally correct because these algorithms might use some other form of communication, e.g. *stigmergy* (Sec. 2.5.1). We could discuss for many pages and chapters about the meaning of the words “direct”, “communication” and also “representation”, but this would be out of the scope of this thesis.[2] This point deserves nevertheless more elaboration because it is an important working hypothesis of ours.

We say that communication is direct if an observer can find in the code that controls a robot a set of instructions that explicitly send a message to another robot. This is a very rough definition, but it is sufficient for the context of this thesis.

Communication-less co-operation algorithms have been proposed only recently. To understand their usefulness is better first to describe the traditional (with communication) approach to collective robotics.

Traditional Approach to Robot Group Design

The most important difference between robots for automation and science-fiction-like robots is the environment that surrounds them. In a factory, robots are along an assembly line. They are always in a set of predefined states. Also the pieces on which they are working are in a set of predefined states. A number of precautions are taken in designing the factory in order to guarantee that this condition is always met. This is thus defined a *constrained* environment as opposed to a *free* environment.

Free environments pose more difficult problems. The most important and most commonly addressed in the literature are the uncertainty of the environment and adaptation to new conditions.

Suppose that one wants to use robots to explore an unknown environment. A typical example could be distant planets. We do not need however to go so far to find unknown environments. Our oceans are probably more unknown than the universe [Schätzing, 2006]. The fundamental problem is that the environment is not unknown only to the robots, but also to their designer.

Uncertainty has usually been tackled by having robots first explore the environment and build a map of it. Each robot explores a part of the environment and communicates its findings to the others. Once a map is available, the robots can use it to plan their actions. There are many sub-problems that have to be solved to pursue this approach, for instance:

- how to elaborate the sensor signals to extract the information useful to build the map;

[2]For instance, Di Paolo [1998] says that “in most studies of the evolution of communication [. . .], authors either provide a new definition of [communication], or at least find it necessary to revise previous definitions”. For an introduction to the problem of defining representation, the reader can look at Harvey [1996].

- how robots can know their position in order to update the map and to plan their actions (this problem is know as *localisation*);
- how to recognise changes in the environment, and to take fast decisions if they are dangerous for the robot;
- how to reduce the sensor noise that might invalidate the results of the previous points;
- how to know and recognise which other robots are present, what they are doing and where they are;
- how to assure the availability of a communication medium for information exchange.

The last point includes:

- the definition of a reliable (i.e., persistent) physical transmission medium;
- the definition of a medium access protocol;
- the definition of a networking protocol;
- the definition of common semantics and ontology for the content of the information to be exchanged.

The last point includes that robots will be able to communicate only what they were instructed to say. They can not create new concepts and can not communicate things that were not foreseen.

A number of researchers have recently questioned this approach. They especially criticise the use of communication channels and planning. In fact, all the problems listed above are due to the decision of building a map and of communicating it. They are not directly related to the original problem, that is, the uncertainty of the environment.

These criticisms are based on recent studies on self-organisation and biological systems (Sec. 2.5.1). These studies have shown that animal societies can achieve goals equally or more complex than those of the robots, but with simpler individual behaviours and using much simpler communication systems. The field in which these researchers are active is called Swarm Intelligence (SI).

The Swarm Intelligence Approach

The swarm intelligence approach to robot design can be summarised in the following way: do not make complex robot behaviours, but exploit the complexity of the environment. The robots are seen as part of a complex dynamic system. There is no more any distinction between robots on one side and environment on the other, but they are merged in a holistic system. Each robots' action is going to modify the environment in a complex way. The other robots can perceive the modifications and react accordingly on the base of simple rules. This is indeed the way in which, for instance, termites, bees and ants build their nests.

We think that the major contribution from SI, or better its application to robotics, Swarm Robotics (SR) (Sec. 2.5), is to clearly show that the use of communication, planning, mapping (or any other explicit representation of the environment) are not a necessity. They are used because they have been common part of the design process: modelling, division in modules, specification of modules interfaces and interactions. This process sees the environment merely as a passive element. SI suggests to use it actively.

Some researchers in this area claim that the traditional approach is wrong, because it is not akin to anything in nature. We consider this a pretty fundamentalist position. On the other hand, researchers of the traditional approach defend their position with claims like: “if my robot has communication capability, why should I not make use of it?”. They then develop their algorithms accordingly. We do not agree with this position either. There are in fact some conditions where it is not possible to have communication, For example:

- The communication channel is not available because it is being set up. For instance, during the synchronisation of the robots to access the wireless channel. The physical transmission devices of the robots have to be in phase when accessing the wireless channel for sending or receiving air frames. If they are not, air frames might get lost or collide. Getting in phase requires robots to co-ordinate, but they can not communicate yet.

- In underwater robotics, acoustic communication remains the only practical methodology for long-range communication. Acoustic communication is however fundamentally limited by the speed of sound in water (around 1500 m/s) and by the fact that acoustic attenuation increases with the sound frequency. The result is that the best communication bandwidth is usually in the range 2400–4800 Baud. The further the robots are, the narrower the bandwidth becomes [Whitcomb, 2000]. It can also reach few bits per second. Such bandwidth can saturate easily and become useless, thus the robots should co-ordinate reducing, or eliminating, the communication.

- Exploration on a planet, which is under magnetic storm. The wireless communication channel is too noisy to be useful.

- The available radio frequencies of a wireless channel are already overloaded (e.g., robots used for rescue after a disaster).

Our interest in communication-less division of labour is not then a mere theoretical exercise. It is based on the remark that the traditional approach in some application domains can not cope with situations likely to occur.

1.1.2 Comparisons of Different Solutions for MRS

When comparing the robotic literature with that of some other field of science, one particular difference appears. There are very few works in robotics which compare solutions obtained following different methodologies. In particular, to the best of our knowledge, no such comparisons have been done in the field of SR.

Let us take for instance the field of Operation Research. Researcher continuously compare their algorithm with others on a particular problem. The best algorithm becomes the “state of the art”. New ideas are then compared and tested against the current state of the art. These experiments do not exist in robotics, neither does the concept of “state of the art” . The works in the literature mostly show particular solutions to particular problems, but there are very few comparisons.

On the one hand, this is understandable. Let us consider a normal cycle of research in autonomous robotics (not for industrial purposes). Researchers usually start developing a prototype of the robot. This is then used to run experiments, during which the problems of the prototype arise. Researchers have to come back and fix them on the hardware. Then experiments can start again but the robots, being still prototypes, often require maintenance. When the hardware starts to be reliable, there is the problem of the power sources: the current robots used for research last for a few experiments in a row, then the robots must stop and the batteries be recharged. When the first round of experiments is over, the fellowship of the researcher too, and a new researcher will have to start from scratch. As it can be seen, the experimental part is highly resource consuming. Thus, it is not surprising that researchers prefer to use the robots mostly for proof-of-concept experiments than for comparisons with other solutions.

On the other hand, this should not be taken for excuse. Other scientific areas face the same problems, but they nevertheless do comparisons. For instance, doctors test drugs on their patients to know which one is more effective. In case of long lasting illnesses, a research might also last ten years. Researchers in these areas have developed a number of different techniques which allows them to reduce the resource usage. A branch of statistics, called *experiment design*, is dedicated to this purpose. We would like to see more of it in the field of robotics.

1.2 Main Contributions

The main contributions of this thesis are as follows:

- Introduction of a new algorithm for division of labour in a group of robots. The algorithm is inspired by ants’ foraging behaviour. It is based on a simple form of learning and does not use any form of direct communication.

- Validation of the algorithm through a formal and methodological use of both simulation and real robots.

- Formal comparisons among different algorithms for division of labour. The formal methodology makes use of concepts from experiment design, which are never used in robotics—at least explicitly. They help us to assess which algorithms are the best within a given set using a minimal number of experiments (recall that they are the most time consuming part of the research). This work paves the way to the development, one day, of the concept of “state of the art algorithm” also in robotics.

- We do not only take inspiration from biology, but we give something back too. The validation of our algorithm for division of labour strongly supports the argument that learning is an important factor in division of labour and that no communication is required in animal societies. There are still many biologists against this argument.
- We developed a new type of simulator for SANETs. The simulator, called BARAKA (Sec. 3.3), combines two different simulation paradigm: discrete event simulation and continuous time simulation.
- Implementation of a new algorithm for division of labour with the choice of more than one task. While the case with many tasks is studied in mainstream robotics, this is new in SR and bio-inspired robotics.
- A new approach to the development of SANETs. This approach features a stricter interaction between application and network layers in robot and sensor node.

1.3 Structure of the Thesis and Relevant Publications

Chapter 2 gives a more detailed introduction on MRSs. It starts with the narration of the birth of the first robots, both real and imaginary. It describes then the history and development of MRSs, highlighting the current problems and giving an overview on current works. The chapter continues describing the concept of self-organisation as it was developed in biology and how this is used as foundation for SR. After having read it, the reader should have all necessary background information for the following chapters.

Chapter 3 describes the tools that were used for the experiments of this thesis. We used both real robots and simulators. The real robots were built using Lego Mindstorms^{TM}. The simulators were either adapted or totally developed for the experiments. The following publications are related to the content of this chapter:

- 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.
- M. Dorigo, E. Tuci, R. Groß, V. Trianni, T.H. Labella, S. Nouyan, C. Ampatzis, J.-L. Deneubourg, G. Baldassarre, S. Nolfi, F. Mondada, D. Floreano, and L.M. Gambardella. The SWARM-BOTS

project. In E. Şahin and W. Spears, editors, *Proceedings of the First International Workshop on Swarm Robotics at SAB 2004*, volume 3342 of *Lecture Notes in Computer Science*, pages 31–44. Springer Verlag, Berlin, Germany, 2004b.

- T.H. Labella, I. Dietrich, and F. Dressler. BARAKA: A hybrid simulator of sensor/actuator networks. In *Proceedings of the Second IEEE/Create-Net/ICST International Conference on COMmunication System softWAre and MiddlewaRE (COMSWARE 2007)*, Bangalore, India, January 7–12, 2007. In press.

Chapter 4 describes a new algorithm for division of labour. The algorithm was developed after a model of ants' foraging. This model was previously tested only through numerical simulations. We provide an empirical and stronger validation of the model. We also analyse how the division of labour changes according to different environmental parameters, and how the group performs in changing environments. This work was published in:

- T.H. Labella, M. Dorigo, and J.-L. Deneubourg. Efficiency and task allocation in prey retrieval. In A.J. Ijspeert, M. Murata, and N. Wakamiya, editors, *Biologically Inspired Approaches to Advanced Information Technology: First International Workshop, BioADIT 2004*, volume 3141 of *Lecture Notes in Computer Science*. Springer Verlag, Heidelberg, Germany, 2004a.
- T.H. Labella, M. Dorigo, and J.-L. Deneubourg. Self-organised task allocation in a group of robots. In R. Alami, editor, *7th International Symposium on Distributed Autonomous Robotic Systems (DARS04)*, pages 371–380, Toulouse, France, June 23–25, 2004b.
- T.H. Labella, M. Dorigo, and J.-L. Deneubourg. Division of labour in a group of robots inspired by ants' foraging behavior. *ACM Transactions on Autonomous and Adaptive Systems*, 1(1):4–25, 2006a.

Chapter 5 considers other algorithms that might be used for division of labour. We evaluate algorithms similar to ours and compare their results. This allows us to clearly state which are the differences between the algorithms, and also gives us some hint on why they are different. The experiments were done using a formal methodology which allowed us to spare a lot of resources and to get, for instance, valid results after only five replications with real robots. This work, which was done in collaboration with Marcello Cirillo, a Master student under our supervision, is new and not yet published. Publications are on-going work at the time of writing.

Chapter 6 extends the work of the previous chapters in the context of SANETs. In a SANET, robots have to interact with sensor nodes. They are small devices that can sense the environment, send and receive messages. The chapter focuses on the division of labour in a SANET when there are more than one task to choose. Related publications are:

- T.H. Labella, G. Fuchs, and F. Dressler. A simulation model for self-organised management of sensor/actuator networks. In *Fachgespräch über Selbstorganisierende, Adaptive, Kontextsensitive verteilte Systeme (SAKS)*, University of Kassel, Germany, March 23–24 2006b.

- T.H. Labella and F. Dressler. A bio-inspired architecture for division of labour in SANETs. In *Proceedings of the First IEEE/ACM International Conference on Bio Inspired Models of Network, Information and Computing Systems (BIONETICS 2006)*, Cavalese, Italy, December 11–13, 2006. In press.

Finally, Chap. 7 draws the final conclusions of our work.

1.3.1 Additional Publications

We published also other articles during our research that are not strictly related to the topic of this thesis. The following list completes our publication list:

- A. Bonarini, M. Matteucci, G. Invernizzi, and T. H. Labella. Context and motivation in coordinating fuzzy behaviors. In M. Colombetti, A. Bonarini, and P.L. Lanzi, editors, *Proceedings of the Seventh Meeting of the Italian Association for Artificial Intelligence (AI*IA 2000)*. AI*IA, Milano, Italy, 2000.

- A. Bonarini, M. Matteucci, G. Invernizzi, and T.H. Labella. An architecture to coordinate fuzzy behaviors to control an autonomous robot. *Fuzzy Sets and Systems*, 134(1):101–115, 2001.

- E. Şahin, T.H. Labella, V. Trianni, J.-L. Deneubourg, P. Rasse, D. Floreano, L.M. Gambardella, F. Mondada, S. Nolfi, and M. Dorigo. SWARM–BOT: Pattern formation in a swarm of self–assembling mobile robots. In A. El Kamel, K. Mellouli, and P. Borne, editors, *Proceedings of IEEE International Conference on System, Man and Cybernetics (SMC2002)*. IEEE Press, New York, NY, 2002.

- V. Trianni, T.H. Labella, R. Groß, E. Şahin, M. Dorigo, and J.-L. Deneubourg. Modeling pattern formation in a swarm of self-assembling robots. Technical Report IRIDIA-TR-2002-12, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2002.

- V. Trianni, R. Groß, T.H. Labella, E. Şahin, P. Rasse, J.-L. Deneubourg, and M. Dorigo. Evolving aggregation behaviors in a swarm of robots. In W. Banzhaf, T. Christaller, P. Dittrich, J.T. Kim, and J. Ziegler, editors, *Advances in Artificial Life - Proceedings of the 7th European Conference on Artificial Life (ECAL)*, volume 2801 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, Berlin, Germany, 2003.

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

- V. Trianni, T.H. Labella, and M. Dorigo. Evolution of direct communication for a swarm-bot performing hole avoidance. In M. Dorigo, M. Birattari, C. Blum, L.M. Gambardella, F. Mondada, and T. Stützle, editors, *Ant Colony Optimization and Swarm Intelligence – Proceedings of ANTS 2004 – Fourth International Workshop*, volume 3172 of *Lecture Notes in Computer Science*, pages 131–142. Springer Verlag, Berlin, Germany, 2004.

- T.H. Labella and M. Birattari. Polyphemus: De alieni generorum abacorum racemo. Technical Report IRIDIA-TR-2004-15, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2004.

- M. Dorigo, E. Tuci, V. Trianni, R. Groß, S. Nouyan, C. Ampatzis, T.H. Labella, R. O'Grady, M. Bonani, and F. Mondada. *Computational Intelligence: Principles and Practice*, chapter SWARM-BOT: Design and Implementation of Colonies of Self-assembling Robots. IEEE Computational Intelligence Society, New York, NY, 2006.

Chapter 2

Background

In this thesis, we study systems composed of more than one robot. Before entering into the details of our research, we want to guide the reader into this field. This chapter can not unfortunately be a complete guide to robotics. Even a rough summary might take more space than this thesis.

We start by giving a historical perspective in Sec. 2.1. We describe some of the tentatives of men to build moving artifacts. Looking at them with nowadays eyes, they look everything but scientific works. In Sec. 2.2, we give a loot at nowadays robotics, giving an overview of the different research areas. We start from a gross classification of the areas, to reach the more restricted sector where we carried out our work. Section 2.3 gives a more proper classification of the specific area of interest in this thesis, Multi Robot Systems (MRSs), and Sec. 2.4 lists some examples of research in the area. Finally, Sec. 2.5 describes a recent approach to MRS, the one we follow in this thesis.

2.1 A Brief History of Robotics

Robotics can be defined as "the technology dealing with the design, construction, and operation of robots in automation"[1] and has developed since the second half of the 20th century. It is a broad and multidisciplinary area. It spans over electronics, mechanics, computer science, artificial intelligence, control theory, and also bio-informatics.

The English word "robot" is older than the technological applications. It appeared in 1923, in the English translation of a 1920 Czech play, "R.U.R." (Rossum's Universal Robots), by Karel Čapek (1890–1938). It is astonishing, if we consider that Turing will propose only thirteen years later his model of a computing machine, which today is known as "Universal Turing Machine" [Turing, 1936]. The original Czech word, "robotnik", means "slave". It comes from "robota" which means "forced labour".[2] The word refers to human artifacts that should

[1]Merriam Webster on-line dictionary at `http://www.m-w.com/`.
[2]The complete etymology can be found at `http://www.etymonline.com/`.

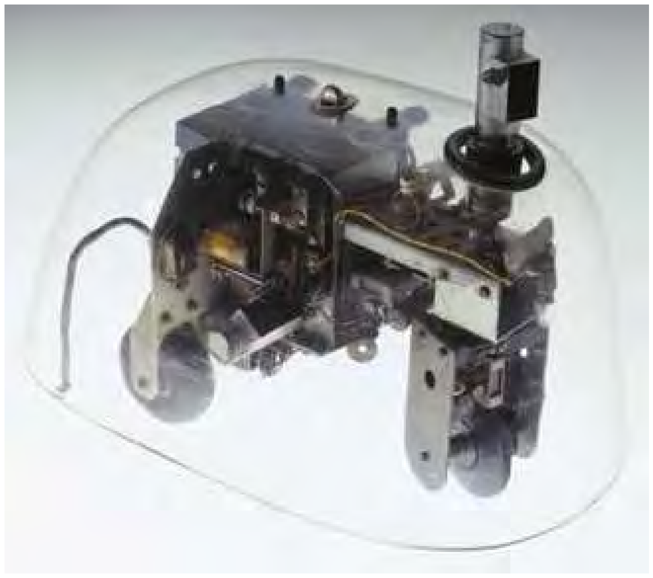


Figure 2.1: A replica of the tortoise by Grey Walter.

perform work for men. It differs from other machines built by men because of the strict integration between the mechanical, electronic and logic components, and for its ability of moving and exploring/modifying the environment.

The concept of a “robot” is one variation of a theme that is really common in the literature and in mythology: the myth of creation. We go at least as far back as the ancient legend of Cadmus, who sowed dragon teeth that turned into soldiers, and the myth of Pygmalion, whose statue of Galatea came to life. In classical mythology, Hephaestus created mechanical servants, ranging from intelligent, golden handmaidens to more utilitarian three-legged tables that could move about under their own power. A Jewish legend tells of the Golem, a clay statue animated by Kabbalistic magic. Similarly, in the Younger Edda, Norse mythology tells of a clay giant, Mökkurkálfi or Mistcalf, constructed to aid the troll Hrungnir in a duel with Thor, the God of Thunder. Other examples in more “modern” literature comprise *Pinocchio* [Collodi, 1883] and *Frankestein* [Shelley, 1818].

The first application of robotics as commonly understood today started to appear after World War II. In 1948, Grey Walter at Bristol University, England, created the first autonomous mobile robots, although he “programmed” them using analogical circuits. The “tortoise” (Fig. 2.1), as it was called, was wired to display animal-like behaviours, like approaching light sources using a photo-receptor. The *Unimate*, the first industrial arm robots, appeared in 1961 and joined the General Motors’ assembly line. In 1966, the Stanford Research Institute produced “Shakey”, the first robot which was able to plan its actions. This period can be considered the beginning of robotics as a scientific and technological discipline. However, the word “robotics” is older than

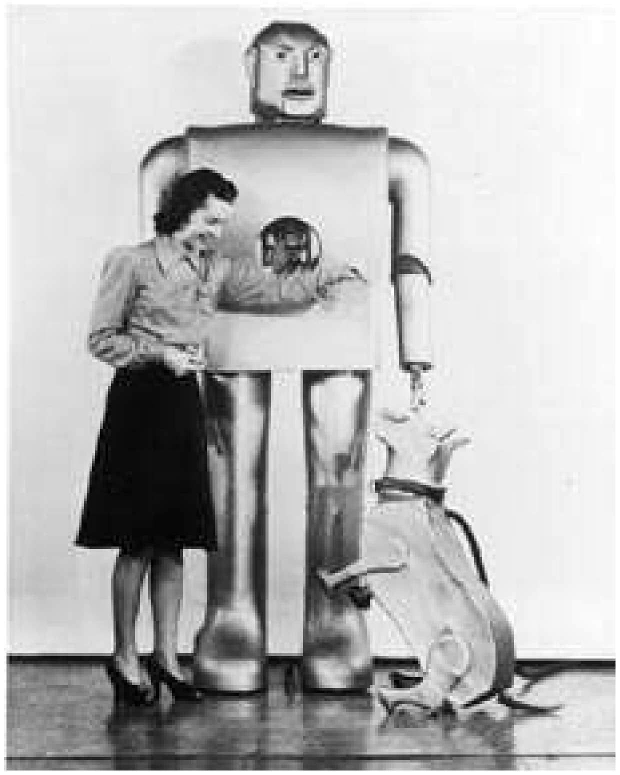


Figure 2.2: Elektro, in the middle, and Sparko, right. The robots were created by Westinghouse in the 30s. Nothing is known about the humanoid on the left.

these examples. It appeared in a 1941 science fiction story by Asimov [1991]: science fiction anticipated reality.

Before World War II, humans built a number of machines that show striking resemblance with the contemporary idea of a robot. We can go far back in time, to ancient Greece. Around 350 BCE, the Greek mathematician Archytas of Tarentum (428–347 BCE—the city is now called Taranto, in south Italy) built "The Pigeon", a mechanical bird used for studies on flight. In 1495, Leonardo da Vinci (1452–1519) designed a mechanical knight that was apparently able to sit up, wave its arms, and move its head and jaw. It is not known however if he actually built this device. In 1738, the french engineer Jacques de Vaucanson (1709–1782) built two "automata" which were able to play the flute and tambourine, plus a duck made of four hundreds moving parts. The duck is considered his masterpiece: it could flap its wings, drink water, digest grain and defecate. In 1898, Nikola Tesla (1856-1943) made a demonstration with a remotely-operated boat. Last, but not least, Westinghouse created in the 30s a human-like robot called Elektro and its companion dog Sparko (Fig. 2.2 and 2.3). Elektro was able to move legs and arms, to turn the head and even to smoke. Sparko was also a masterpiece of technology: it could stay sit.

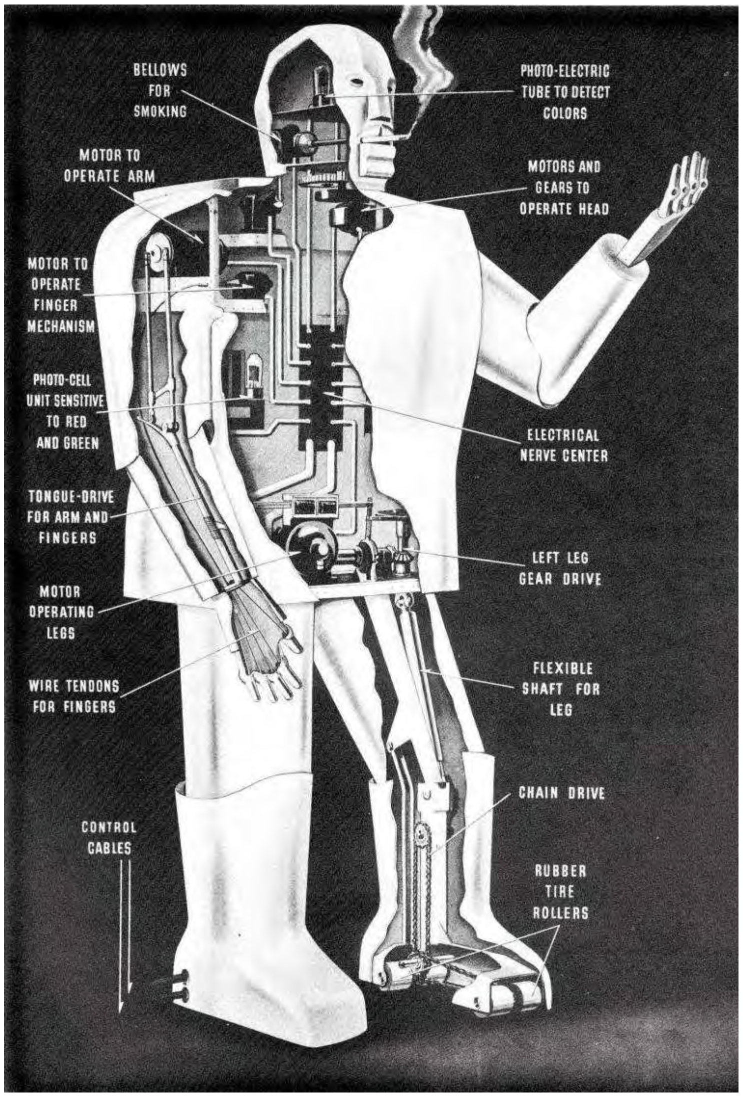


Figure 2.3: Internals of Elektro

2.2 Robotics Nowadays: a Gross Classification

After robotics moved out of the science fiction context, it expanded considerably. It is possible to classify the works of the last years in a number of ways, which are usually not mutually exclusive. We do not want to list all of them in this chapter. We make use of some broad classification, in order to immediately cut away those works that are not directly related with what we present in this thesis. What follows should not be considered as an authoritative and accepted categorisation of robotics. It is only meant as an aid to the reader to place the research field of this thesis in a broader context.

Robots can be remote-controlled or autonomous. The main difference is that a remote-controlled robot executes a set of low level commands coming from a remote operator. For example, it could be a robot used for submarine exploration or by a surgeon for a delicate operation. In case of autonomous robots, the computer on board has some autonomy in taking decisions on which action should be done according to the information coming from the sensors.[3] For an autonomous robot to work, it is essential that it is provided with some means of perceiving the environment (i.e., the sensors) and of modifying it (i.e., the actuators).

Autonomous robots can be fixed in the environments or can be free to move. An assembly line in a factory is an example of fixed robot, the Sojourner rover which landed on Mars in 1997 is a quite convincing example of a moving one.

Different works in the literature focus on different aspects of a robot: the mechanics, the electronics, or the control program. The latter refers to the program that makes one or more robots move and act in the environment and is the focus of this thesis. Control programs can be divided into several layers. At the lower level, there is fine control of the actuators and the drivers of the sensors. At the middle layer, there is usually the code that takes care of elaborating the raw data coming from the sensors in order to have a better and more schematic representation of the state of the environment. This part is usually called *feature extraction*. Finally, at the top level, there is the decision centre: it reads the information coming from the sensors and uses it to send the appropriate commands to the actuators. This part of program is usually called the “high level control”. In this thesis, we also refer to it using the words “robot program” or “robot controller”.

The final gross classification useful for the moment concerns the number of robots taken into account in the experiments. During the first decades of robotics, experiments were done using only one robot. This was due most likely to the high costs of the robots. They were also made of very delicate pieces, and required a lot of maintenance. In the last two decades, researchers have given more and more attention to systems made of several robots, on which we focus now. They are

[3]Obviously, a robot can never be totally autonomous, at least until it will be able to switch itself on and off.

called *Multi Robot Systems (MRSs)*.

2.3 Overview of Multi Robot Systems

The recent increase of interest in Multi Robot Systems was certainly helped by the reduction of hardware prices. This alone does not explain why so many researchers have started working on this subject. As some author proposed, MRSs are interesting because some

> *[. . .] tasks may be inherently too complex (or impossible) for a single robot to accomplish, or performance benefits can be gained from using multiple robots; building and using several simple robots can be easier, cheaper, more flexible and more fault tolerant than having a single powerful robot for each separate task; and the constructive, synthetic approach inherent in cooperative mobile robotics can possibly yield insights into fundamental problems in the social sciences (organisation theory, economics, cognitive psychology), and life sciences (theoretical biology, animal ethology).* [Cao et al., 1997, page 7.]

We focus below only on works in which robots collaborate in order to achieve a task, a sub-field of MRSs that is often called *co-operative robotics*. The reader should remember however that there are also works in which autonomous robots are in competition among them. The most spectacular case is probably RoboCup.[4] In a RoboCup competition, two teams of robots (either simulated or real) play against each other in a soccer match. Scientific and practical issues include combining reactive approaches and modelling/planning approaches, real-time recognition, planning, reasoning, strategy acquisition, and agent modelling [Kitano et al., 1997, Asada and Kitano, 1999, Asada et al., 1999].

Even if we look only at co-operative robotics, the literature offers a massive number of works. To account for all of them, researchers have proposed several classification criteria [see Cao et al., 1997, Farinelli et al., 2004, Gerkey and Matarić, 2004, for some examples]. The most extensive and acknowledged classification was proposed by Dudek et al. [1996]. They propose a number of taxonomic axes and then list for each axis the solutions usually found in the literature. They also review the best known works in order to classify them. Their paper is useful therefore not only for their classification criteria, but also as a basis for an overview of MRSs. The axes they propose are as follows:

Size : a multi robot system can be formed by one robot (the “minimal collective” [Dudek et al., 1996, p. 379]), two robots (the “simplest group”), or by a limited number n of robots.

Communication range : in most systems, each robot can communicate only to a certain distance. To the extremes of this axis, there are robots which do not communicate (their range is 0) or which can communicate with all the other robots in the environments (so

[4] `http://www.robocup.org/`

that the range can be considered as infinite). Limitless communication is a property that depends both on the characteristic of the environment and on the means of communication. Between the two extremes, there is the case in which robots can communicate only with nearby robots. Their communication range is finite and smaller than the environment. The authors refer however only to direct forms of communication. Even in case of a zero-range, "it is possible for robots to communicate with each other *indirectly* by observing their presence, absence or behaviour (as many animals seem to)" [Dudek et al., 1996, p. 380].

Communication topology : if robots communicate, they might not directly communicate with every other robot, even if they are near. In some cases, there is a hierarchy along which messages should pass. Dudek et al. identified four common communication topologies in the literature: robots can broadcast their messages, which are received by any other robot within communication range; communication is address-based, so that each robot can communicate with only one other robot; communication must follow the path in a tree that defines the hierarchy; finally, the robots are linked in a general graph, along whose edges robots communicate.

Communication bandwidth : there can be costs related to communication which affect the available bandwidth. There might be no costs and thus infinite bandwidth or high costs and low bandwidth. Between them, Dudek et al. place another category, which is relevant for the applications discussed in Sec. 2.5. For these kinds of robots, the costs of communicating are comparable to those of moving in the environment. In fact, the robots communicate changing the environment. The extreme case is the one without bandwidth at all. In such case, robots are not even able to sense each other. This is considered to be an "impractical case if coordinated collective behaviour is desired" [Dudek et al., 1996, p. 381].

Collective reconfigurability : it refers to the rate at which the collective can spatially reorganise itself and it is usually related to the communication range and topology. Robots can have fixed positions, change their position occasionally following communication (as in formation control) or have completely dynamic and arbitrary locations.

Processing abilities : there are several software architectures that can be used for the controllers of the robots. The common and general models are: a finite state automata, a push-down automata, Turing machines and Neural Networks.

Collective composition : the robots in a MRS might be different both in the hardware and in the software. Differences in the former usually imply differences in the latter. This is why Dudek et al. give only three possible values for this axis: groups where robots are completely identical (hardware and software), where they are physically homogeneous and groups which are heterogeneous. The authors point out that even in case of identical robots, they can still assume different roles based on environmental and statistical factors.

The control system of a single robot operating alone, that is, not a MRS, can be described as follows: given a condition, or state, of the environment, the robot has to perform a series of actions to reach a goal. The goal can be to have an object carried from one place to another, to have the robot in a fixed position, to move so that all the environment has been explored, and so forth. The control system analyses sensor data to identify the current state and to select the best action in order to reach the goal. There are two approaches to the decision process: a *reactive* system considers only the current state and binds it to a specific action, creating a mapping from the state space to the action space; a *planning* approach selects the best action according to predictions of future states which are elaborated using a model of the environment. Sensor data is then reanalysed to decide on the next action and to give feedback to the system.

When dealing with only one robot, control algorithms must take care of a number of problems, listed as follows:

Unpredictable changes. If the environment is *dynamic*, it can change because of factors that are not under the control of the robot. If a reactive architecture is used, the state-action mapping may not be valid any more. If planning is used, the foreseen states do not occur and new predictions are needed, but the environment can change again before the new predictions are available.

Sensor reliability. Sensors can be unreliable or they can return the same data in different states (a phenomenon known as *sensor aliasing* or *partial observability*). For instance, if a new state of the environment was reached, e.g., the goal is closer, and the control system reads the same data as at the beginning, the robot would think that no progress has been made and select the wrong action.

Unknown environments. If a robot worked in a new environment, the outcome of any of its actions might be partially or completely unknown. In this case the model used for the planning or the mapping in a reactive system may be wrong and ineffective. Experience collected in the past can be used by the robot to improve the state-action mapping or the model of the environment, for instance by means of *reinforcement learning* (App. A).

Real time requirements. The product space between state and action is generally huge. Looking for the best action in case of planning, or for the best mapping in the case of a reactive system, can take too much time. This is usually a problem when the robot has to operate in a critical environment where decisions must be taken fast or where fast learning is desired.

MRSs amplify these problems. Each robot must take care of the others. From the point of view of each robot, the environment is more dynamic because the other robots can change it by means of their actions. Each control system shall consider both the actual state of the environment and all possible combinations of the states of the other robots. This gives rise to a combinatorial increase of the state space with the number of robots. Moreover, if no communication is used, the states of other robots are unknown, hence the environment is only partially observable. If communication is used, the capacity of the communication channel can easily saturate with a high number of robots.

2.4 Examples of MRSs

The following examples show some of the problems and the solutions that have been studied in the literature on MRSs. They show a sort of "mainstream" approach for the design of MRSs on which we discuss in the following section.

Goldberg and Matarić [2000] use Augmented Markov Models (AMMs) to control the robots. An AMMs is a kind of Markov Chain that is incrementally generated through node splitting in order to catch hidden states. They use a foraging task to test their algorithm: robots have to search the environment for different objects and retrieve them to a predefined area (this task is described in more detail in Sec. 4.1). The authors wanted to tackle the problem of dynamic environments. They use in each robot several AMMs. Each of them is used to track a different timescale. This feature is tested in environments where the number of the items to retrieve changes during the experiments.

Matarić [1997b] addresses the problem of learning in a puck retrieval task when the state-action space is large. The dimension is drastically decreased using behaviours instead of actions as basic blocks of the decision process. Behaviours are goal-driven control laws that achieve sub-goals and are not learnt during experiments. She uses a reinforcement function that is the sum of three different components. The first one takes care of internal events triggered by behaviour activations, like the collection or the dropping of pucks. The second one considers the distance to neighbours and has a positive value when it increases. The third one, initialised when a puck is grasped, gives positive reinforcement when the distance to the home decreases while carrying a puck.

In Matarić [1997a], the task is to learn social behaviours in order to reduce interference among robots that is "an unavoidable aspect of multi-agent interaction and is the primary impetus behind social rules" [Matarić, 1997a, p. 192]. Reinforcement is composed again of three parts. A progress estimator gives a reward whenever a progress toward the immediate goal is done. The second reinforcement comes from the observation and imitation of the behaviours of the others. Finally, a third reward is given by other agents in order to share reinforcement when involved in social interactions.

Balch and Arkin [1994] analyse the effects of communication in

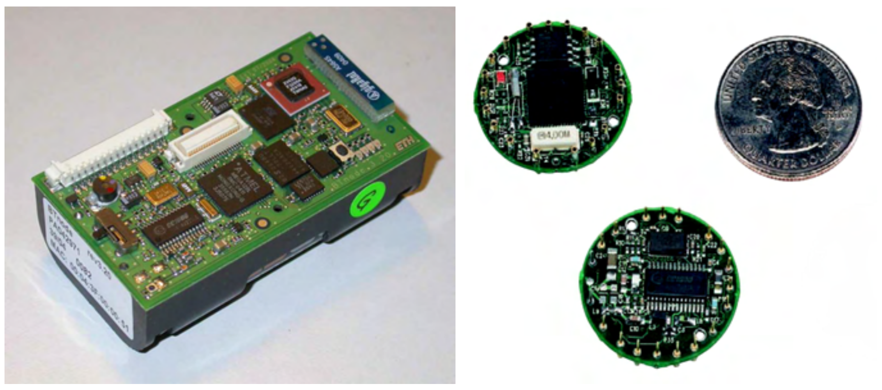


Figure 2.4: Pictures of sensor nodes. On the left a picture of a BTNode. On the right, front and bottom of a MICA2DOT.

foraging and other tasks. Their conclusion is that communication improves performances when little environmental communication is available. However, it is not essential for tasks which include implicit communication and the benefit of using complex communication strategies instead of simple ones is small.

Other researchers have focused on issues related to multi-robot planning. A plan is usually created in a centralised fashion and then distributed to each robot to be executed, as in Bruce and Veloso [2002]. Their planning method is based on Rapidly-Exploring Random Trees (RTT) and is used for navigation tasks. The planning algorithm expands a path one step from the current position to a goal or, with a small probability, toward a random position. This algorithm has a good performance when the application is as real-time constrained and dynamic as a RoboCup match.

Interesting forms of learning can be studied when the system is composed of two groups of robots in competition. Riley and Veloso [2002] try to learn the opponent's strategies during a RoboCup match to find a counter-strategy to apply. A centralised system keeps statistics about the opponent's positions and the ball movements to feed a number of different opponent models. The best fitting one is then used to plan the counter-strategy.

A recent research field studies a group of mobile robots interacting with a Sensor Network (SN). A SN is made of a number of "sensor nodes", also known as "motes". Sensor nodes are small embedded systems, provided with their own power supplies, with the ability of sensing the environment, processing data and communicating with the neighbours via a wireless channel. Figure 2.4 shows two types of sensors that were developed in recent years. Sensor networks can be applied to several areas [Akyildiz et al., 2002]: military applications, environmental protections (e.g., flood or fire detection), health (e.g., monitoring of physiological data of patients in a hospital), home automation, and so on (see Culler and Mulder [2004] for a general overview).

Systems made of mobile robots and sensor nodes are called Sensor/Actuator Networks (SANETs). The USC Robotics Research Lab was probably among the first ones to study SANET [Sukhatme and Matarić, 2000]. The robots they use are called *Robomotes* [Dantu et al., 2005, Sibley et al., 2002], but are provided with limited actuators. In fact, they can be considered more as mobile sensors than robots, because they can modify the environment only by moving. Chapter 6 deals with SANETs. The reader can find in Sec. 6.1 more details about them.

2.5 Challenging the Traditional Approach

The previous examples of MRSs show something in common. Although they differ in the applications and in the architectures of the controllers, the algorithms are designed to learn, handle, and update a model of the environment. The models were the AMM, topological map of the environment, strategies of the opponent and so on. This approach has been recently questioned.

Researchers in robotics have looked into biological system to take inspiration for their control algorithms. The reasons are easy to understand. Some biological systems comprise many animals which have to work together, exactly as in a MRS. The observed final behaviour of the group is usually robust to changes in the environment. The animals work in a distributed fashion and the resulting behaviour scales well with the number of individuals involved. Moreover, the group can easily adapt to unknown environments. These are all precious qualities that every researcher wishes for her/his MRS.

The next subsection describes some of the most important findings in biological systems. We discuss how they can be applied in robotics in Sec. 2.5.2.

2.5.1 Biological Systems and Self-organisation

Throughout its history, computer science has now and then tried to apply ideas coming from biology and other natural sciences to its domain. We observed the birth of *Genetic Algorithms*, *Cellular Automata* and *Neural Networks*, just to name a few. The same is happening with robotics. In fact, Nature is full of examples of systems made of many individuals that can collaborate with each other in order to achieve complex behaviours. Such behaviours are often robust with respect to changes in the environment. Moreover, the final result is usually above the capabilities of the individuals. In this section, we are going to briefly introduce and describe some of these biological systems.

These systems are instances of *self-organising* systems. Self-organisation usually refers to a broad range of pattern-formation processes, such as the stripes of zebras and tigers (Fig. 2.5) [Camazine et al., 2001]. It is also used for sand grains assembling into rippled dunes, chemical reactants forming swirling spirals and fish joining together in schools.

A self-organising system is characterised by the following components [Camazine et al., 2001]:

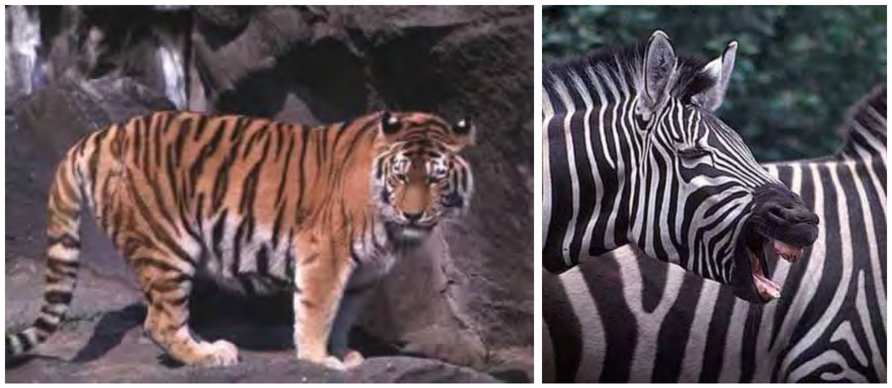


Figure 2.5: Examples of self-organised pattern. Images copied with permission from Camazine et al. [2001].

Positive feedback: it occurs when small perturbations of the system change its dynamics in such a way as to increase the perturbations themselves. The feedback leads to a snowball amplification of the original perturbations. Perturbations can come from changes in the environment or by some random behaviour of the system.

Negative feedback: it is the opposite of the positive process feedback. Perturbations of the system changing its dynamic have the effect of reducing the causes of the perturbations. In self-organising systems, it usually starts having effects after positive feedback and has the purpose of keeping the system under control. Without negative feedback, a system can literally explode.

Information gathering from one's neighbours: individuals in a self-organising system are usually unable to perceive all the environment and act solely on the base of the information they can collect from the local neighbourhood. For instance, a fish in a school follows the direction of the fish immediately around it.

Information gathering from work in progress: if the individuals are involved in a collective effort, as a colony of termites building a nest, stimuli coming from the emerging structure can be a source of information for the individuals. Such form of communication is called *stigmergy* and was proposed for the first time by Grassé [1959]. Stigmergic communication explains why social insects, for instance, are able to build structures as complex as their nest without a central instance directing the work.

Turing [1952] explained how patterns like those in Fig. 2.5 can emerge. He proposed an interaction between two chemicals, a reactant and an inhibitor, which diffuse at different speeds. The most important point of his theory is that the system is not at equilibrium. The result is locally determined by the concentration of the two chemicals. The equation that Turing proposed are now known as *Reaction-Diffusion equations*.

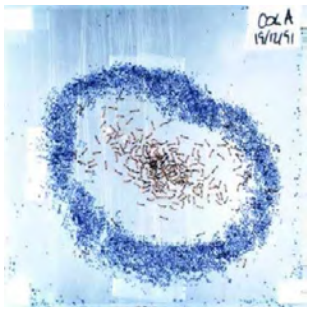
Self-organisation is found in cells as well as in the behaviours of animals. We cited above the case in which termites build their nest. It occurs also in other building processes by other animals, such as ants (Fig. 2.6(a) and 2.6(b)). Ants of the family *Leptothorax albipennis* colonise crevices in rocks. They do not have therefore to build a roof or a floor for their nest, but they build a wall around the brood using sand grains, leaving a small entrance. The area surrounded by the wall is proportional to the size of the colony, and it is adapted if some of their nest-mates are removed. Additionally, these ants can adapt the shape of the wall to the cavity and exploit, for instance, partial walls made of rocks [Camazine et al., 2001]. Two are the main behaviours involved in this process. First, ants collect sand grains and bring them near the brood, depositing them with higher probability where more grains are already present (this is an example of stigmergy). Second, the ants inside the nest push the grain outwards. The probability of each ant both of depositing a grain and of picking it up increases with the grain density (positive feedback). There is also an additional effect, given by the pheromone emitted by the brood in the centre. The pheromone concentration decreases when moving away from the brood. Ants have a higher probability of dropping their grains if this concentration is small. All these factors, combined together, are enough to explain the pattern observed in Fig. 2.6(a).

Similar procedures, in which the action of an individual depends only on what it perceives in its neighbourhood, can also be used to simulate the construction process of, for instance, a wasp nest. The results are shown in Fig. 2.6(c) and are explained in detail by Camazine et al. [2001].

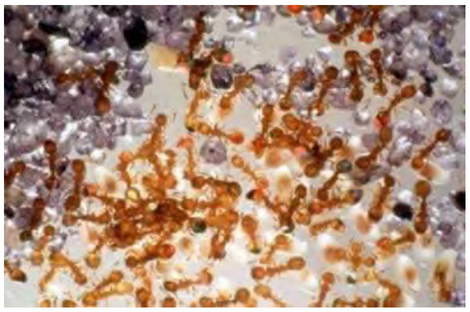
Self-organisation appears also in case of aggregation of animals. Ants can, for instance, form chains to reach a target far away or to pull together leaves for their nest (see Fig. 2.7).

2.5.2 Swarm Robotics

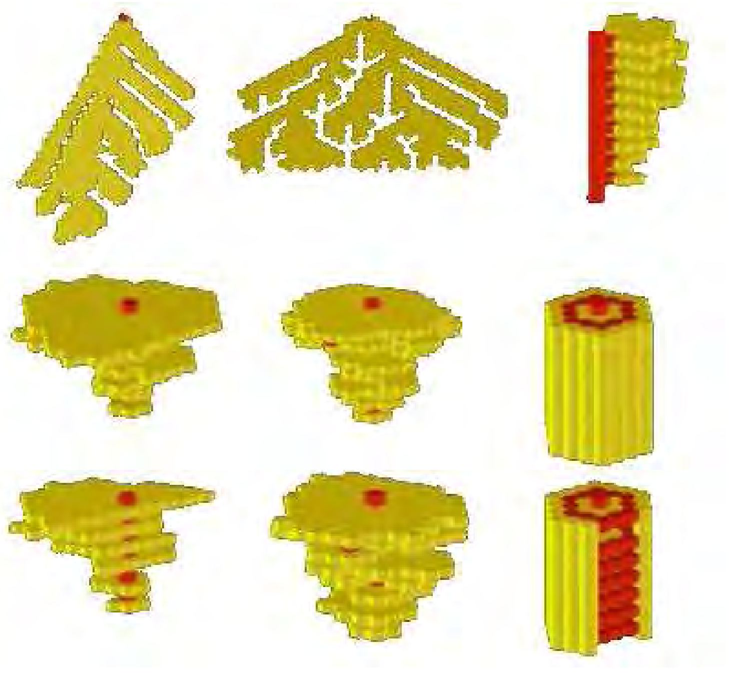
The word *self-organisation* has been used more and more often in computer science and lately also in robotics. It is at the base of Swarm Intelligence (SI) [Beni and Wang, 1989, Bonabeau et al., 1999]. SI can be described as “any attempt to design algorithms or distributed problem-solving devices inspired by the collective behaviour of social insect colonies” [Bonabeau et al., 1999]. Martinoli [2001] points out that the SI approach emphasises “parallelism, distributedness, and exploitation of direct (agent-to-agent) or indirect (via the environment) local interactions among relatively simple agents”. SI depends on decentralised local control of a large number of simple agents. The role of the environment is stressed, although this environment is usually virtual rather than real. A swarm intelligence system will contain no explicit model of the environment, individual agents can both receive information about the environment and act on that environment to



(a) *Leptothorax albipennis* ants building a circular wall using grains of sand.



(b) Closeup on *Leptothorax albipennis* ants while building the wall.



(c) Simulation of collective building using cubic and hexagonal 3D lattices.

Figure 2.6: Examples of self-organised collective building. Images copied with permission from Camazine et al. [2001].

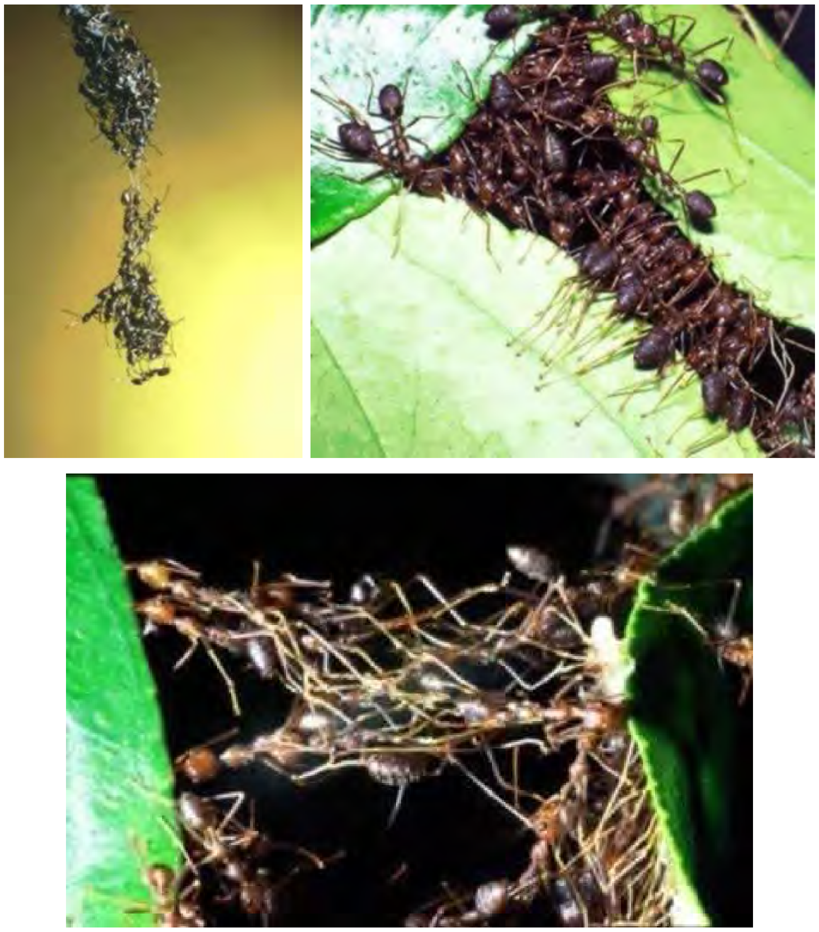


Figure 2.7: Example of self-organised aggregation. Images copied with permission from Camazine et al. [2001].

change it.

When applied to robotics, SI is referred to as Swarm Robotics (SR). This definition might however be criticised, because there are a "plethora" of definitions of SR "often with vague and overlapping meanings". The quotes are from Dorigo and Şahin [2004] that propose some criteria to label a work as belonging to SR: (i) should aim for scalability and should be relevant for the coordination of large numbers of robots; (ii) should consist of a small number of homogenous groups, each containing a large number of members; (iii) should be based on robots that are ill-equipped to carry out tasks on their own and that perform better when they cooperate; and (iv) should be based on robots that should have only local and limited sensing and communication ability.

We have to point out that the characterisation of self-organisation given in Sec. 2.5.1 is only one of the possible. Self-organisation can be found also in physics, mathematics and chemistry. Every discipline has its own description of self-organisation, which can be slightly different from the others. For instance, stigmergy does not play an important role in the examples of SR that we describe later. We chose to give the biological definition because biology is the source of the algorithm studied in Chap. 4.

Control algorithms in SR are very simple and do not use complex representations of the environment or of the other robots (compare this approach to a Partially Observable Markov Decision Process (POMDP), as described in App. A). Achievement of a task is solely based on inter-robot interactions. Attention is more on robot–robot and robot–environment interactions than on the control systems, which are consequently mostly reactive. The use of probabilistic decisions is also common, and the communication is mostly indirect.[5]

The main advantage of SR, among those listed in the literature, is that the control architecture of the robot is scalable and can be used both for a few or thousand units. This leads to robust behaviour of the group, because robots can be easily added and removed and because of the redundancy of the units. Moreover, since the control algorithms do not depend on a detailed model of the environment and of the others, the group of robots can easily adapt to unknown and dynamic environments.

It should be noted that other works in the literature have a similar approach, but they refer to themselves as *Minimalist Robotics* [Jones and Matarić, 2003, Wilson et al., 2004]. They use a *minimal* design for both the robots and the control algorithms, as in SR. They also stress the importance of the interactions among robots and with the environment. Researchers are still debating about the differences between the two fields. The main difference seems to be that the minimalist ap-

[5]There is still a lot of debate on how to define and classify the different forms of communication. We use the term "indirect" communication, which is unfortunately all but well settled and defined. By "indirect" communication we mean that there is nothing in the robots' program that an observer can interpret as an act of transmitting information to other robots. In the experiments of Chap. 4 and following, robots use neither wireless or infrared communication nor signalling. Nevertheless, they do communicate by modifying the environment—for instance, by retrieving a prey. The information is hidden in this modification, and thus not "directly" available.

proach does not stress the biological inspiration. In the following, we present some of the most representative works coming from both fields without making any distinction. The issues addressed by both fields are important in the context of this thesis.

Kube and Zhang [1996] describe a box pushing system in which a number of robots move boxes to a goal location. Kube and Bonabeau [2000] further explore the same task, and its relationship to the co-operative transport of prey performed by ants. When cooperatively transporting prey, ants show realigning and repositioning behaviours. These same behaviours were adapted for use with robots, to overcome the stagnation that can result from several robots pushing the object in different directions. The system relies on the robots being phototropic, and hence attracted to the brightly lit goal. The approach of these studies differs from earlier approaches to box pushing that used a combination of centralised planning, conflict resolution and explicit communication between robots to coordinate their actions (e.g., Mataric et al. [1995]).

Holland and Melhuish [1999] used groups from six to ten physical robots in their experiments, which consisted in clustering pucks of two different colours. The robots were programmed with a fixed set of behaviours, they used no memory and no communication. They had no means of self-localisation either. They wandered in the arena avoiding obstacles, and they picked up pucks when they encountered them. They dropped their load if they encountered another puck. The authors demonstrated that the robots were able to cluster the pucks in groups. Then, they augmented the robots' behaviours with a "pull back" rule that was differentially applied, depending on the colour of the puck the robot was carrying. The pull back rule required the robot to pull pucks of one colour back for some distance before releasing them. The effect of its application was that the pucks were sorted such that those to which the pull-back rule applied formed an annular ring around a central cluster of the other frisbees. The result was that frisbees scattered across the arena were collected up by the robots, and sorted into clusters of different colours, even though the robots were only following simple local rules. Holland and Melhuish address the problem of over-crowding by measuring the number of collisions between robots because they are "responsible for the large deterioration in performance when the number of robots [is] increased beyond a small limit" [Holland and Melhuish, 1999, p. 181]. They analyse the system by looking at the qualitative and quantitative effects of parameter changes in the control algorithm. The interested reader is referred to the original paper. Their work was recently continued by Wilson et al. [2004]. These works demonstrate that sorting and clustering can be achieved by a group of simple, autonomous and reactive robots that can communicate via the environment.

Krieger and Billeter [2000] show an example of bio inspired division of labour (see Sec. 4.1). They use the *activation-threshold model* [Bonabeau et al., 1996], coming from the biological literature to explain the division of labour in animal societies. According to this model, there are stimuli associated with tasks (e.g., the larval pheromone concentration for the task of feeding) and "individuals start to become engaged in task performance when the level of the task-associated stimuli exceeds their threshold" [Bonabeau et al., 1997]. In the experiments by Krieger and Billeter, the robots have to collect items and bring them back to the nest. Each item helps to increase the level of "nest energy". Robots search for items when the level of nest energy is below a threshold. The thresholds are assigned at the beginning and do not change during the experiments. The results show that the group of robots can allocate tasks efficiently also in noisy environments.

Melhuish et al. [2001] did experiments in *patch sorting*. It is a form of clustering where more than one type of object is present in the arena. The goal is to have one cluster for each item class in the environment. They show that a simple 4-rule behaviour implemented by a swarm of robots can achieve the task. They observe that the performance of the system decreases with the number of classes of objects to cluster and that the time to complete the task is minimal when four classes are used.

Agassounon et al. [2001] and Agassounon and Martinoli [2002] address the problem of task allocation in a colony. The task is the clustering of small cylindrical pucks. They try three different algorithms. The first one uses two timers to synchronise robot activities. Robots search for a maximum time of T_{search}, after which they rest for T_{rest}. The counter for the search phase is reset when a new puck is found. In this way, workers can estimate the local density of items, and if this is too small they rest, decreasing the total number of workers. The values for T_{search} and T_{rest} are fixed and therefore the algorithm is not robust with respect to changes in the environment. Moreover, environmental conditions must be known *a priori* to use optimal values for T_{search} and T_{rest}. The authors then developed an auto-calibrating system. At the beginning, each robot measures the mean time it takes to find a puck, and then adapts its T_{search} using this statistic. A third improvement is achieved allowing the robots to communicate their estimations to neighbours, which can use this information to improve the value of T_{search}.

Ijspeert et al. [2001] showed how robots can collaborate in order to pull a stick out of a hole without using communication. The behaviour of the robots is characterised by a *gripping time parameter* (GTP), which sets the time to wait for help from other robots. Li et al. [2002, 2004] proposed an adaptation algorithm to adjust the GTP in order to improve the collaboration rate. They tested it only in simulation and in an extended version of the problem where k robots are needed. This algorithm is discussed in more detail in Chap. 5.

In Jones and Matarić [2002, 2003], robots have to collect items of different colours and in a predefined sequence. Two control algorithms were implemented. The first one is based on timers connected to each item colour. If too much time has passed without any item of one colour found, robots focus on the next colour. The second algorithm is probability based. Each robot has associated probabilities to ignore items of a given class or to drop them before reaching the home region.

As the reader has probably already noted, what these examples have in common is the simplicity of the control algorithms with respect to the complexity of the task. The drawback of the simplicity is often the performance of the group. Let us take for example the patch sorting problem. A group of robots with perfect knowledge of the environment, of the number and the types of items and of the activity of the other robots can achieve the result in optimal time.

Chapter 3

Experimental Tools

The work presented in the next chapters was carried out using several tools. We describe them in this chapter, instead of talking about them wherever they were used. The information that follows is more technical than scientific. If the reader wants to get informed immediately on the scientific contributions of our work, she/he can skip to the next chapters. We think however that it is more complete if we also describe our work tools, because we invested a considerable amount of time in their development.

We are not going to give many details. Most of the tools were already described somewhere else. We only report the features that might be interesting and useful to better understand the next chapters. There is only one exception. Section 3.3 describes a new hybrid simulator, called BARAKA, that we developed for the experiments of Chap. 6. To the best of our knowledge, there is not yet a simulator like ours, and thus we provide all the details that are required to understand its features.

Section 3.1 describes the real robots used in this thesis for experiments, the *MindS-bots*. Section 3.2 introduces the concept of rigid-body simulations. It is used by the software libraries that were employed to develop the robot simulator of Section 3.2.1, 3.2.2 and 3.3.

3.1 The *MindS-bots*

The *MindS-bot* (Fig. 3.1 and 3.2) are real robots, created using Lego Mindstorms^{TM}. The advantage of Lego is that it allows a faster prototyping phase because it is composed of modular pieces easy to connect to each other. The disadvantage is the mechanical reliability, given the type of plastic used by Lego. The robots need frequent maintenance because of the elasticity of and the connections between some of its components.

The processing unit is an Hitachi H8300-HMS 1 MHz microprocessor with 32Kb RAM and an on-board ROM. The processor is embedded in the central top block of the *MindS-bots*. The block has also three inputs for the sensors and three outputs for the motors (used for the two

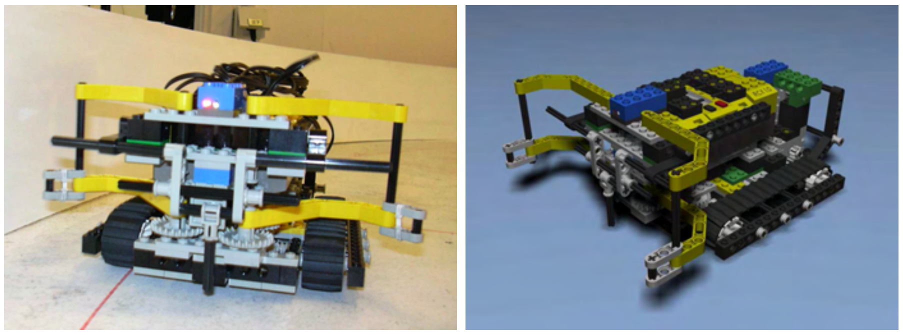
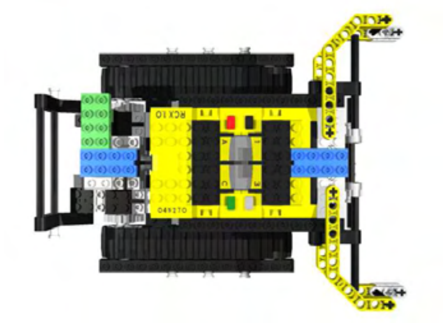
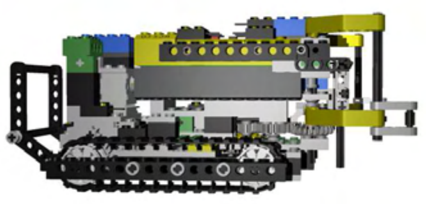


Figure 3.1: A picture of a real *MindS-bot* (left) and a computer generated view (right). See the text for more details.



(a) Top view of an *MindS-bot*. The blue bricks are light sensors. The gripper arms are half opened to have an idea of the grasping area. The front bumper takes less space on the horizontal axis than the back one in order to permit an easier grasping of prey.



(b) Side view of an *MindS-bot*. Tracks do not occupy vertical space that can be used by the gripper when it is open.

Figure 3.2: Computer generated views of an *MindS-bot*.

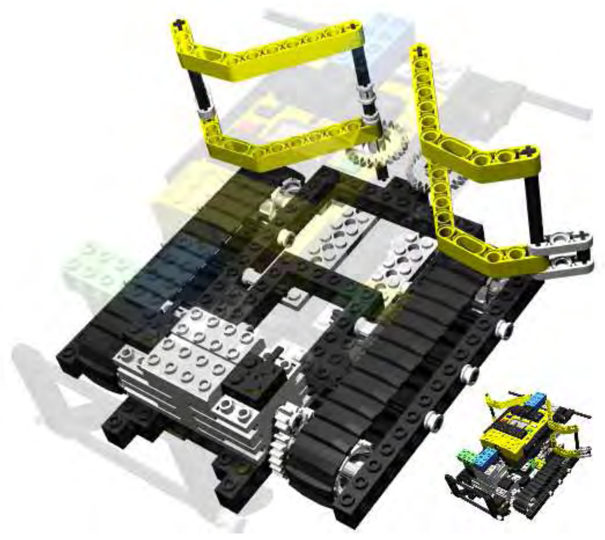


Figure 3.3: Details of the traction system. The gripper's arms can be placed over the tracks when the gripper is open. A small reduction in the transmission between the motors and the tracks allows for a stronger traction at the cost of a slower speed. The parts of the robots that are not of interest are transparent. The small picture on the bottom right corner shows the perspective of the *MindS-bot*.

tracks and for the gripper). The front part of the *MindS-bots* features also an Infra-Red receiver and transmitter. The latter is used to download the program on the robots and to upload the data collected during the experiments. It could also used for robot-to-robot communication, but this feature is not used in this thesis.

When we built the robot, we had to face some mechanical issues that were important for the experiments in prey retrieval of Chap. 4 and 5. For instance, the robot should be strong enough to pull a prey, and it should have a strong gripper not to lose it. The following subsections detail more the *MindS-bot*'s components.

3.1.1 Traction

The *MindS-bot* uses tracks to move on the ground (Figure 3.3). They offer a good compromise between traction power and space compactness. Tracks allow a good traction because of a high friction with the ground. Moreover, the gripper's arms can be placed over them when the gripper is open, as can be noticed in Figure 3.1 and 3.3. Low arms allow to have a more stable and resistant grasp, but it is important that arms do not protrude too much from the *MindS-bot*'s shape in order to reduce the danger of getting stuck into other objects.

A reduction between the motors' and the tracks' gears increases the traction power, although it slows down the *MindS-bot*. However,

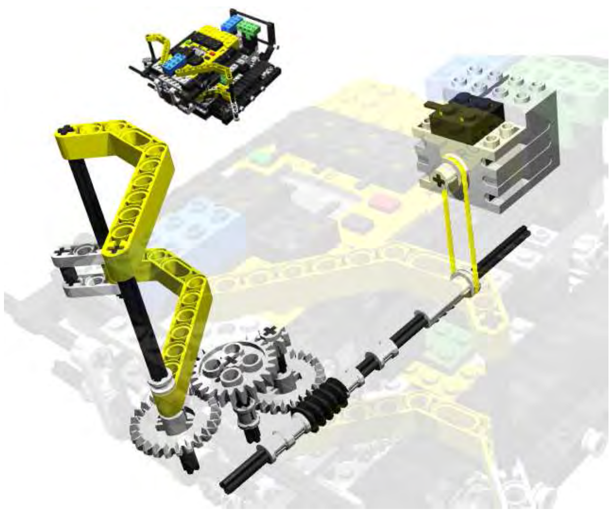


Figure 3.4: Details of the gripper. The series of gears create a reduction which is strong enough to block the gripper. The parts of the robots that are not of interest are transparent. The small picture on the top left corner shows the perspective of the *MindS-bot*.

its speed is still about 12 cm/sec, which is considered enough for the experiments.

3.1.2 Gripper

Two arms, placed symmetrically with respect to the centre of the robot and actuated by the same motor, form the gripper (Figure 3.4). The movement is controlled by a motor, whose pulley is connected by an elastic ring to an axle at the bottom of the *MindS-bot*. The rotation of the motor is transmitted to a worm screw placed on the other side of the axle. The worm screw is connected to the two arms by means of a series of gears. The centres of rotation of the arms result to be nearly at the front end of the robot, increasing the area in which a prey can be grasped.

The series of reductions, especially with the endless screw, allows the gripper to be strong and fix in a position. However, the arms are not completely blocked because of the mobility between gears. Having more gears in a series amplifies this phenomenon, which results that a *MindS-bot* might lose a prey while pulling it.

The rubber connecting the two pulleys is also used to prevent damages to the motor. If the gripper is stuck and cannot move, the rubber will slide over the pulley, allowing the motor to rotate freely.

3.1.3 Sensors

MindS-bots perceive the environment by means of two light sensors and two bumpers. The front light sensor (the blue brick on the top-right of

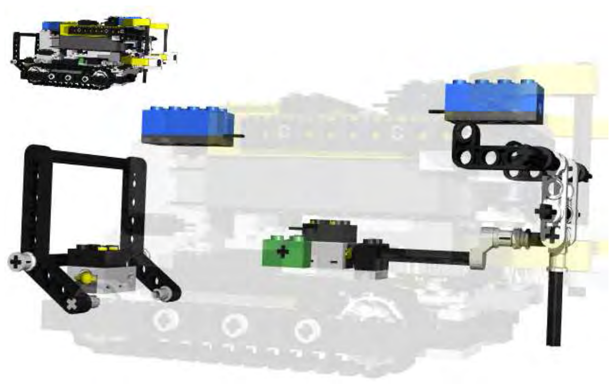


Figure 3.5: Details of the sensors. Light sensors are on the top to reduce problems with the shadows of other *MindS-bots*. The back touch sensor is pressed when the back lever is pushed against the *MindS-bot*. When the front whisker is pressed, an axle connected to it pushes the button of the touch sensor that is placed under the robot. The parts of the robots that are not of interest are transparent. The small picture on the top left corner shows the perspective of the *MindS-bot*.

Figure 3.5) is used to sense the prey. The back light sensor (top left of the same picture) is used to search for the nest, identified by a light source in the experiments of Chap. 4 and 5. The front and back touch sensors are activated respectively by the movement of the back lever and by a pressure on the front whisker.

We recall that the processing unit of the *MindS-bot* has only input for three sensors. In order to use all four sensors, two of them have to share the same input. We connected the back bumper and the front light sensor to the same input. The back bumper is only useful when the *MindS-bot* moves backward. We see later (Sec. 4.5) that the *MindS-bot* move backwards only when it comes back to the nest or when it is retrieving a prey. In these cases, the robot is not interested in prey anymore and it does not need to use the front light sensor.

3.2 Rigid-body Simulation

Rigid-Body simulation is a simulation framework that has become more and more popular in robotics, but also in other fields of computer science, like computer games. It basically consists in simulating the Newtonian physics of body movements. The increase popularity is due to the realistic simulations that can be achieved, but also from the increasing computational power of nowadays computer.

There are several software libraries that simulate rigid bodies. We used Vortex$^{\text{TM}}$ [1] and Open Dynamics Engine (ODE)[2] in our work. Vor-

[1]http://www.cm-labs.com/products/vortex/
[2]http://www.ode.org

tex$^{\mathrm{TM}}$ is a commercial product, ODE is open source. They work however approximately in the same way.

The libraries provide primitives to define a body by its mass, momentum of inertia, initial position and velocity. Different bodies can be attached to each other through a number of joints: free, extensible, hinges, ball&sockets, and so on. Each body can also have more geometries attached to it, which are used to give a shape to the body. The libraries offer also primitives to apply forces and torques to the body (as a motor does on the wheel of the car).

Additionally, Vortex$^{\mathrm{TM}}$ and ODE offers two very important functions. The first one takes the state of the environment at time t and computes the new state at time $t + \Delta$, where Δ is a user defined parameter. This function integrates the equation of motion and returns the solution at time $t + \Delta$. The second function checks whether any two objects are colliding. If it is the case the function introduces new forces between the objects in order to avoid the penetration of the bodies. The latter function is also used to compute friction at the contact points. In this way, if we apply a torque to the hinges that connect four spheres to a parallelepiped, and the spheres touch the ground, we can simulate the wheels of a car on a road.

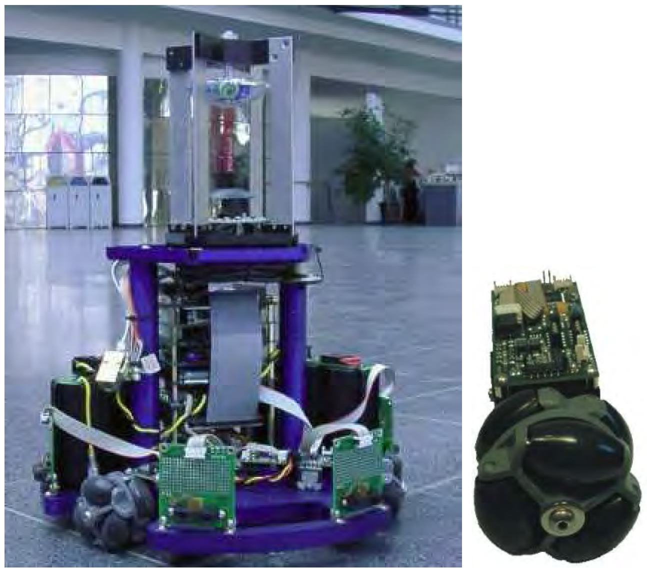
It is up to the user to set up the objects correctly and to iteratively call the two functions to advance the simulation. Between two calls to the integration step, the program can perform whatever task it needs to do. It can change, for instance, the torque applied to some robots' wheel in order to avoid an obstacle.

An example can clarify these concepts. In Chap. 6 we need to simulate Robertino robots.[3] Robertino is a three-wheeled omnidirectional robot, with six infrared sensors around the body and an omnidirectional camera. Figure 3.6 shows the robot and the detail of one of its wheels. It is a Swedish wheel, which allows the robot to have a strong grip in the direction of the rotation of the motor, but a nearly null friction along the axis of the motor. Three such wheels grant the robot the capability to reach every configuration (position plus rotation) on a plane.

Figure 3.7 illustrates how we simulate the Robertino robot with ODE. Wheels are simulated with three spheres. The spheres are connected to the main body through three hinge joints. The main body of the robot is placed in the centre of mass. The robot is approximated by two cylinders (two geometries) linked to the main body. Weight of the body and dimensions of the geometries are the same as those of the real Robertino. The hinges are free to turn around the radial axes. The motors of the robot are simulated by applying torques to the spheres along the rotational axes.

ODE allows to specify two friction coefficients for each geometry. We set very low friction between wheels and ground in the direction of the radial axes and high friction in the perpendicular direction, that is, the direction in which the wheel turns. This can effectively simulate a Swedish wheel.

[3] `http://www.openrobertino.org/`



(a) Robertino

(b) Close-up on one of the three Swedish wheels

Figure 3.6: Picture of a Robertino robot.

3.2.1 MindS-miss: a *MindS-bot* Simulator

MindS-miss was developed to simulate the *MindS-bots*. It is based on ODE and simulates a dynamical model of the *MindS-bots*, with its mass, maximal velocity, inertia momentum, and so on. The model is not a perfect replica of the *MindS-bots* because it would make the simulation too slow. Only the most important feature are simulated, as shown in Fig. 3.8. Each robot was modelled with a limited number of bodies and geometries: main body, front and back bumpers and six wheels (a replacement of the tracks, three at each side of the main body). When the robot's controller decides to move, MindS-miss applies a torque to the six wheels to let them move.

The simulation of the gripper is somewhat simplified because it is obtained creating a temporary joint between the robot main body and the grasped object. The *MindS-bots* can not grip immediately, because of the time it takes to move the arms. MindS-miss simulates this by introducing a delay between the command to close the gripper and the creation of the joint.

The simulator was developed following the indications of Jakobi et al. [1995]. The authors discuss in this paper how to create reliable simulator. A simulator is considered reliable if a controller developed only in simulation can be used as it is also on the real robot. This includes reliable simulation of the sensors and of the actuators.

We used the sampling technique to obtain a reliable model of the two light sensors. We sampled sensor readings in the real environ-

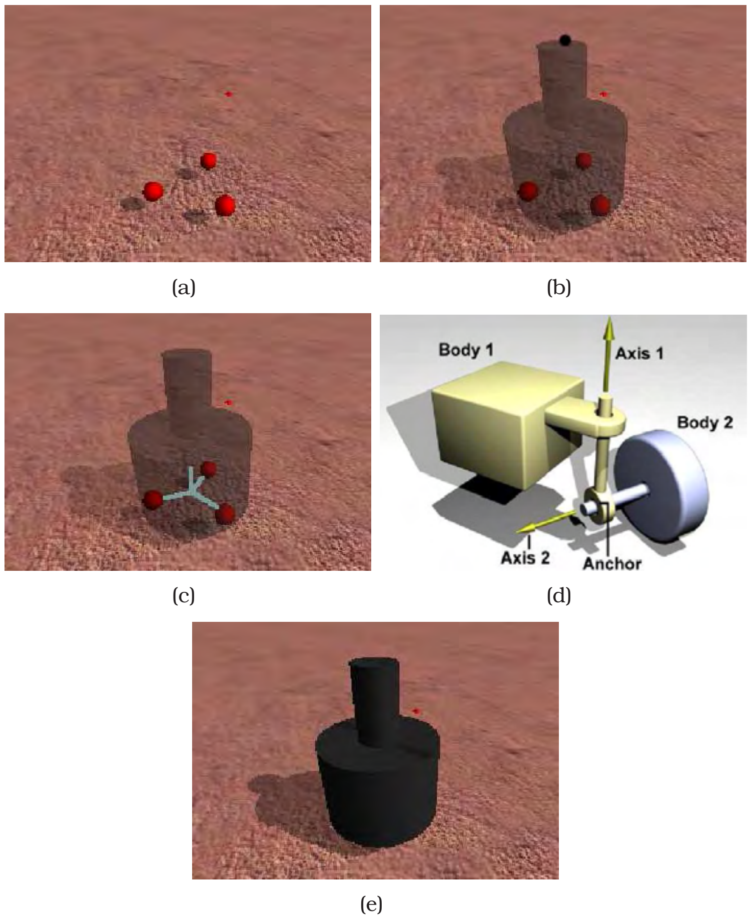


Figure 3.7: This sequence shows how the Robertino robots (Fig. 3.6) is built and simulated in ODE. (a) We take three spheres which simulate the wheels. (b) We define a body with the same weight of the real robot, and attach two cylinders to it. The dimension of the cylinders follows that of the real robot. (c) The wheels are connected to the main body through three hinges like the one depicted in (d). The first axis of each hinge is blocked, so that the second axis, the one around which the sphere rotates, points toward the centre of the robot. Two different friction coefficients are defined for each wheel: a small one for the direction along the line that connects the centre of the robot to the wheel, and a big one for the perpendicular direction. In this way we can efficiently simulate the Swedish wheels of the robot. The resulting simulated object is shown in (e).

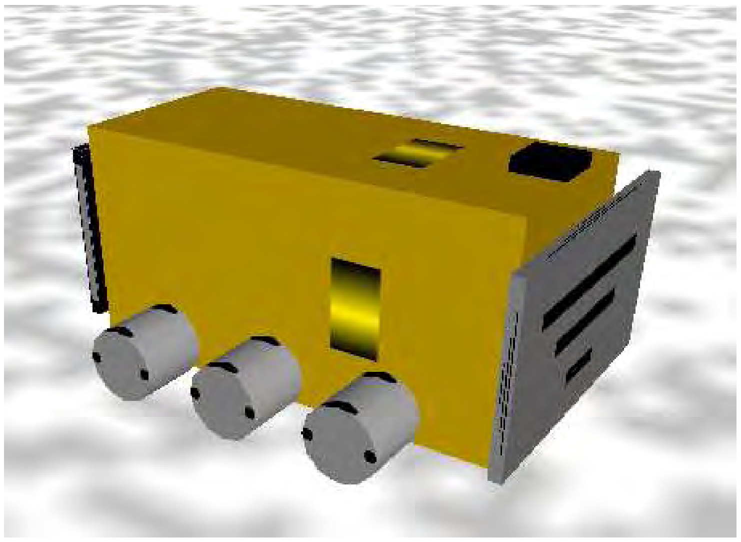


Figure 3.8: A simulated *MindS-bot* is a simplified but physically accurate version of the original one. Six cylinders are used to simulate the tracks. The gripper is simulated by a sticky plate on the front, which works also as bumper. When the plate touches a prey, MindS-miss simulates the gripper by dynamically creating a joint between prey and robot.

ment for a robot in front of an object. The samples were collected for different distances and orientations. We compiled a look-up table with the data. When the controller requires a sensor reading, MindS-miss finds in the respective look-up tables the readings to pass to the specific controller. If the information is missing for a particular orientation/distance pair, MindS-miss interpolates the data from the nearest data points. A random noise is then added to the value. Noise is also added to the wheel speed requested by the controller before applying the torque to the wheels.

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 in order to obtain the same physical behaviour of the *MindS-bots*.

The resulting simulated environment is depicted in Fig. 3.9. Visual rendering was not used during the experiments. A render was however available to visually control the behaviour of the robots. The render is based on OpenGL[4] libraries.

More details about the implementation of MindS-miss can be found in Cirillo [2005].

[4]`http://www.opengl.org/`

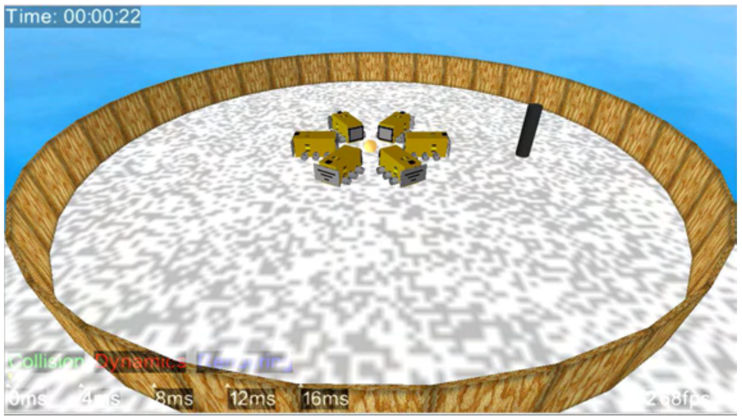


Figure 3.9: A group of six simulated *MindS-bots* in their starting positions, at the beginning of an experiment. The black cylinder is a prey.

3.2.2 swarmbot3d: an *S-bot* Simulator

swarmbot3d was developed by the members of the SWARM-BOTS project.[5] The project aimed at studying new approaches to the design and implementation of self-organising and self-assembling artifacts, called *s-bots*. The *s-bots* (Fig. 3.10) are small autonomous robots, capable of moving in and sensing the environment, and with some small capabilities of changing it. They can however join together and form a bigger structure, called *swarm-bot* [Mondada et al., 2004]. In such formation they can overcome limitations of their structure and achieve task that are beyond the capability of the single robots.

An *s-bot* has fifteen infrared sensors around its main circular body, eight light sensors and eight triplet of red-blue-green LEDs. There are four additional infrared sensors on the bottom, between the tracks. The cylinder on the top holds the mirror for an omnidirectional camera, and near to it there are two speakers and four microphones. Finally, it also has two accelerometer to know its inclination and two temperature/humidity sensors.

Two motors move the tracks independently, and another let the circular body rotate around the vertical axis. One gripper on the front allows connection with other *s-bots*. The gripper is strong enough to lift another *s-bot*. Another gripper on the side, fixed on an actuated arm, allows for looser connections with the others. The microprocessor is an XSCALE, with 64MB RAM, running Linux. Wireless communication is used to transfer the program and data to and from the robots.

swarmbot3d is based on Vortex. It allows the user to choose between different models to be simulated, ranging from a very simple one to a highly detailed one (Fig. 3.11), which is nearly a perfect replica of the real *s-bot*. Obviously, there is a trade-off between the detail level and the simulation speed. When simulating many *s-bots* using the detailed model, the simulation can be slower than reality.

[5] `http://www.swarm-bots.org`

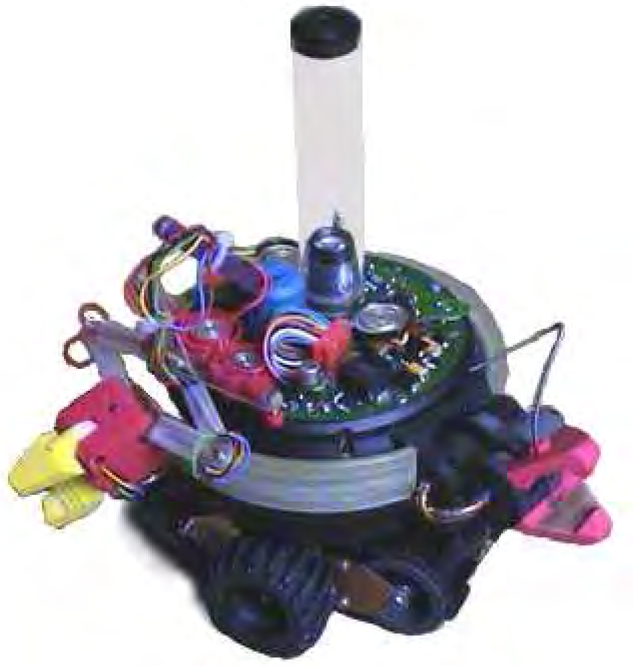


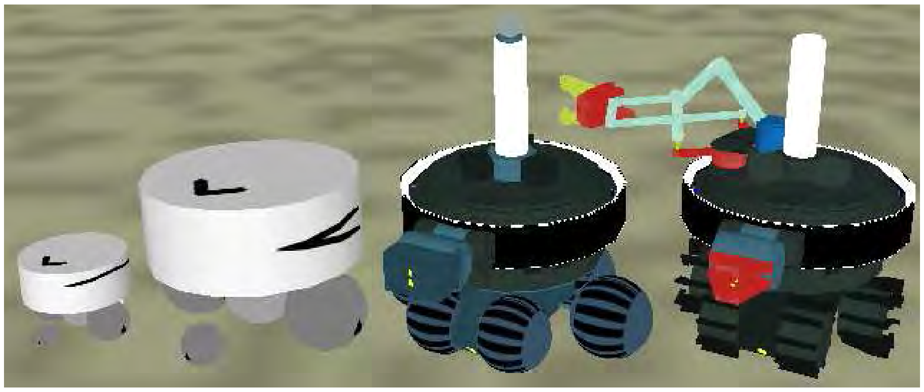
Figure 3.10: Picture of an *s-bot*.

The design of swarmbot3d followed the same criteria as MindS-miss to obtain a reliable simulation.

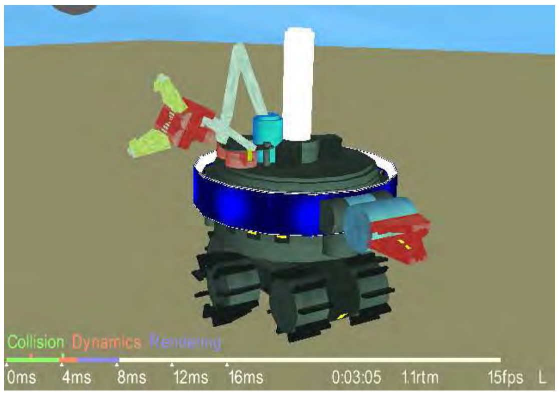
3.3 BARAKA: a SANET Simulator

When we were looking for a Sensor/Actuator Network simulator to use for the experiments in Chap. 6, we found ourselves in front of a serious problem: there is no simulator for SANETs. Or better, there is no simulator that takes equally care of both the networking of the nodes and the realistic movements of the robots. The simulators described so far, for instance, are realistic for what concerns the physics and the movements of the robots in the environments. If we wanted to simulate also wireless communication between them, this would be no easy task. The simulators lack an efficient way of simulating the physical layer, the medium access layer, and the networking layer. These are needed if we want to implement new algorithms that work on the networking of the robots.

The networking community already offers good tools to simulate the protocol behaviour needed for inter-nodes communication, but the environment is often grossly modelled. We decided to take one of these network simulators and extend it in order to accurately simulate also the movements of the robots using ODE. The resulting simulator, called *BARAKA*, is, to the best of our knowledge and at the moment of writing, the only simulator that accurately simulates both the net-



(a) Picture of all the models that can be simulated with swarmbot3d.



(b) Close-up on the detailed simulated model.

Figure 3.11: Models of simulated *s-bots*.

working and interactions with the environment. For this reason, we are going to detail it more than the simulators above.

The following section describes the features of the network simulator from which we started our work. Then, we describe how we joined the network simulation with ODE.

3.3.1 OMNeT++

From `http://www.omnetpp.org/`:

> OMNeT++ is a public-source, component-based, modular and open-architecture simulation environment with strong GUI support and an embeddable simulation kernel. Its primary application area is the simulation of communication networks and because of its generic and flexible architecture, it has been successfully used in other areas like the simulation of IT systems, queueing networks, hardware architectures and business processes as well. OMNeT++ is rapidly becoming a popular simulation platform in the scientific community as well as in industrial settings.

OMNeT++ is a discrete event simulator. It simulates modules that can send messages to each other if they are connected through channels (Fig. 3.12). Modules, channels and messages are implemented as C++ objects. Each message represents an event and is stored in the scheduler of OMNeT++. The simulator, after having initialised the modules, takes the first event in the list and delivers it to its destination. The delivery occurs by calling a method of the module and giving the message as parameter. The module processes the event, then the control returns to the simulation kernel, which takes the following event in the list. Modules can send messages to others (if they are connected through a channel) or to themselves (in this case they mostly simulate internal timers). A delivery time is associated to each message and determines its position in the scheduler list.

An example helps to better understand. Let us take the module *comp[0]* in Fig. 3.12, which simulates a computer connected in a Token Ring topology with the other computers. Let us suppose that the computer sends a message to its neighbour (*comp[1]*) every 30 s. What happens is summarised in Fig. 3.13. During the initialisation phase, the module that simulates *comp[0]* schedules a message for itself at time 0 s, that is, at the beginning of the simulation. When the simulation starts, the scheduler of OMNeT++ takes this message out of the list, and delivers it to the module *comp[0]*. The module is waken up by this message and prepares a message to be sent to *comp[1]*. Then *comp[0]* sends the message, that is, calls a function of OMNeT++ to store the message in the schedule list. Given the length of the message, the speed of the connection between the computers and the current status of the channel (busy or free), OMNeT++ calculates the delivery time of the message to *comp[1]*. After having sent the message, *comp[0]* sends another message, but this time to itself and specifying that the arrival

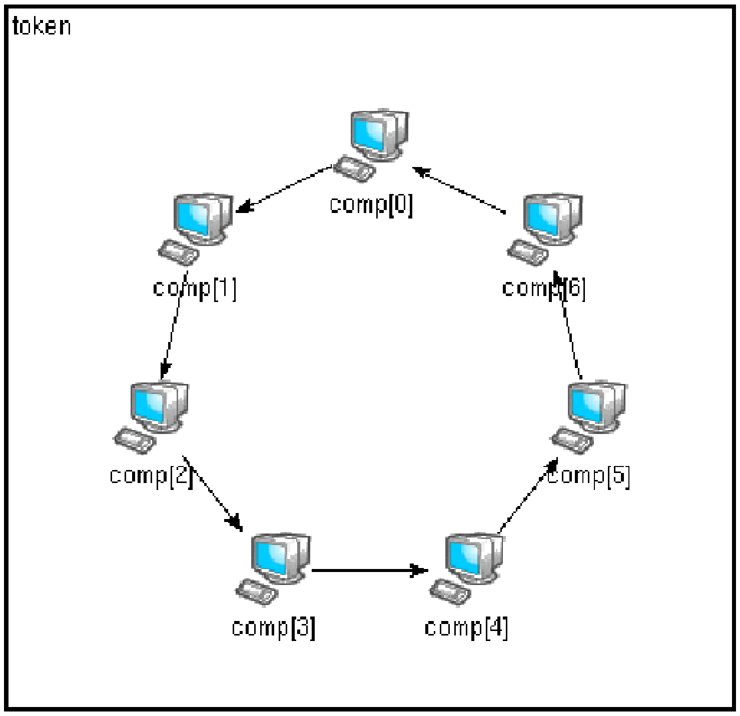


Figure 3.12: Examples of modules and channels in OMNeT++. Modules can receive and send messages through connections.

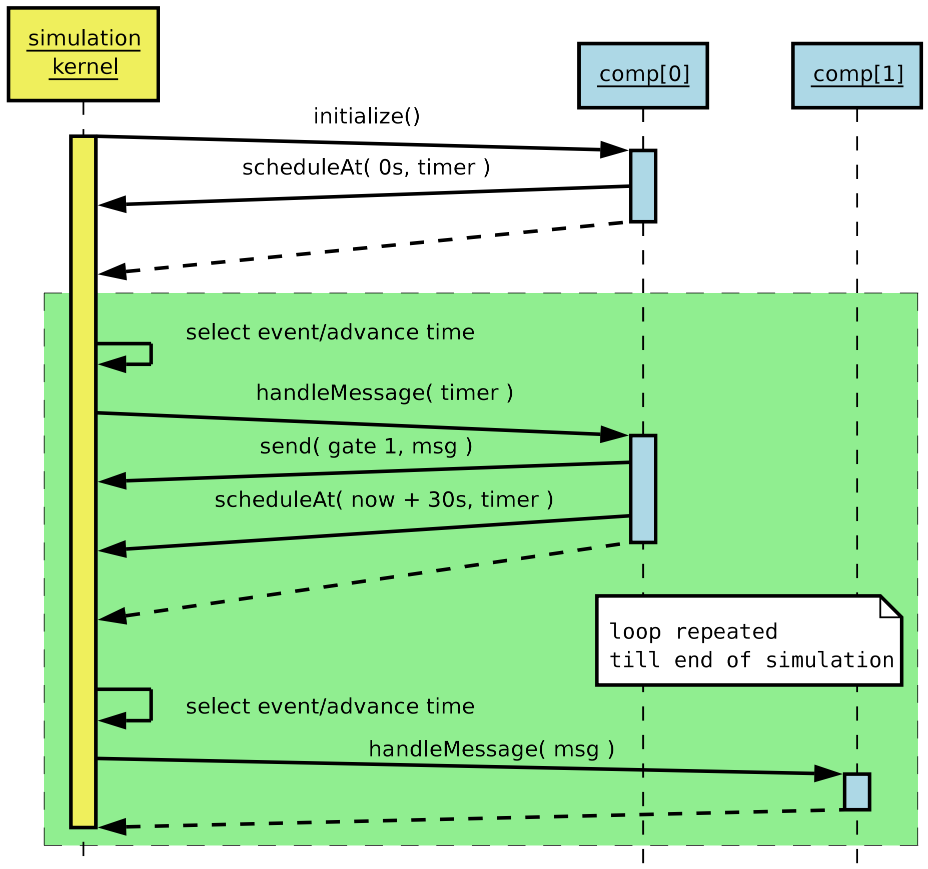


Figure 3.13: UML sequence diagram of the OMNeT++ simulation kernel. The left bar represents the simulation kernel, the other two the objects that simulate the computers in the Token Ring network of Fig. 3.12. The continuous-line arrows from one bar to another represent normal C++ methods call. The parameters of the calls are those between brackets. For instance, `scheduleAt( 0s, timer )` schedules the message `timer` to be delivered to the calling module at time 0 s. `send( gate 1, msg )` send the message `msg` through the channel connected to the first gate of the module. The simulation kernel checks who is connected to the other end of the channel and delivers the message accordingly. Dashed lines show when a module returns the control flow to the caller.

time should be 30 s later. When the module has finished, OMNeT++ processes the next message in the list.

The simulation continues till there are no more messages to be delivered or till a specified time limit has been reached. The current simulated time is the time of the message that is being processed, thus it might advance with big steps if nothing is supposed to happen in the meanwhile, or it might not advance at all, if the simulator is processing two concurrent events. Here comes the definition of "discrete event simulator".

OMNeT++ has several other nice features, like a scripting language and a very useful GUI, but this is not the place to discuss about them. We want only to mention two facts, because we need them later. First, a module in OMNeT++ can be a compound module, made of several sub-modules. Each sub-module can communicate with the sub-modules of the same module, or can be connected with the parent module. A message sent from a sub-module to the parent module is in fact sent to the outside, and can thus reach other modules in the simulation. Secondly, modules can communicate also using the standard C++ way, that is, by direct calls of each other methods (this last way is in fact more efficient that using messaging).

Networks are probably the best systems to be simulated in this way, but OMNeT++ can be used to simulate other systems as well. It is in fact more a simulation framework than an actual simulators. Many other details must be implemented in order to simulate, for instance, mobile networks and SANETs.

The networking community has developed a number of additional libraries to be used with OMNeT++. For instance, the "INET Framework" implements all the IP/TCP communication mechanisms. For our work, we used the "Mobility Framework (MF)".[6]

The MF takes care of placing the simulated hosts in the environment. A special module, the *channelcontrol* module, connects hosts that could theoretically communicate with each other (Fig. 3.14). After every movement of any host, the *channelcontrol* updates its connections according to the new position.

Each host is simulated as a compound module. It contains five other modules (Fig. 3.15(a)). Three of them simulate the standard networking layers: application layer, networking layer and physical transmission device (the NIC). There is an additional module to update the position of the host in the OMNeT++ arena (*mobility*), and a blackboard for cross-layer communication. The working of the application and network layer are explained in more detail in Chap. 6 and below. We focus now on the NIC module, which is another compound module (Fig. 3.15(b)).

We see in Fig. 3.15(a) that the NIC module is the one connected to the parent module, that is, it is the one that receives messages from outside. Being it another compound module, the message reaches first the *snrEval* module (Fig. 3.15(b)). The purpose of the *snrEval* module is to evaluate the signal-to-noise ratio of the incoming packet. The *snrEval* receives all the message immediately, as if it was sent with infi-

[6] `http://mobility-fw.sourceforge.net/`

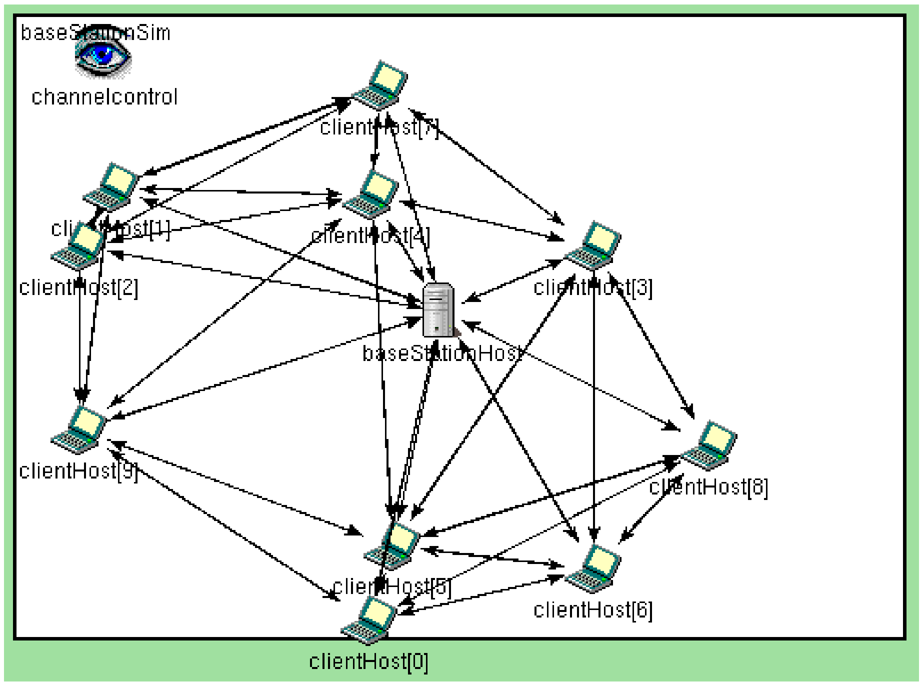
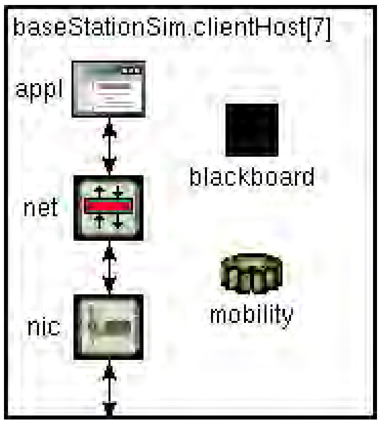
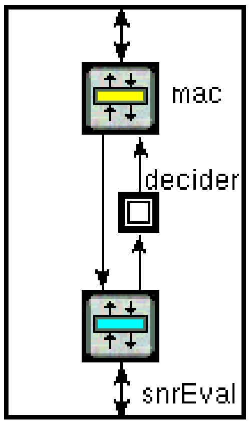


Figure 3.14: Snapshot of a simulation using the Mobility Framework. Several mobile hosts, represented with the icon of a laptop, are spread in the environment. Nodes are connected to those hosts which could theoretically receive their messages. According to the simulated power transmission, their position and the characteristics of the environment, the *channelcontrol* module (top left) takes care of updating the connections between modules.



(a) Architecture of the modules that simulate mobile hosts.



(b) Details of the NIC layer.

Figure 3.15: Implementation of a mobile host in the Mobility Framework. Any mobile host is simulated using five different modules (left). Three of them (on the left of Fig. 3.15(a)) take care of the simulation of the physical transmission device (the NIC), the network layer and the application layer. One module (bottom right) controls the position of the host in the simulated environments. The last one (top right) is a blackboard that is used for cross-layer communication. On the right, we see that the NIC layer is also a compound module, made of a signal-to-noise-ratio evaluator, a decider and the MAC layer. See the text for a description of the functions of all modules.

nite speed. After the arrival of the message, it changes into a "receiving" state. Knowing the length of the message and the transmission speed of the other nodes, the *snrEval* schedules a timer for itself when the real transmission should be concluded. Whatever other message might be received in the meanwhile is considered just noise and decreases the signal-to-noise ratio. The *snrEval* does not discard messages, but just records the level of noise during the transmission. When the transmission is over, the message is pushed up to the *snrDecider* module.

The *snrDecider* receives the message and the signal-to-noise ratios measured by the *snrEval* during the transmission. If the noise was too high the packet might be dropped. The reason for decoupling the receiving process in two parts in motivated by the fact that in the *snrDecider* it is possible to implement some form of error correction mechanisms.

If the *snrDecider* decides that the message is valid, it is pushed up to the *mac* module, which takes care of the medium access control protocol in use. The *mac* is also connected to the NIC compound module, thus the *mac* can receive and send messages to the network layer. In case the *mac* receives a message from the network layer, it is pushed immediately to the *snrEval* (if it is not currently busy with another transmission) for sending. In this case the *snrDecider* is not useful.

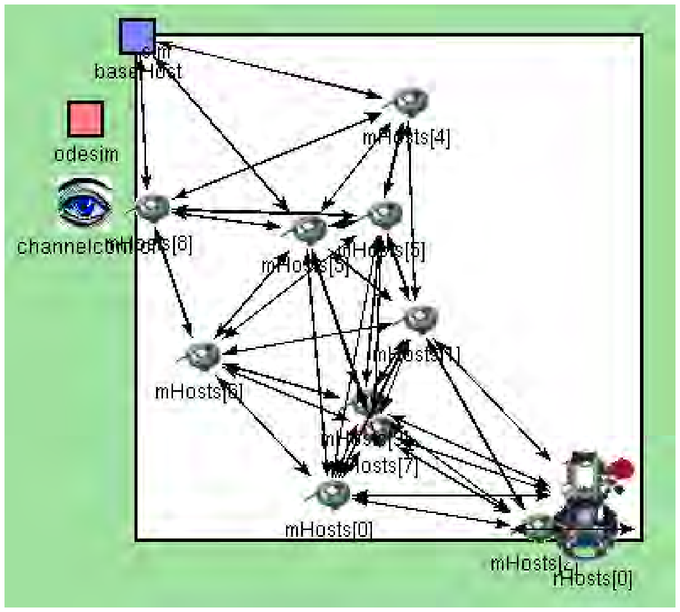
We described the MF to give the user the feeling of how detailed a simulation with OMNeT++ can be, and also to let her/him understand better how discrete event simulators work. For this thesis, we modified only the application and the network layers, the upper ones, and used what was already available in the community for the lower layers. Namely, we used an already implemented simulation of the IEEE 802.11 protocol for wireless communication. Network and application layer are described in Chap. 6.

3.3.2 Integrating ODE into OMNeT++

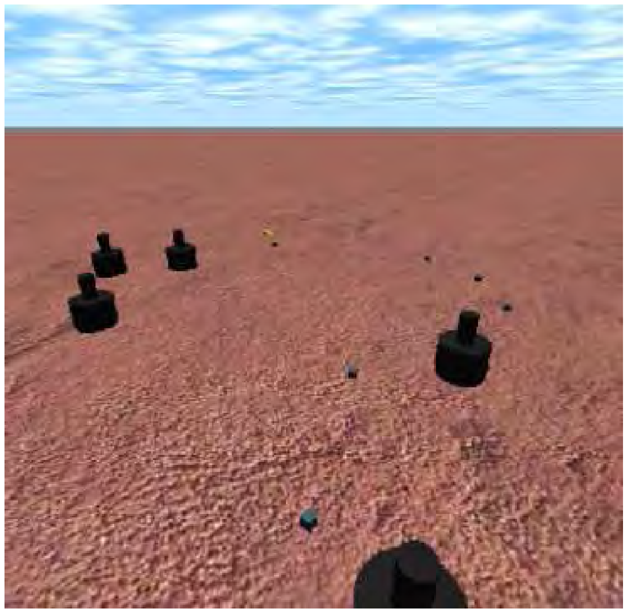
To integrate the two different types of simulation, we decided to port ODE inside OMNeT++. The integration takes two steps: first, implement a module that simulates the physical environment and that takes care of calling the integration and collision detection steps of ODE; second, allow the hosts to interact with this module (e.g., to check whether there are obstacles to be avoided).

We created a new module called *odesim* (Fig. 3.16(a)) belonging to the class `ODESimulator` (Fig. 3.18, described below). It has no connection to any other module in the simulation. This module neither receives messages from nor send messages to the others. It only sends messages to itself regularly every Δ in order to take an integration step and to perform collision detection and resolution (Fig. 3.17).

The application layer module of every other node is in charge of interacting with the physics module. More specifically, we defined two module types, or C++ classes, `SimulatedMote` and `SimulatedRobot`, which are in charge of interacting with OMNeT++ and ODE. The relevant part of the class diagram of BARAKA is shown in Fig. 3.18.



(a) Snapshot of the network simulation



(b) Snapshot of the physical environment

Figure 3.16: Integration of ODE into OMNeT++. The new simulator can handle both the details of network communication and of physical environments.

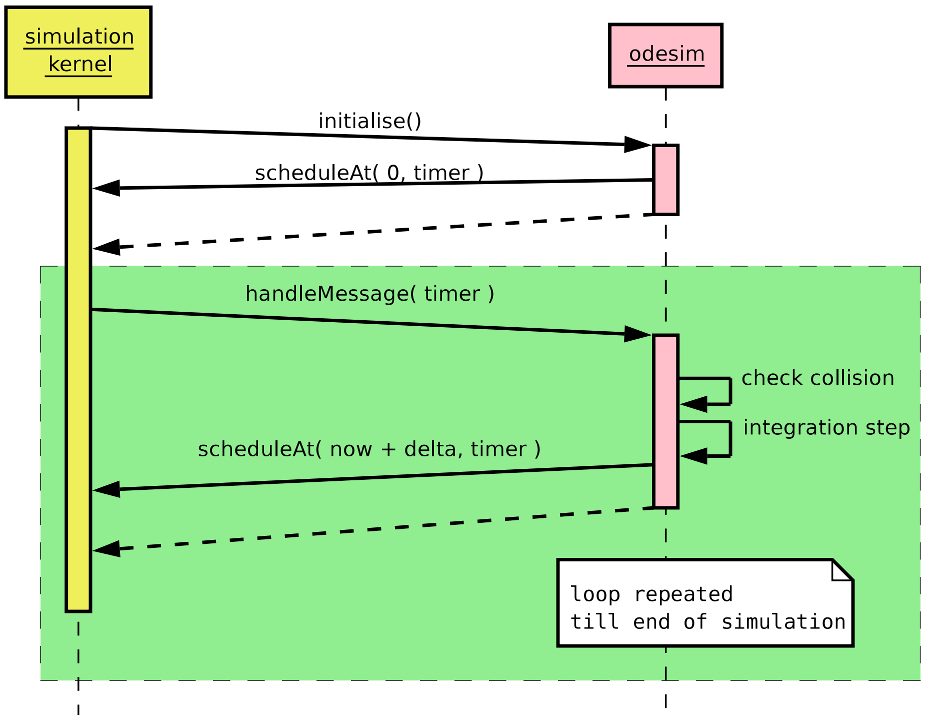


Figure 3.17: UML sequence diagram of the integration between OMNeT++ and ODE. The left bar refers to the simulation kernel of OMNeT++. The right bar refers to the OMNeT++ module which implements the real world simulation with ODE.

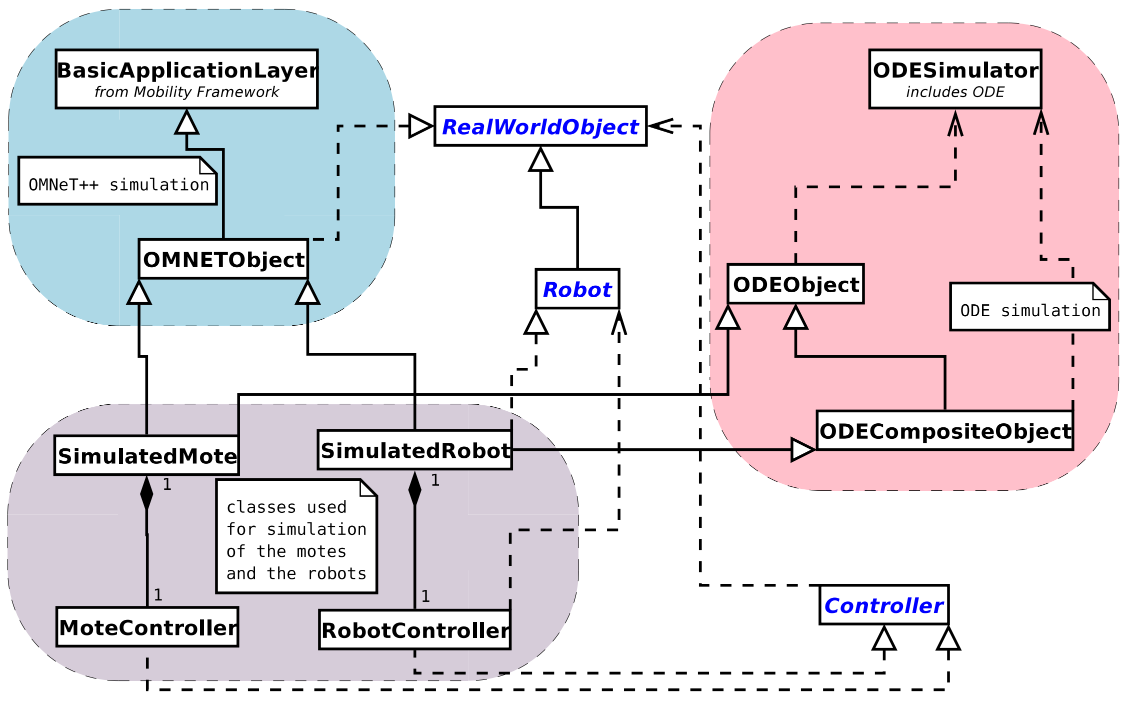


Figure 3.18: UML class diagram of the most relevant classes of BARAKA. Classes with blue and italic names are abstract interfaces. The classes on the top left corner of the pictures deal mostly with the simulation of motes and robots as network nodes. The classes on the top right corner implement motes and robots as physical objects. The classes on the bottom, which derive from the other two groups, are the one actually used for the simulation of motes and robots in BARAKA. See the text for more details.

Two classes implement the controllers of the motes, `MoteController`, and of the robots, `RobotController`. They both derive from the abstract interface class `Controller`. The way in which the controllers can issue commands to motes and robots is defined by two interfaces, `RealWorldObject` and `Robot`. `RealWorldObject` describes the capabilities of both motes and robots (like sending a message, or recording sound). `Robot` describes the additional capabilities of a robot (typically, to move).

The classes `SimulatedMote` and `SimulatedRobot` are an implementation of respectively `RealWorldObject` and `Robot`. They take the commands of the controllers and implement them in the simulated environments. As future work, it will be possible to implements other classes that work on the real hardware. The controllers' code can be then run, as it is, both on the real hardware and on simulation.

`SimulatedMote` and `SimulatedRobot` derives from two group of classes. The first group is represented by the classes `OMNETObject` and `BasicApplicationLayer`. `BasicApplicationLayer` is a class provided by the Mobility Framework. `OMNETObject` derives from it and implements the interface given by `RealWorldObject`. `OMNETObject` basically implements all the capabilities of robots and motes that have to deal with networking, like sending and receiving of messages, or the use of timers.

The second group is formed by the classes `ODEObject` and `ODECompositeObject`. They are general classes used to implement the physical aspects of robots and motes. An object of type `ODEObject` has one body and one geometry. It is mainly used to simulate the motes and other obstacles that might be placed in the environment. `ODEObject` interacts with `ODESimulator`, which simulates the physical world using the ODE library. `ODEObject` can refer to `ODESimulator` to create the body in the simulated world, but also to know the current position and rotation of the object. `ODECompositeObject` is an extension to `ODEObject` that allows the object to have more bodies connected through joints and several geometries. It is used to simulate the robots. `ODECompositeObject` can use `ODESimulator` to obtain the list of nearby objects. This is useful to simulate proximity sensors or the camera of the robots.

`SimulatedMote` and `SimulatedRobot` can simulate both the networking of the objects and their behaviours in the physical world by deriving from both groups of classes. The simulation kernel contains several instances of the classes `SimulatedMote` and `SimulatedRobot`, one for each robot or mote that has to be simulated. We call each instance a "simulated object".

During the initialisation phase of the simulation kernel, the simulated objects can create bodies and geometries in the physical world (Fig. 3.19). This process is visually represented in Fig. 3.7. After that robots and motes have been built, timers are set and used to simulate the control cycles of robots' and motes' controllers.

When the control cycle timers expire, the simulated objects activate their controllers. These might in their turn call methods of the simulated objects in order to, e.g., obtain the infrared sensor values or set

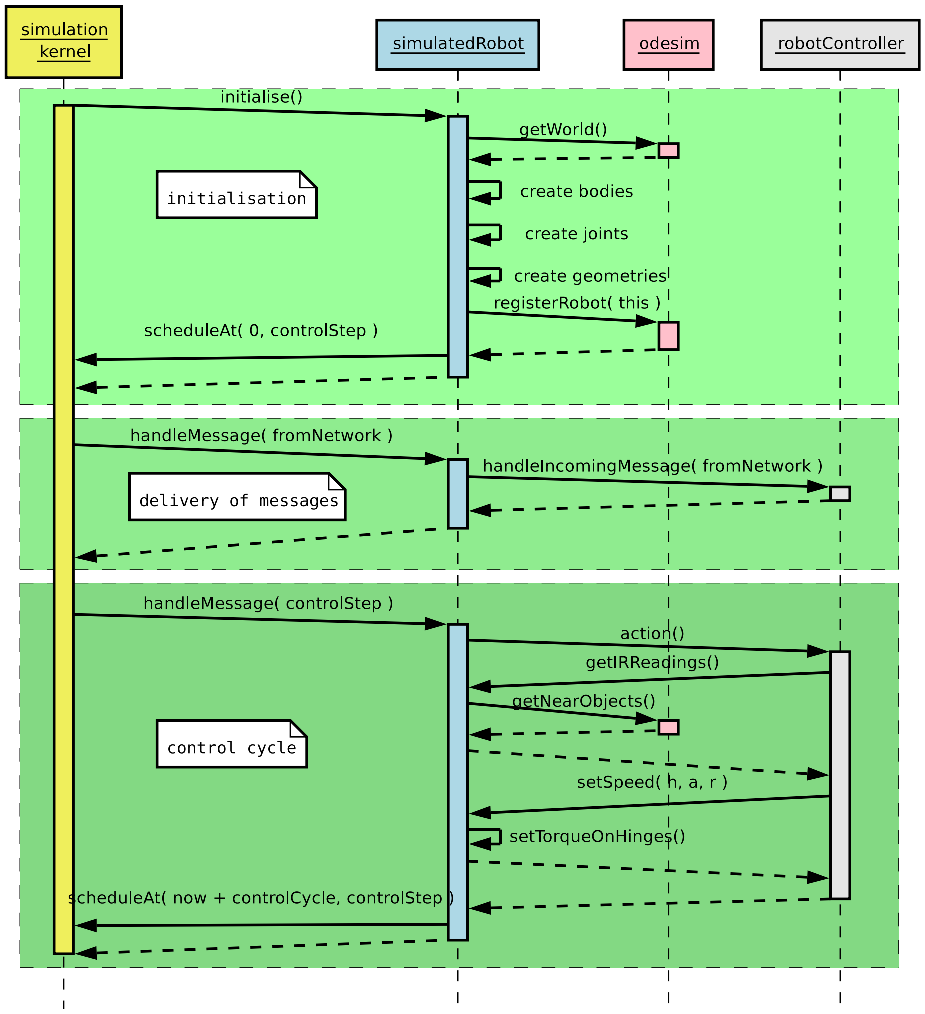


Figure 3.19: UML sequence diagram that shows how OMNeT++ can be used to simulate a physical robot. During initialisation, the class that simulates the robot creates its physical components. To create them with the ODE library, the class needs some parameters. The class gets them via a call to the *odesim* module. When the robot is created, it is registered within *odesim* for the simulation of its physical behaviour. The delivery of messages from other modules by the simulation kernel is implemented by forwarding the incoming message to the robot's controller class. Finally, at every control cycle, implemented via timers, the *simulatedRobot* modules simply activate the robot's controller. This might in its turn call functions of *simulatedRobot* to obtain, e.g., the value of the infrared sensors or to set the direction and speed of motion.

the speed of the wheels. The simulated objects might need to interact with *odesim* to complete the required operation.

The network connections to other agents are kept up to date by the mobility module of each node. It regularly queries *odesim* for the position of the agents and then inform *channelControl*.

The physical simulation based on ODE has shown to be reliable. The results obtained by the two other simulators described earlier in this chapter could be successfully replicated also with real robots. OMNeT++ has also shown in the networking community to be a valid simulation tool. We are thus confident that the two tools joint together can be useful for simulation of Sensor/Actuator Networks. A formal validation is however still missing.

Chapter 4

A Bio-inspired Algorithm for Prey Retrieval

This chapter starts the discussion about the central topic of our work: division of labour. Here, we describe an algorithm for division of labour that is inspired by an ants' foraging model. The model is particularly interesting because it emphasises the role played by individual learning and stresses the distributed aspect of foraging, the lack of a governing hierarchy, and the self-organisation of the ants. In few words, it is a good candidate for Swarm Robotics. We test the algorithm in a test problem that is much similar to ants' foraging: prey retrieval. It is also known in the literature as foraging or search&retrieve.

We start describing the problem of prey retrieval in Sec. 4.1. The same section gives also a formal definition of division of labour. Section 4.2 points out different uses of the words "division of labour" and "task allocation" throughout the literature and fixes the meaning used in this thesis. Section 4.3 provides an overview of what is known about ants' foraging. Section 4.4 introduces and describes the model we are going to study. Section 4.5 depicts the robots controller and our version of the algorithm. Sections 4.8 to 4.11 describe the characteristics of our algorithm, as we observed them during the experiments, and its analysis.

4.1 Prey Retrieval

Prey retrieval is easy to describe: a group of robots has to look for objects (i.e., *prey*) spread in the environment and retrieve them to a special area, called *nest*. This problem is often used in the literature, although under different names, such as *search & rescue* or *foraging*. We use the words *prey retrieval* to emphasise the similarity with the corresponding behaviour observed in ants.

Prey retrieval is often used as a model for other real-world applications such as toxic-waste cleanup, search and rescue, demining or collection of terrain samples in unknown environments. It is among the canonical tasks for collective robotics listed by Cao et al. [1997]. The main scientific question is whether there is an actual performance gain in using more than one robot, since the task can be accomplished by a single one [Cao et al., 1997]. Stated in another way, the question is whether more robots are also more efficient.

This problem can be better understood if we look at what happens in animal societies, for instance ants. Ants need to search the environment for food items in order to survive and to provide energy to the colony. For the energetic economy of the nest, prey items are an *income*.

Not all ants can forage: if all ants were foraging, none could defend the nest or feed the brood. Searching also has drawbacks, that can come from dangers in the environment or from predators. It is also possible that foragers interact in an inefficient way, for instance trying to collectively retrieve an item that could be carried by one ant only or obstructing the way one to the other. From the nest-energy point of view, these effects represent *costs* that decrease the net income of energy.

Income and costs depend on the number X of ants that are exploring the environment. We can assume that the retrieval rate of prey is approximately αXP, where P is the total amount of prey in the environment and α is the probability that one ant finds one prey in one unit of time, i.e., the discovery rate.[1] P is generally not constant in time. It can change because new prey appear or because foragers retrieved some of them. To be more general, prey can also disappear (e.g., if it is an insect that is walking through the area). Prey dynamics can be modelled according to the following equation:

$$\frac{dP}{dt} = \phi - \alpha XP - \beta P \ ,$$

where ϕ represents the "appearing" and β the "disappearing" prey rate. This equation tells that the equilibrium is reached for

$$P = \frac{\phi}{\alpha X + \beta} \ ,$$

which is a stable point since $\alpha > 0$, $X > 0$, $\beta \geq 0$ and therefore $-\alpha X - \beta$, the eigenvalue of the system, is less than 0. The energetic income rate of the colony $i(X)$ is proportional to the retrieved-prey rate at the equilibrium:

$$i(X) \propto \alpha X \frac{\phi}{\alpha X + \beta} \ . \tag{4.1}$$

It is harder to find a good approximation for a proper cost function $c(X)$, given that it depends on factors in the environment that are *a priori* unknown. It is still possible to derive some of its characteristics considering what it should represent. Since it must consider the total amount of energy spent for survival by all the individuals of the colony,

[1]The following model of foraging is not intended as a model of the results that we show later. It is meant only to help the reader understand the problem of prey retrieval.

it must be proportional to the total number of colony members, N, which is constant. It must also take into account the energy spent by foragers and the possibility that they get lost or killed (in this case the colony loses resources to exploit). Therefore, the first derivative of $c(X)$, must be greater than 0. Finally, we should also consider the influence of negative interactions between foragers, which increases with the square of their number. Thus the second derivative must also be greater than 0. As a first approximation, $c(X)$ can be given by:

$$c(X) = \delta N + \epsilon X + \gamma X^2 \ , \tag{4.2}$$

with $\delta > 0$, $\epsilon > 0$ and $\gamma > 0$.

Both income and costs increase with X, but not in the same way. The income saturates when X becomes too high (robots can not retrieve more prey than their actual number in the environment), but costs potentially increase without limit. On the other hand, if $X = 0$, the income is null, but the costs are not. In both situations, the colony can not survive. The colony should "operate" between a minimum and a maximum value of X.

The *efficiency* of foraging can be defined as

$$\eta = \frac{i(X)}{c(X)} \ . \tag{4.3}$$

We come back now to robots. From the engineering perspective, an important issue is how to improve the efficiency of the group. We see three possible strategies:

Increase the income: an increase of $i(X)$ increases also η. In a Multi Robot System (MRS), it can be achieved by using better sensors or improving the search strategy of the single robots. This improves the retrieval rates of the robots and therefore the prey income. This strategy implies that the robots have to move less to find a prey, and therefore they consume less energy. This is accounted in the linear term of (4.2).

Decrease the costs: much of the costs come from negative interferences among the robots. Several works in the literature recognise that this is one of the major problems in a MRS [Balch, 1999, Goldberg and Matarić, 1997]. Interferences increase theoretically with the square of the number of robots, and therefore their contribution to the costs can be more relevant.

Use an optimal number of robots: this strategy comes from the observation that if $X = 0$, then $i(0) = 0$, $c(0) = \delta N$, and therefore $\eta = 0$. If $X \to \infty$, then $i(X) \to K$, $c(X) \to \infty$ and therefore $\eta \to 0$. This means that there is an optimal value of X that maximises η, which can be represented as:

$$\hat{X} = \underset{X \in [0,N]}{\operatorname{argmax}} \ \frac{i(X)}{c(X)} \ .$$

The robotics literature offers several examples of how to implement these strategies. The reduction of costs can be obtained, for instance, by using communication to co-ordinate the robots and avoid interferences. Balch and Arkin [1994] studied the effects of different forms of communication in different tasks.

Other solutions implement better behaviours. Goldberg and Matarić [1997] estimate where the interferences occur the most, by counting the number of collisions or manoeuvres to avoid other robots. With this data, they design a control algorithm that avoids the most problematic zones. Schneider-Fontán and Matarić [1996] reduce the interferences by assigning a predefined part of the arena to each robot, and each robot to one particular area.

The optimal number of robots can be estimated *a priori* if the characteristics of the environment are well known and fixed, as done for instance by Hayes [2002]. Otherwise, the group of robots should be provided with some form of adaptation in order to cope with uncertain and dynamic environments.

We refer to the mechanism that tunes the number of robots involved in the retrieval task as *division of labour*.

4.2 Division of Labour and Task Allocation

There are some problems with our definition of division of labour. In the robotics literature the terms “division of labour” and “task allocation” are often used as synonym, although we see some important differences in their use.

“Task allocation” is the term more often used in more traditional robotic studies. These studies include many tasks that can be performed by one or more robots. By “task”, the authors mean a set of operations with a starting time and a duration. The problem is to find the best assignment robots/tasks, once the robots’ qualities are known. This is the meaning used, for instance, by Gerkey and Matarić [2004], who recently proposed a taxonomy of task allocation problems. They analyse some of the known solutions which use *intentional* cooperation [Parker, 1998]. Focusing only on intentional cooperation, they exclude basically all the field of SR. Other examples of task allocation are found in Jin et al. [2003] and Flint et al. [2004]. They study distributed control and task allocation for Unmanned Air Vehicles (UAV). In these examples however, robots have enough information in order to create explicit models of the environments and of their own capabilities, unlike the SR approach.

“Division of labour” is more typical of bio-inspired researches. Here, authors tackle the problem of “how many robot shall perform one task?” [Labella et al., 2006a, Agassounon et al., 2004, Li et al., 2004]. By “task”, these authors mean a set of operations that live as long as the system itself.

Biology gives to “division of labour” a meaning that does not always match the one used in the robotics literature. The works in the robotics literature describe as “division of labour” the fact that some individual

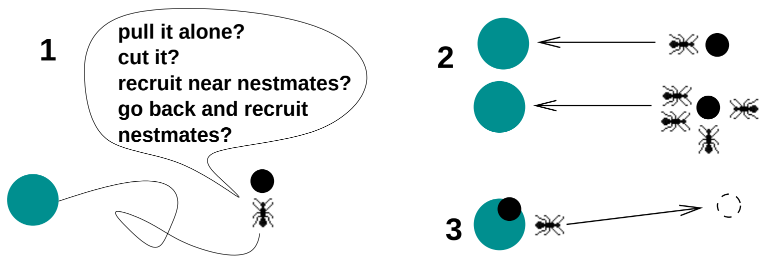


Figure 4.1: Foraging and retrieving behaviour of ants. An ant performs a random walk in the environment until it finds a prey. Then, it takes some decisions about how to retrieve it. Transport can be done alone or in group. Once the prey is in the nest, the ant goes back directly to where the prey was found.

performs a task more and more often (as we do in Sec. 4.9), therefore they reason at the individual level. For biologists, division of labour is a phenomenon that is related to observations of the group, not of the individuals (e.g., measuring the average number of robots foraging in the environment). What roboticists call "division of labour" is for biologist "specialisation". On the other hand, a roboticist might think that "specialisation" implies adaptation to a task, so that an individual "learns" to perform it more efficiently, which is not what biologists mean. It is not our intention here to argue for one definition or the other, but only to let the reader understand which is the one that we use in this thesis.

4.3 Prey Retrieval in Biological Systems

A wide range of different foraging behaviours is observed during prey scavenging and prey retrieval, but the mechanisms governing their emergence remain unclear. Figure 4.1 sketches the general behaviour of ants. When they find a prey after having randomly explored the environment, they take decisions following this schema [Detrain and Deneubourg, 1997, Hölldobler and Wilson, 1990]:

- they first try to pull the prey and if it is too heavy they can recruit local nest-mates by emitting a chemical signal;
- they can decide to return to the nest and recruit nest-mates by laying a pheromone trail;
- they can cut the prey on place with their jaws and retrieve smaller pieces.

The retrieval can be done solitary or in group, if the prey is too big. Once back in the nest, the forager exits and directly returns to the place where the prey was found.

In many species [Cammaerts, 1980, Cammaerts and Cammaerts, 1980, Detrain and Pasteels, 1991, 1994], the colony activity is regulated through the use of chemical trails and recruitment in the nest, which is more intense for large than for small prey. These collective responses (cooperative retrieval, trail recruitment, and so forth) result from decision-making systems which are poorly understood. The main issues are about the behaviours of ants, the criteria they use to estimate the characteristics of the prey and the way they coordinate their movements and decide whether to recruit or not. Some of these problems are solved through cooperation and synchronisation emerging from simple interactions among individuals and between individuals and the prey. Coordination in collective transport seems to occur through the item transported: a movement of one ant engaged in group transport is likely to modify the stimuli perceived by the other group members. This is an example of stigmergy [Sudd, 1963]. The task in progress generates new stimuli to which the insects react by continuing the task and possibly creating new information sources. The prey is at the same time the object transported and the media allowing the coordination of transporters (including the decision to recruit) in order to retrieve it. The movements of the prey contain the essential information used by ants: any movement indicates, without the use of any measure of the prey size and weight, that the pulling force is sufficient. This criterion, although very simple, is of high importance for a colony since it is used to decide whether to recruit and involve more individuals in a task.

The Mediterranean terrestrial species *Pheidole pallidula* and *Œcophylla longinoda* are main models for the understanding of prey retrieval. *Pheidole* (and also *Œcophylla*) helps understanding how dead prey of different sizes induce different global foraging patterns and different levels of cooperation. It also raises several questions about the informative content of the food itself and the modulation of individual behaviour at the food source.

Pheidole pallidula is divided in two castes: minor and major. One minor can carry small items while medium-sized prey or cumbersome body parts are retrieved by groups of cooperating minors. Most scavenged insects individually retrieved show an average weight of 0.86 mg. Larger prey are carried back to the nest by the ants collectively [Detrain, 1990]. A very large food item induces a massive recruitment of both minors and majors which dissect the prey directly at the food site. Food discovery and trail recruitment is done exclusively by the minors, whose poison gland contains trail pheromone. Majors perceive and follow these trails but cannot produce the pheromone [Ali et al., 1988, Detrain and Pasteels, 1991]. The majors caste is only involved in recruitment for heavy prey which they cut into pieces.

In *Œcophylla longinoda*, the collective and individual behaviours are similar to those of *Pheidole*. Retrieval consists of two periods (motionless and high speed retrieval) constituted by a succession of moves and stops. These periods are increasing exponentially with the mass of the prey. The proportion of the motionless period is approximately 70-60%, but decreases as the mass of the prey increases (77% to 57% respectively for 0.4 g and 2.8 g). The population size around the prey follows the same dynamics. However, for *Œcophylla*, when a plateau is reached, a decrease of the population around the prey is observed.

Comparing different masses, the higher the mass, the lower this decrease, which shows the impact of the retrieval speed on the population size around the prey.

It has been noted that the foraging behaviour of a single ant may be influenced by several factors, like age or genetic differences. A few authors pointed out that learning might play a key role. Different forms of learning related to ants have been described:

- food acceptance in relation to food quality [Sudd and Sudd, 1985];
- sectorial fidelity in individual foragers [Hölldobler, 1976, 1980, Wehner et al., 1983]
- route-fidelity to permanent food sources, characterised by a time scale of weeks or months, in species using trunk-trails [Rosengren and Fortelius, 1986].

Among these works, we focused our attention on the learning model proposed by Deneubourg et al. [1987]. The model emphasises the role played by learning during individual lifetime. No direct communication, not even signalling, is necessary for the colony to adapt to the environment. It is appealing and tempting to use it also for a group of robots whenever division of labour is needed.

4.4 A Model of Prey Retrieval in Ants

Deneubourg et al. [1987] proposed a simple learning model to explain the foraging patterns observed in *Neoponera apicalis* (now known as *Pachycondyla apicalis*). This species is characterised by a small colony size, around one hundred adults. The ants are diurnal and forage individually, meaning that no recruitment is observed. They retrieve usually small insects and larvae. One group of ants usually forage next to the nest in overlapping individual zones. Another group forages far from the nest, where a marked spatial fidelity can be noted. This means that these foragers have “personal” zones where they return regularly over a period of one month or more [Fresneau, 1985].

Deneubourg et al. assume that each ant has a probability P_{l} of leaving the nest at each iteration. When it does, it goes with probability Q_{i}, $i \in \{1, 2, \ldots N\}$, to one of the N possible foraging sites. Each site has probability r_{i} of containing a prey. The idea of the model is that the ant increases its P_{l} and Q_{i} if the trip to the site i is successful. The increments are constant, respectively Δ_{P}^{+} and Δ_{Q}^{+}. If the trip is unsuccessful, the ant decreases both P_{l} and Q_{i} using Δ_{P}^{-} and Δ_{Q}^{-}. The part that concerns the adaptation of P_{l} is summarised in Alg. 1.

What the model suggests is that there is a positive feedback that brings the ants to specialise in foraging or in remaining inactive in the nest. The same feedback brings the ants into focusing only in a small set of possible foraging sites. Deneubourg et al. prove that their model is able to explain:

Algorithm 1 The original ants' learning model proposed by Deneubourg et al. [1987] to model ants' learning. Here, only the adaptation of P_l, the probability to leave the nest, is reproduced.

```
$P_l \leftarrow P_{\min}$;

if success then                              if failure then
    $P_l \leftarrow min\{1, P_l + \Delta_P^+\}$          $P_l \leftarrow max\{P_{\min}, P_l - \Delta_P^-\}$
fi                                           fi
```

- the efficient distribution of foragers in the foraging area, as a function of different characteristic of the food distribution;
- the foraging pattern similar to that of the spatial fidelity in ants or the flower constancy in bumblebees;
- the division of initially identical potential foragers into highly active and largely inactive ones;
- the age-dependent division of foragers into active and inactive ones.

However, in their words, the model is "theoretical and somewhat speculative". In fact, they tested the model only with numerical simulations.

This model emphasises the role played by learning during individual lifetime. It is appealing and tempting to use it also for a group of robots whenever division of labour is needed. However, before doing so, a few initial steps are required. First of all, the theoretical model needs more robust validation. Secondly, it must undergo a deeper analysis in order to be effectively used in other contexts. This is the rationale behind our experiments with the robots.

4.5 Robots' Control Algorithm

We implemented this learning algorithm and tested it with the *MindS-bots* (Sec. 3.1) and the simulated *s-bots* (Sec. 3.2.2).[2] The algorithm was slightly modified to take care of convergence problems. Before presenting the results, this section describes the control system of the robots.

The control algorithm is based on a simple finite state machine. The robots change state according to some predefined conditions, except in one case where the transition is controlled by the learning algorithm. In each state, the robot can perform a predefined set of behaviours, like random walk or obstacle avoidance. In the following, we detail the finite state machine (Sec. 4.5.1), the behaviours (Sec. 4.5.2), and the learning algorithm (Sec. 4.5.3).

[2]For simplicity, we refer to the simulated *s-bot* only as "*s-bot*" in the rest of the chapter

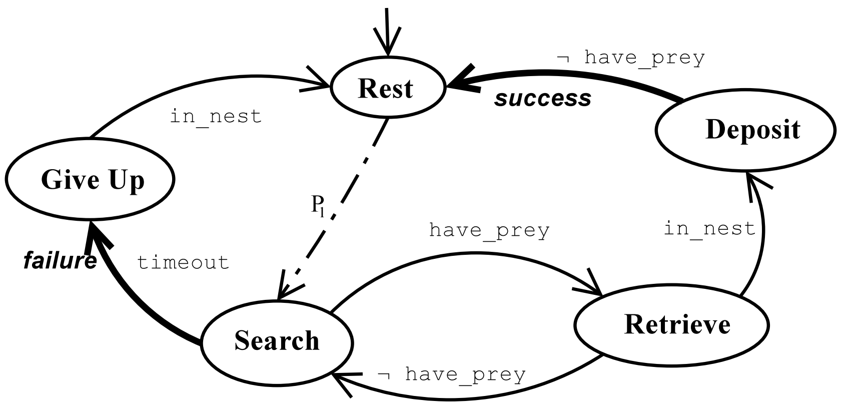


Figure 4.2: Finite state machine that describes the transitions between the states in the robots. The labels on each edge represent the predicates that let the transition occur whenever they are true. Their meaning and definition are given in Table 4.1 and 4.2. The robots start in the **Rest** state. The transition from **Rest** to **Search** (dash-dotted) is based on the probability P_l. The transition from **Deposit** to **Rest** (bold arrow) represents a successful retrieval (P_l is increased), the one from **Search** to **Give Up** (also a bold arrow) is a failure (P_l is decreased).

4.5.1 The Finite State Machine

The finite state machine is depicted in Fig. 4.2. Different states represent the different phases of prey retrieval, that is, the sub-tasks in which the overall prey retrieval task is decomposed. These sub-tasks are as follows:

Search The robot looks for a prey. It has to avoid collision with other robots. If a prey is found, the robot grasps it. If it has spent too much time searching a prey without finding any, it gives up.

Retrieve The robot looks for the nest and pulls a prey into it.

Deposit The robot leaves the prey in the nest and turns on the spot so that its front points outward and its back to the centre of the nest.

Give Up The robot looks for the nest and returns to it.

Rest The robot rests in the nest before restarting searching.

Transitions between states occur on the base of events that are either external (e.g. finding a prey or entering the nest) or internal to the robot (e.g. a timeout). The labels on the edges in the graph of Fig. 4.2 show the conditions (called also *predicates*) that must be true for the transition to occur. The way of evaluating the predicates is slightly different for the *MindS-bots* and the *s-bots*, because of the different sensors. The complete list of the predicates, their meanings and how they are evaluated is given in Table 4.1, for the *MindS-bots*, and in Table 4.2 for the *s-bots*. Their truth values are evaluated from raw sensor

Table 4.1: Definition of predicates and constants used by the control algorithm of a *MindS-bot*. On the left column there is the symbolic name of constants (in italic in the upper part of the table) and of conditions (bottom part). On the right there is a brief explanation of their meanings.

B_T	front light sensor reading when a black object is in the gripper
N_T	back light sensor reading when the *MindS-bot* is in the nest
T_L	maximum time to spend to look for a prey
`in_nest`	back light $\geq$ *N_T*
`gripper_close`	the last command issued to the gripper motor was to close it
`prey_in_gripper`	front light reading $\leq$ *B_T*
`have_prey`	`gripper_close` $\wedge$ `prey_in_gripper`
`obstacle_back`	the back bumper is pressed
`obstacle_front`	$\neg$ `prey_in_gripper` $\wedge$ the front bumper is pressed
`timeout`	time elapsed from the beginning of the search phase $>$ *T_L*

Table 4.2: Definition of predicates and constants used by the control algorithm of an *s-bot*. The format is the same as in Table 4.2.

G_T	maximum gripping distance
N_T	distance of the lamp from the border of the nest
T_L	maximum time to spend to look for a prey
`in_nest`	lamp distance $\leq$ *N_T*
`gripper_close`	the last command issued to the gripper motor was to close it
`prey_in_gripper`	distance of the prey $\leq$ *G_T*
`have_prey`	`gripper_close` $\wedge$ `prey_in_gripper`
`object_back`	the IR sensors on the back signal the presence of an object
`object_front`	$\neg$ `prey_in_gripper` $\wedge$ the IR sensors on the front signal the presence of an object
`timeout`	time elapsed from the beginning of the search phase $>$ *T_L*

readings at every control cycle.

The transition between **Rest** and **Search** occurs with probability P_l each second. Updates of P_l occur during the transitions from **Search** to **Give Up** (a *failure*) and from **Deposit** to **Rest** (a *success*). Section 4.5.3 explains how the probability changes during these transitions.

4.5.2 Behaviours

The robots use a number of behaviours, i.e., sub-procedures that directly react to sensor inputs, to achieve each sub-task. Each behaviour can be executed only if some predicates, called *activation conditions*, are true. Theoretically, there could be more than one behaviour that could be executed and whose actions could be in conflict. For instance, one behaviour could try to avoid an obstacle by going backward and a second one could search in the environment by going forward. To avoid this problem, behaviours are hierarchically ordered and only the first whose activation condition is true is executed. This architecture is known in the literature as *subsumption architecture* [Brooks, 1991].

The behaviours used by the robots are as follows:

AvoidObstacle It tries to avoid an obstacle. If there is only an obstacle on the back of the robot, it moves forward for short time. If there is only an obstacle on the front, the robot moves backward, and then rotates with equal probability either to the right or to the left. Rotation was introduced to avoid dead locks. In fact, other behaviours start by moving the robot forward. If they were executed after *AvoidObstacle*, the robot would hit the obstacle again and again. Finally, if there are obstacles on both sides, the motors are switched off and the robot does not move.

CloseGripper This behaviour activates the gripper of the robot and records the new state of the gripper in the robot's memory.

OpenGripper It opens the gripper and records the new state of the gripper in the robot's memory.

Explore It deals with all the aspects of searching and finding a prey. It starts by choosing a random direction, and then moves the robot straight on. If the robot senses a prey, this behaviour moves the robot toward the prey.

We recall that the *MindS-bots* can sense the prey only with their light sensor on the front, while the *s-bots* can use their omnidirectional camera. The *MindS-bots* are penalised because they have to approach the prey directly in order to recognise it, while *s-bots* can just pass nearby. We modified the *Explore* behaviour in the *MindS-bots* to account for this. The *MindS-bots* can randomly stop and turn on the spot to look whether there are prey around. In this way, they increase the probability to find a prey.

SearchNest It searches the direction of nest by following the light gradient. It then moves backward the robot toward it.

Table 4.3: List of behaviours activated in each state of the *MindS-bot*. For each behaviour, its activation condition is described using the formalism of predicate logic. The meaning of each predicate is illustrated in Table 4.1 and 4.2. In each state, behaviours are listed in activation order, the one with highest priority being on the top. The execution of a behaviour inhibits the activation of all the others below it.

State	Behaviour	Conditions
Search	*CloseGripper*	`hit_prey`
	AvoidObstacle	`object_front` $\wedge\neg$`have_prey` $\vee$ `object_back`
	OpenGripper	`gripper_closed` $\wedge\neg$`prey_in_gripper`
	Explore	
Give Up	*AvoidObstacle*	`object_back`
	SearchNest	
Retrieve	*AvoidObstacle*	`object_back`
	SearchNest	
Deposit	*LeavePrey*	`have_prey`
	SearchNest	
Rest	*ExitFromNest*	

LeavePrey It opens the gripper to leave the prey. The *MindS-bots* turn 90° before opening the gripper, and then realign in order to point outward. This gives time to the experimenter to take the prey out of the arena before the *MindS-bot* might exit and hit again against the prey. With the *s-bots*, free prey in the nest are automatically removed by the simulator.

ExitFromNest This behaviour does not move the robot, but is in charge of deciding when the robot should start looking for prey.

The robot might execute one or more of these behaviours in each state using different priorities, as summarised in Table 4.3.

4.5.3 The Learning Algorithm

P_l, the probability to leave the nest, is modified according to Alg. 2. It keeps trace of the consecutive number of successes and failures of the robots. This number is then multiplied by a fixed amount Δ before being added to or subtracted from the probability. The variation of the probability to leave the nest is continuously increasing in case of continuous successes or failures. The larger the number of successes (failures) is, the larger is the reward (penalty) given by this algorithm, which thereby introduces a form of non-linearity. In the following, we refer to this algorithm as Variable Delta (VD) learning algorithm.

The range of P_l is limited to $[P_{\min},\ P_{\max}]$ in order to avoid that a null or a too high value is reached. In fact, if all robots had $P_l = 0$, none of them would exit the nest any more. If P_l was too high, adding or

Algorithm 2 Variable Delta learning of P_l, the probability to leave the nest. The variables *succ* and *fail* are the number of consecutive successes and failures.

```
$succ \leftarrow 0$;   $fail \leftarrow 0$;   $P_l \leftarrow P_{\text{init}}$;

if success then
    $succ \leftarrow succ + 1$
    $fail \leftarrow 0$
    $P_l \leftarrow min\{P_{\max}, P_l + succ \cdot \Delta\}$
fi

if failure then
    $succ \leftarrow 0$
    $fail \leftarrow fail + 1$
    $P_l \leftarrow max\{P_{\min}, P_l - fail \cdot \Delta\}$
fi
```

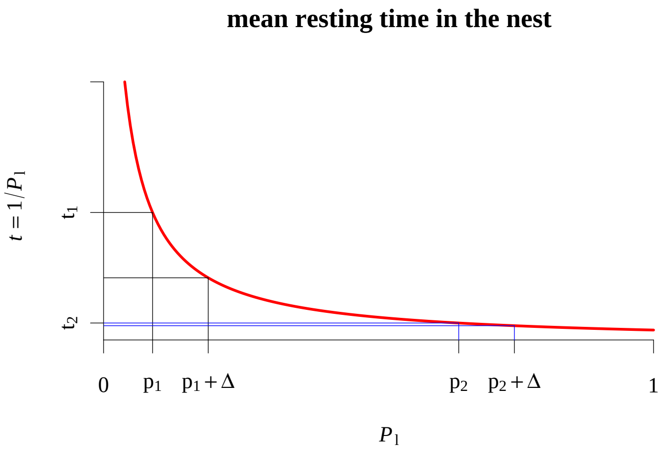


Figure 4.3: Relationship between P_l and the mean time spent in the nest. Adding a fixed value Δ to p_2 has no noticeable effect on the mean time spent in the nest t_2. The effect is more sensible with low probabilities, for instance p_1.

subtracting Δ to it would have no sensible effects on the mean time spent in the nest. The relationship between P_l and the mean resting time is in fact as depicted in Fig. 4.3. It is clear from the plot that adding Δ to p_2 does not change much the mean time when compared to what is obtained adding Δ to p_1.

The algorithm that we implemented is slightly different from the original one proposed by Deneubourg et al. [1987]. More precisely, the original schema uses fixed increments and decrements. In our implementation, the update of P_l is variable, and depends on the number of successes and failures. It is not the purpose of this chapter to discuss and compare different algorithms, but the difference between the two deserves an explanation here.

At the time of the experiments, we were concerned with the time it takes to run the experiments. Such value is particularly important for the experiments with the *MindS-bots*. They are not so robust to stand long experiments, and especially their batteries tends to empty quite fast.[3] In a preliminary set of experiments, we wanted to understand how much time it would take to the robots to adapt to unknown environments. We took four *MindS-bots* and put them in an arena with eight prey, which is a relative high number given the dimensions of the arena and the number of robots. Then we observed the dynamics of the P_l of the robots. Figure 4.4(a) shows a typical result when the Alg. 1 is used. The *MindS-bots* reach $P_{\min}$ more than 2000 s after the last prey was retrieved. Using the VD algorithm, this time is reduced to about 1000 s (Fig. 4.4(b)). This suggest that the VD allows faster convergence to the equilibrium point of the system. Therefore, it allows to carry out the experiments in less time.

4.6 "Learning" vs. "Adaptation"

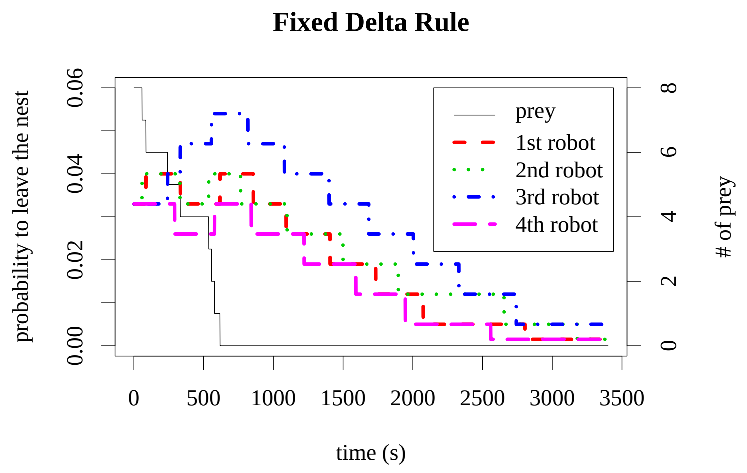
The reader might probably wonder if the Variable Delta is a proper learning algorithm, or maybe just a simple form of adaptation. Our opinion is that this is a matter of definitions. It is the same problem as the use of the words "division of labour", "task allocation" and "specialisation" discussed in Sec. 4.2.

The robotics community tends to associate "learning" with "Reinforcement Learning (RL)" (App. A). The most fundamentalist ones would argue that every work that use the word "learning" should be done in the framework of RL. Algorithms such as VD should be called "adaptation". In biology however, "learning" has a different meaning. It refers to a behaviour observed in animals that produces a "durable modification of [another] behavior in response to information acquired from specific experiences [within a given time scale]" [Alcock, 1995].

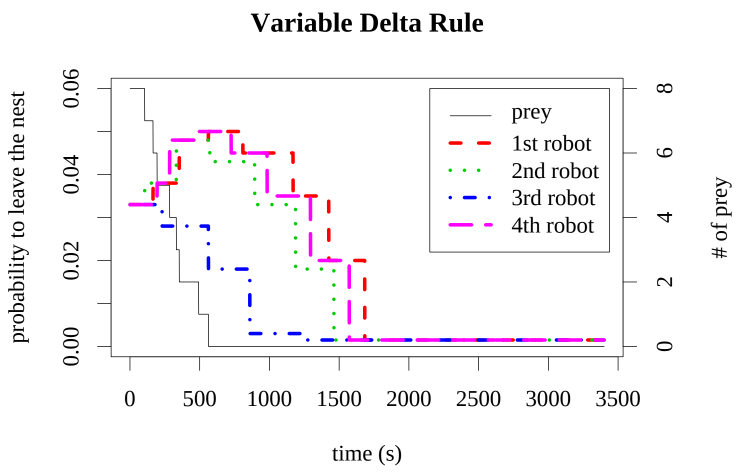
How learning (in the biological meaning) occurs is mostly unknown. There might be different ways of obtaining the "durable modification of behaviours". Identifying RL with learning means to mix the definition of an observed phenomenon with one of its implementation. RL is not the only form of learning, albeit one of the most used in robotics. In the Machine Learning field, for instance, one can find also *Supervised* or *Unsupervised Learning*, *Transduction*, and so forth [Vapnik, 1998, Hinton and Sejnowski, 1999]. The VD algorithm studied in this chapter is another possibility.

It would be of no particular use to compare the VD algorithm with those already proposed for RL, such as Q-learning and $TD(\lambda)$ [Sutton and Barto, 1998]. This becomes clear if we try to model the prey retrieval task using the formalism normally used in these cases, the Partially Observable Markov Decision Process, as we briefly show in App. A. Given the capabilities of our robots, the result is a degenerate case. The methods already developed for RL are best suited for a more complex Markov Decision Process.

[3]The experiments in Sec. 4.8 took 40 min. Full batteries were enough to run no more than 3 experiments in a row.



(a) Fixed increments and decrements to update the probability to leave the nest. The *MindS-bots* reach the minimum value nearly two 2000 s after the last prey was retrieved.



(b) VD algorithm to update the probability to leave the nest. The *MindS-bots* reach the minimum value nearly 1000 s after the last prey was retrieved.

Figure 4.4: Comparison between the dynamics of the probabilities of leaving the nest in two example cases when a system of four *MindS-bots* is given eight prey to retrieve at the beginning of an experiment. Data of the upper plot shows results for *MindS-bots* that used the original algorithm from Deneubourg et al. [1987] (Alg. 1). The bottom plot was obtained with *MindS-bots* that used the VD (see text for details) (Alg. 2). The thin black line is the number of prey in the environment (see the scale on the right axes). The thick lines are the probabilities of leaving the nest for each robot used in the experiment (see the scale on the left axes).

4.7 Experiments

The first concern of our experiments is to validate the theoretical model by Deneubourg et al. [1987]. Afterwards, we can analyse it further in order to understand its' features. For this purpose, we implemented Alg. 2 in a group of robots using two different platforms, as anticipated in Sec. 4.5: the *MindS-bots* (Sec. 3.1) and an *s-bot* simulator (Sec. 3.2.2).

The question whether to use real robots or simulation (or both) has been an important issue in the history of robotics. Till the end of the 80s, it was quite common to use for Artificial Intelligence in general, and robotics in particular, simulators mostly based on a logical representation of the world. This method was challenged by the new born *behaviour-based robotics* [Arkin, 1998]. The position of these researcher was that any simulation is limited, because the model on which it is based will never be able to reproduce the complexity of the environment. Moreover, since the simulation is created by human beings, it could be that we do not implement some important aspects of which we are unaware. Put in simpler word, their position is that the environment is the best model of itself.

Although their argumentation is sound, it conflicts with a very useful aspect of simulation: it is faster to run a program than to run an experiments with real robots. It is also safer: the worst that can happen to a program is that it crashes, but if a robot does the same, it costs money and time to repair it. The speed issue is particularly important for those fields where researcher have to run hundreds of experiments, such as *Evolutionary Robotics* [Nolfi and Floreano, 2000]. Jakobi et al. solved the conflict proposing the concept of *Minimal Simulation* [Jakobi et al., 1995]. In short, they suggested to create a simulator that implements a minimal set of important features of the environment, and wrap the rest with noise and uncertainty. The authors claim that the controller developed with this simulator is indeed more robust. In fact the controller can not be based on "tricks" or particular features that might result from a bad implementation or from wrong modelling of the environment.

We decided to use both simulation and real robots in order to take the best out of them. The experiments with real robots validate the theoretical model of Deneubourg et al., and might be used to define the path for further analyses. The experiments in simulation are first verified against the result with the real robots. Then, they are used to evaluate more efficiently different set-ups and hypotheses. Moreover, using heterogeneous experimental tools, we are more confident that the learning algorithm under study can be applied in a broad range of situations.

In applying Alg. 2 in a group of robots, we expect to observe:

Efficiency increase In the case where the number of exploring robots (those with high P_l) is much higher than the number of prey in the arena, the success rate would be low for most of the robots. The unsuccessful robots would decrease their P_l and spend more time in the nest, leaving more room for the others to work. In the other case, where there are far fewer exploring robots than prey in the arena, some of the robots in the nest would eventually exit. These exiting robots would have a high probability of success. They would therefore increase their P_l and spend more time in foraging. This is an amplification phenomenon typical of many biological systems (Sec. 2.5.1 and Camazine et al. [2001]). The efficiency of the group would improve in both cases without external intervention.

Division of labour Some robots could retrieve by chance more prey than others. The P_l of these "lucky" robots would increase and they would spend more time in foraging. The more the time they would spend foraging, the more prey they would retrieve and the higher their P_l would become. The opposite would hold true for those robots that were less successful. These are again amplification phenomena like those above. Moreover, every time a particular robot retrieves a prey, the probability for any of the other robots to immediately retrieve a prey decreases. Competition among the robots would therefore arise, because one robot can increase its P_l only by making more likely that the P_l of the others decreases. After a while, we would observe that P_l has either high or low values, unevenly distributed among the robots, and only those with high P_l would be actively searching the environment.

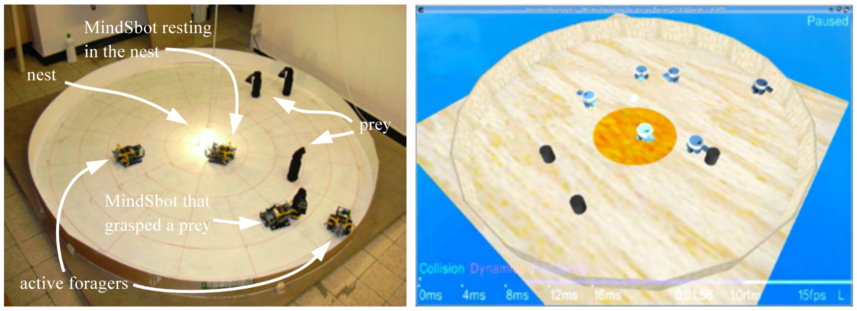
Selection of best individuals In case of a heterogeneous group, some robots might be better suited for retrieving. These individuals would retrieve on average more prey than the others, therefore their P_l would also tend to be higher. At the group level, we would therefore observe a mechanism for division of labour that takes into account differences among the robots without having neither a global knowledge of the individuals in the group, nor a model of their capabilities.

Adaptation to dynamic environments If the density of prey changes, say it increases, some of the robots in the nest would eventually exit and be successful in retrieving. They would therefore increase their P_l. The number of active robots and the efficiency would increase. The same would hold true in the opposite case, when the density decreases: robots that were active would decrease their P_l and spend more time in the nest.

The following section gives additional details about the set-up used during the experiments. Section 4.8 to Sec. 4.11 describe the experiments that we carried out to test these hypotheses.

4.7.1 Experimental Set-up

The experiments were carried out in a circular arena (Fig. 4.5) with a diameter of 2.40 m. A light bulb is used to signal the position of



(a) Snapshot of an experiment with four *MindS-bots*.

(b) Snapshot of an experiment with six *s-bots*.

Figure 4.5: Set-ups of the experiments. The nest is indicated by a light in the centre. The robots in the nest are resting and not active. The other robots are searching the environment, and one in each picture has found and is retrieving a prey.

the nest in the centre of the arena. Walls and floors are white in the experiments with the *MindS-bots*, prey are black.

The search timeout is fixed to 228 s for the *MindS-bots* and 71.2 s for the *s-bots*. These values are the estimated median times needed by a single robot to find one prey when it is alone in the arena.[4] Δ is set to 0.005. $P_{\min}$ is set to 0.0015, $P_{\max}$ to 0.05, P_{init} to 0.033, which correspond to a mean time spent in the nest of approximately 11 min, 20 s and 30 s respectively. The experiments described in the following sections lasted 40 min. The values of the parameters were chosen on the basis of a trial-and-error methodology and on some *a priori* considerations. $P_{\min}$, for instance, is such that the mean resting time in the nest is long enough compared to the duration of the experiments, leaving however to the robots the opportunity of exploring the environment a few times. The mean resting time corresponding to $P_{\max}$ is negligible with respect to the duration of the experiments.

Prey appear randomly in the environment during the experiments. The probability with which this happens each second is referred to as *prey density* and changed across the experiments. A new prey is placed randomly in the arena so that its distance from the centre is in $[0.5 \text{ m}, \ 1.1 \text{ m}]$.

4.8 Experiments: Efficiency

To test whether the VD algorithm increases the efficiency of the group, we generated a set of random *instances* before running the experiments. An instance, parametrised by prey density, describes where and when prey appear in the environment. Then a group of robot was

[4]Notice that these values do not depend only on the speed of the robots, but also on their sensors. The *s-bots* can detect a prey in their surroundings with the omnidirectional camera more easily than the *MindS-bots*, which need to have the prey in front of their light sensor to perceive it.

tested with and without learning for each instance. In case learning was not used, the P_l of the robots is fixed to 1. In such group, all robots search and retrieve prey, which is the solution most often seen in the literature when division of labour is not used.

In Sec. 4.1, we gave a definition of efficiency (4.3). This definition was useful to discuss the issues related to prey retrieval, but it is unfortunately of little practical use. The problem is that the costs of retrieval are difficult to quantify. Therefore, we make use in this section of an estimation of the costs, the *group duty time*. The group duty time is the sum of the time that each robot spent in searching or retrieving, that is, the time it was "on duty". Costs are a monotonically increasing function of the group duty time: the higher it is, the higher the probability that some robot gets lost or breaks down, the higher the energy consumption, and so forth. Thus, the efficiency index that we use is

$$\nu = \frac{\textit{performance}}{\sum_{robots} \textit{duty time}} \ , \qquad (4.4)$$

where *performance* is the number of retrieved prey. It is easy to see that if ν increases, η increases too.

4.8.1 Real Robots

We used groups of four *MindS-bots* chosen out of a pool of six. The four robots were changed after each experiment.[5] Each trial lasted 2400 s (40 min). We created ten instances with prey density set to 0.006 s^{-1}. Each experiment was replicated with a control group made of the same robots with P_l fixed to 1 and using the same instances.[6] Figure 4.6 summarises the results: on the left side, there are the values of ν for both the adapting and the control group; on the right side, there is the ratio between the number of retrieved prey and the total number of prey appeared during the experiment. When the robots used adaptation, there were 2.57 active robots and 2.44 prey on average in the arena in the period between 1000 s and 2400 s. In the control experiments, there were 3.63 active robots and 3.49 prey.

4.8.2 Simulation

The simulation experiments used groups that varied from 2 to 8 *s-bots* with increments of two units. The groups were tested with prey density equal to 0.005 s^{-1}, 0.01 s^{-1}, 0.02 s^{-1} and 0.04 s^{-1}. We used fifty instances for each combination of prey density/group size. To have a reference point, we tested a single robot also on the same instances. The experiments lasted 2400 s (simulated time). As we did with the *MindS-bots*, the experiments where replicated with a control group which did

[5]The choice was not completely random but was biased by the status of the *MindS-bots* after each experiment. For instance, those with low battery or those which needed some maintenance were taken out and new ones were added to the group.

[6]In both original and control experiments, a computer next to the arena signalled the time and the position of the new prey.

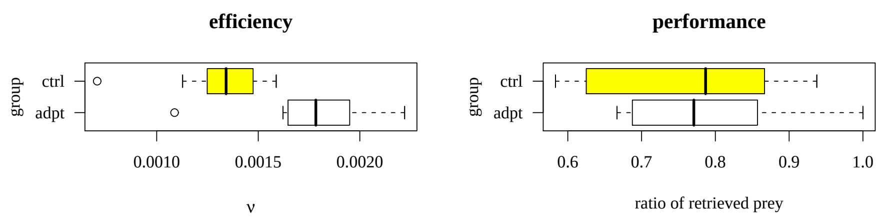


Figure 4.6: Left: Value of ν (Eq. 4.4) when P_1 is adapted (bottom) and when it is not (top) in the *MindS-bots*. Right: relative performance of the adapting and control group (1 on the x axis means 100% of prey retrieved in an experiment). The right and left limits of a box extend from the first to the third inter-quartile of the distribution of the results. The line in the box shows the median value. The whiskers extend to the most extreme data point which is no more than 1.5 times the inter-quartile range from the box. Circles are considered outliers. Data refers to ten experiments.

not use adaptation. The results are summarised in Fig. 4.7, which reports the final distribution of the values of ν for different combination of prey density and group size. Figure 4.8 reports the final distribution of the performances of the different groups in each environment. Figure 4.9 summarises the relative performance of the groups, that is, the number of retrieved prey divided by the total number of prey appeared during the experiments.

4.8.3 Discussion

The group that uses adaptation is significantly more efficient both in the case of the real robots (after 1400 s)[7] and of simulation[8] (confidence level 95%).

There is no statistical difference in the performances between the two groups of *MindS-bots*,[9] while in simulation the control group performs better.[10] However, in the latter case the average difference of retrieved prey is 3.4 units, that is, a negligible amount with respect to the total (see Fig. 4.8). The only exception is for prey density 0.04 s^{-1} and 2 *s-bots*, where the control group retrieved on average 8.9 prey more. The group size seems not to have effect on the performance of the group, but this is mainly due to the fact that the relative number of retrieved prey, that is, the number of retrieved prey divided the total number of prey, is on average already close to 1, as shown if Fig. 4.9.

[7]Sign test. Null hypothesis: ν is the same in the two colonies. Alternative hypothesis: ν, and therefore the efficiency, improves with adaptation.

[8]Wilcoxon rank sum test. Null hypothesis: as in footnote 7. Alternative hypothesis: as in footnote 7.

[9]Permutation test. Null hypothesis: the performances are the same. Alternative hypothesis: the learning group performs better. p-value: 0.53.

[10]Wilcoxon rank sum test. Null hypothesis: the same as in footnote 9. Alternative hypothesis: the same as in footnote 9.

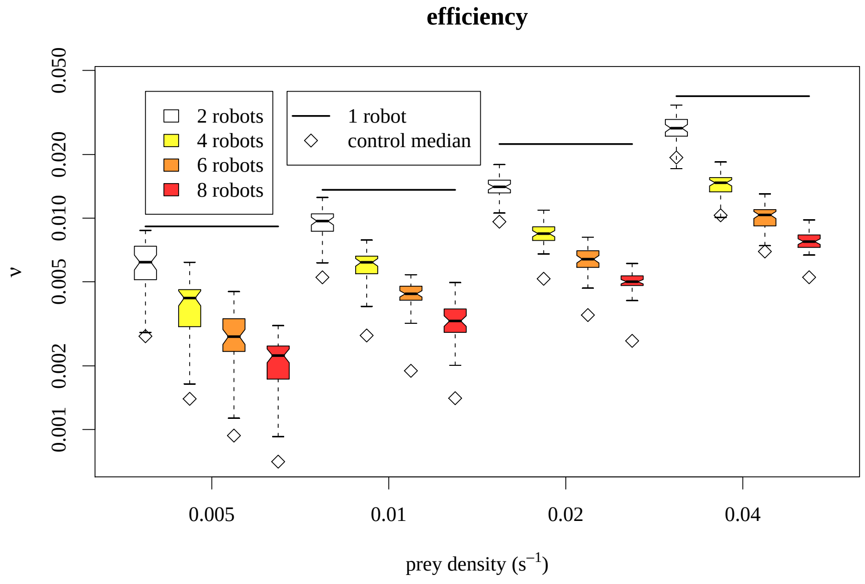


Figure 4.7: Effects of prey density and group size on the efficiency of retrieval in the simulated *s-bots*. The plots report the results of fifty experiments for each prey density (on the x axis), for each group dimension (different filling of the boxes). Both the x and the y axes use a logarithmic scale. The horizontal black line shows the median value of the efficiency of one adapting robot tested in the same conditions as the other groups. The diamonds show the median value obtained in the control experiments. We show only the median values and not the whole distribution for the sake of readability of the plot. See text for the discussion.

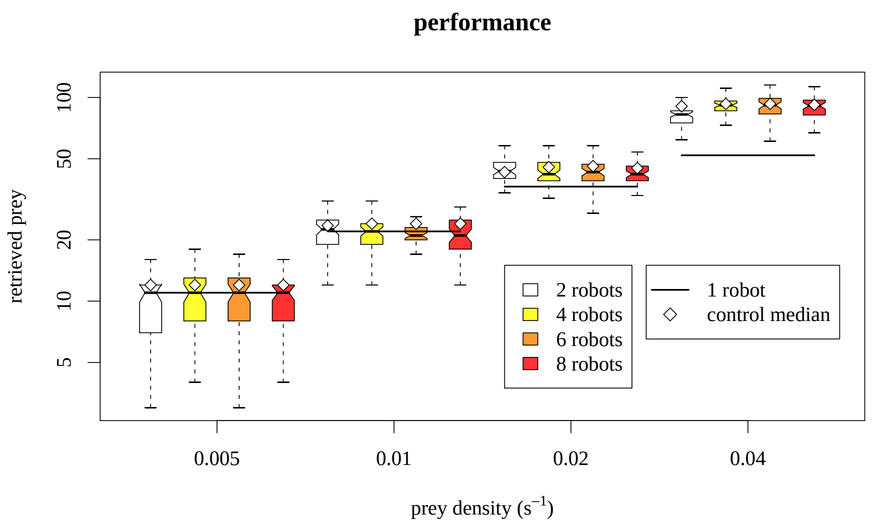


Figure 4.8: Performance of different set-ups in simulation. The x and y axes are in logarithmic scale. The value plotted is the number of retrieved prey. For the meaning of the boxes and the other signs, see Fig. 4.6 and Fig. 4.7. Data refers to fifty replications.

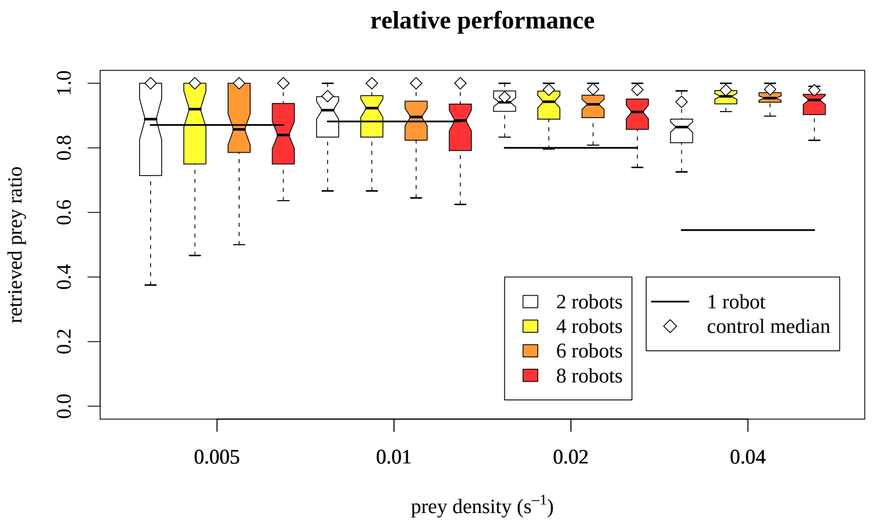


Figure 4.9: Relative performance of different set-ups in simulation. The x is in logarithmic scale. The value plotted is the number of retrieved prey divided by the total number of prey appeared in the experiments. For the meaning of the boxes and the other signs, see Fig. 4.6 and Fig. 4.7. Data refers to fifty replications.

Both in simulation and with the real robot, the differences in performance, if any, are not enough to explain the difference in efficiency. Therefore, the improvement is due to the decrease of the group duty time. We show in Sec. 4.9 that this is achieved by division of labour.

We see in Fig. 4.7 that the gap between adapting and control group tends to decrease when the prey density increases. This is not surprising, because it is better to use all available robots, as the control group does, in rich environments. However, it is important to observe that ν decreases with the group size also when adaptation is used. One possible explanation is that the VD algorithm is not good in reducing the number of explorers. For instance, if the optimal number of explorers is 2 for a given prey density, the robots might end up with 2.5 for a group of 4 *s-bots* and 3.5 for 8 *s-bots*.

An alternative explanation is related to the way we measure ν, which depends on the group duty time. The latter depends on the group size. In fact, all the robots in a group spend some time in searching because none of them can have $P_l = 0$ (recall that $P_l \in [P_{\min}, P_{\max}]$ and $P_{\min} > 0$), so each robot contributes to the final group duty time. These two explanations do not exclude each other, but it is important to test if the first case does occur, as it would show a limit of the algorithm. Section 4.9 provides a partial answer to this issue.

4.8.4 An Analytical Model for the Efficiency

The high quantity of data for the *s-bots* allows us to find a relationship between ν, the group size G, and the density of the prey D. It shall be pointed out that such relationship can show a local trend of ν, that is, it is limited to the set of parameter that we used for the experiments. We try later to generalise the results, but we want to make clear now that it should be considered as a mere speculation and that more experiments are required in order to validate it. Nevertheless, we do believe it is worth to show the results of this generalisation.

The data in Fig. 4.7 shows a nearly linear dependency between the abscissa and the ordinate. Recalling that the graph uses logarithmic scale for both, the relationship between ν and D might be exponential. For what concerns the dependency on G, little can be said for prey density 0.005 s^{-1} and 0.01 s^{-1}, but the results for 0.02 s^{-1} and 0.04 s^{-1} suggest that it might also be exponential. The same observations are valid also for the data of the control group.

The relationship between ν, D and G can be easily found through linear regression. We chose to fit several models based on the considerations above, for example:

$$\begin{aligned}
\nu &= \alpha D - \beta G + \delta \ , \\
\nu &= \alpha \ln D - \beta G + \delta \ , \\
\nu &= \alpha \ln D - \beta \ln G + \delta \ .
\end{aligned}$$

The best fitting model turned out however to be:

$$\ln \nu = \alpha \ln D - \beta \ln G + \delta \ ,$$

which simplifies, after simple modifications, into:

$$\nu = \gamma \frac{D^\alpha}{G^\beta} \ , \tag{4.5}$$

where $\gamma = e^\delta$.

Eq. (4.5) is the best fitting model for both the learning and the control group. The goodness of the fitting was evaluated using standard procedures. Outliers were first discarded (3 out of 800 data points for both type of groups) to find the values of the parameters of the the model. Then, we chose the model which minimises the sum of the square of the residuals. A residual is the difference between the prediction of (4.5) and the empirical value at one data point. We also controlled that the distribution of residuals may be considered a Gaussian distribution. If the residuals were normally distributed, then they would be uncorrelated and could be seen as a white noise applied to the output of the model.

The Quantile-Quantile plots in Fig. 4.10 show that the residual distribution of the model (4.5) approximate quite well that of a normal distribution, except for low values of residuals. The lower tail of the residual distribution is “thicker” than the Gaussian, meaning that there are more low values in the residuals distribution than in the normal distribution. This anomaly accounts however for no more than 5% of all the data points in both the learning and in the control group. We think therefore the residual distribution can still be well approximated by the normal distribution.

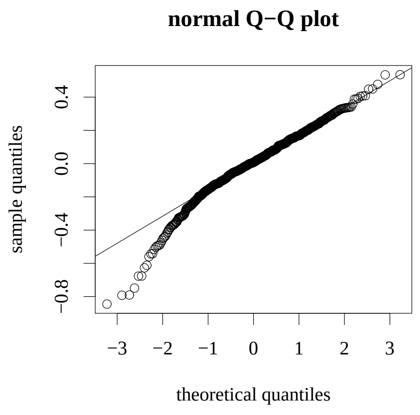
The fact that ν depends on D and G in the same way both in the learning and in the control group suggests that the VD algorithm brings forth only a quantitative improvement in the group. Table 4.4 shows the estimated value for $\langle \alpha, \beta, \gamma \rangle$ and their 95%-confidence interval. The values suggest that the group of learning *s-bots* is more efficient because of a combination of two factors. On the one hand, its lower α means that the contribution given by prey density to ν is higher than in the control group (note that D is a probability per second, thus $D \leq 1\ \mathrm{s}^{-1}$). This suggests that the learning group is better in retrieving prey. On the other hand, the lower β increases the efficiency too, therefore the learning colony is also better in coping with the interferences coming from increasing group size.

Given that

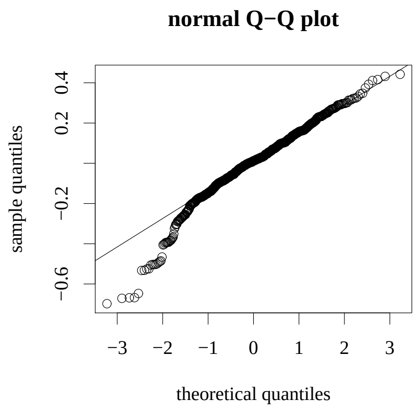
$$\begin{cases} \dfrac{\partial \nu}{\partial D} > 0 \\ \dfrac{\partial \nu}{\partial G} < 0 \end{cases} \quad \forall\ D \in [0,1]\ , \forall\ G > 0\ ,$$

the efficiency is always increasing for increasing prey density and decreasing for increasing group size. This is true for both learning and control groups. This observation is supported by Fig. 4.7.

Figure 4.7 shows also that the efficiency gap between learning and control group decreases when increasing the prey density. Given the model (4.5) and the values in Table 4.4, it is possible to estimate the values of D and G for which the learning group is more efficient than



(a) Q-Q plot for the learning group.



(b) Q-Q plot for the control group.

Figure 4.10: Q-Q plot (Quantile-Quantile plot) of the residuals of the linear models for learning an control groups. The quantiles of the residual sample distribution is plotted against the theoretical quantiles of the normal distribution. If the two distributions were identical, the points should lie on a line. The plots show that the residual distribution is similar to the Gaussian, except for the lower tail. Lower sample quantiles tend to have lower values than the corresponding normal quantiles. The number of data points that are sensibly away from the line account however for less than the 5% of all the data points in both cases.

Table 4.4: Values of the parameters of Eq. (4.5) obtained for the learning and the control colony. Each row reports the estimated value and, below it, the 95%-confidence interval of each parameter.

parameter	*learning*	*control*
α	0.65	0.97
	$[0.63, 0.66]$	$[0.95, 0.98]$
β	0.78	0.95
	$[0.76, 0.81]$	$[0.93, 0.98]$
γ	0.33	0.85
	$[0.30, 0.36]$	$[0.79, 0.92]$

the control group. From

$$0.33\frac{D^{0.65}}{G^{0.78}} > 0.85\frac{D^{0.97}}{G^{0.95}}$$

and after simple simplifications, we obtain that the VD improves the efficiency of the robots if

$$D < 0.52G^{0.53} \ . \tag{4.6}$$

There are two possible reasons to explain why the control group becomes more efficient. Firstly, the maximum value that P_l can have is at maximum $P_{\max} = 0.05$, that is, each *s-bot* spends on average 20 s in the nest. When D is high, an *s-bot* might find and retrieve a prey in 20 s, and therefore it is a waste of time to rest in the nest. Secondly, adaptation is not costless for the group, since it takes some time for P_l to move from the initial value P_{init} to $P_{\max}$, which is again a waste if D is high. These are however hypotheses that have to be tested.

To conclude this section, we want to remind once again that the results of this section come from a series of approximations, and are therefore quantitatively not reliable. This is particularly true for the values in Table 4.4 and for (4.6). They are interesting not for the nominal values *per se*, but for the qualitative features that they highlight. It might still be that the results of this sections are quantitatively correct, but it is not possible to state anything without further experiments. Unfortunately, the particular set-up we used does not allow to test bigger group sizes because there is not enough room in the nest for all the robot. It does not allow higher prey density either, because this would require more experimenter and prey item for the experiments with the *MindS-bots*. Both of this resources were limited at the time of the experiments.

The procedure described in this section has however a very interesting outcome. The parameters $\langle \alpha, \beta, \gamma \rangle$ bind the effect of learning to the environmental conditions. They can be used as a measurement of the quality of an algorithm and thus can be used to compare different solutions, as we do in Chap. 5.

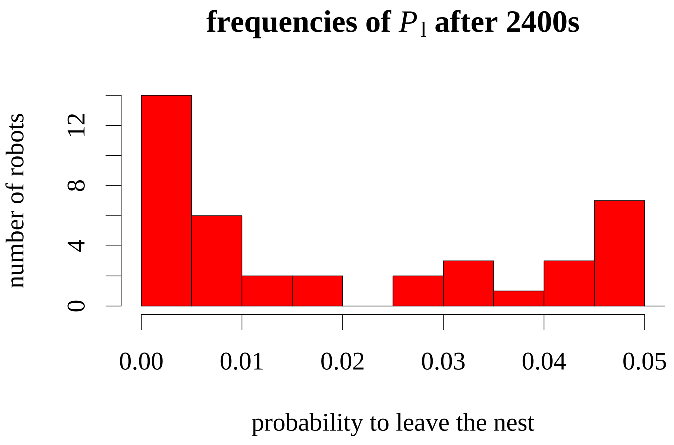


Figure 4.11: Frequencies of P_1 observed in the *MindS-bots* 2400 s after the beginning of experiments. The two peaks demonstrate the occurrence of division of labour. 40% of the observation are above 0.025.

4.9 Experiments: Division of Labour

Section 4.8 shows that the learning group is more efficient. The differences of the performances in both simulation and hardware, when present, are not enough to explain the improvement. Therefore, we deduce that adaptation reduces the group duty time. There are two ways in which the group can achieve this: in the first case, all robots end up having the same, albeit low, P_1 so that the mean number of robots in the foraging area is reduced; in the second case, only few robots are active forager with high P_1 and the others have low P_1. Obviously, the robots with high P_1 would spend more time in searching than the others, therefore we could observe division of labour.

At any given instant t after the beginning of the experiment, the value of P_1 in a robot is a random variable which has different values for every robot and every experiment. Whether the group uses division of labour or not can be observed in the distribution of P_1: if it does, then at the end of the experiments the distribution of P_1 will present two peaks; otherwise it will have only one peak.

4.9.1 Real Robots

The value of P_1 for each *MindS-bot* were recorded during the experiments of Sec. 4.8.1, in order to estimate its distribution. The result after 2400 s, plotted in Fig. 4.11, clearly shows two peaks. The lowest point of the valley seems to be around 0.0225. In fact, there was no *MindS-bot* with P_1 between 0.02 and 0.025. Figure 4.12 shows the distribution of the number of *MindS-bots* with $P_1 > 0.025$ in the ten experiments. Figure 4.13 reports the distribution of P_1 over time.

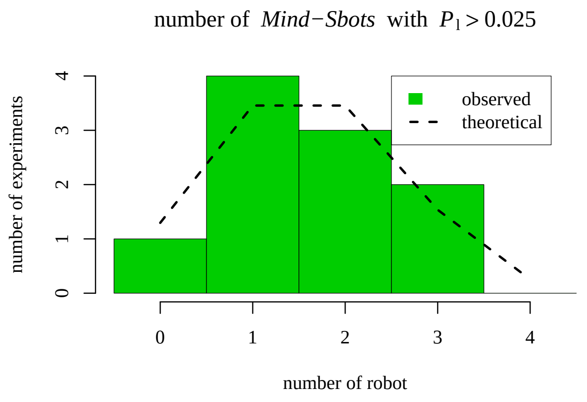


Figure 4.12: Distribution of the number of *MindS-bots* with $P_1 > 0.025$ observed in each experiment compared with the theoretical binomial distribution with $p = 0.4$.

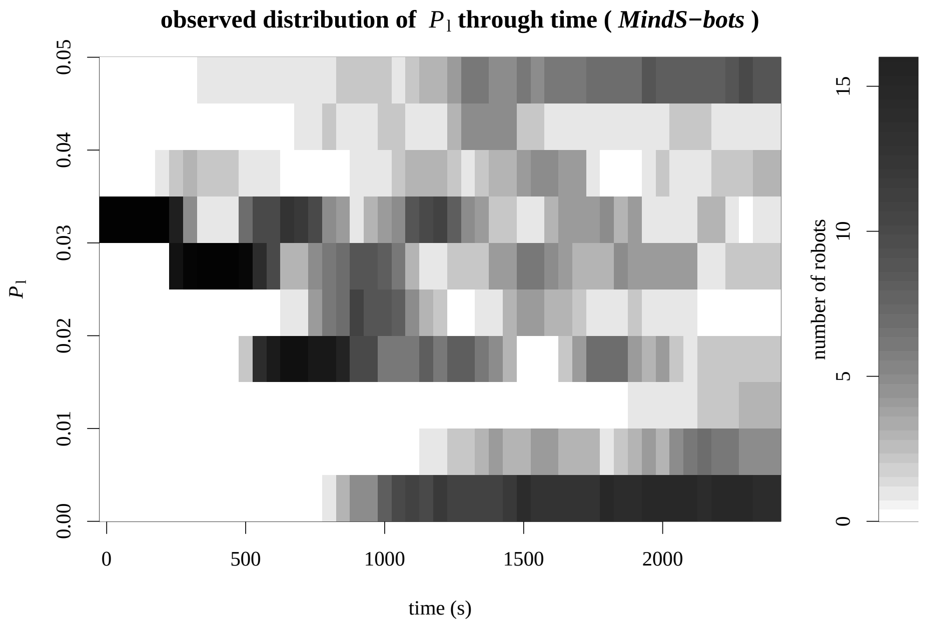


Figure 4.13: Dynamics of the observed frequency of P_1 in the *MindS-bots*. The darkness of a cell in position (t, p) is proportional to the number of *MindS-bots* with $p = P_1$ after t seconds from the beginning of the experiment. The relationship is given by the bar on the right. At $t = 0$ all the *MindS-bots* have $P_1 = 0.033$ (see the black stripe on the left). After 1000 s the number of *MindS-bots* with low P_1 (the loafers) drastically increases (see the dark stripe on the bottom). Similarly, after 1500 s, the number of robots with high P_1 (the foragers) increases, although slowly and reaching a lower value than that of the loafers (top-right part of the plot).

4.9.2 Simulation

We analysed the effects of group size and prey density on division of labour using the data from the experiments of Sec. 4.8.2. The evolution of the distribution of P_1 (Fig. 4.14) is similar to Fig. 4.13, for each combination of prey density and group size. The distributions if Fig. 4.14 show a wider gap between the two peaks than in the case of the *MindS-bots*. To better account for it, we classify the *s-bots* in three classes: *foragers*, *loafers* and *undecided*. *Foragers* are those *s-bots* whose P_1 is bigger than 0.042, while *loafers* have P_1 lower than 0.007, and the rest are undecided (notice that this last group spans over a range of values for P_1 that is five times bigger than the others). Foragers are in fact those in the top stripes in Fig. 4.14, loafers those in the bottom one. Figure 4.15 plots the proportions of *s-bots* belonging to each class at 2400 s. The graphs clearly show a strong division of labour in the colonies, whose individuals tend to have either high or low P_1 but, remembering that the group of undecided spans over a broader range of P_1, seldom values in between.

4.9.3 Discussion

It might be objected that the right peak of Fig. 4.11 could be the result of a few experiments in which all the *MindS-bots* happened to have high P_1. To see that this is not the case, it is enough to look at the number of *MindS-bots* with $P_1 > 0.025$ at the end of the experiments, and how this number is distributed. From the data in Fig. 4.11, we know that 40% of the population has $P_1 > 0.025$. Therefore, the number of *MindS-bots* with $P_1 > 0.025$ in each experiment should follow a binomial distribution with $p = 0.4$. Figure 4.12 shows that the profiles of the theoretical and the observed distributions are indeed very similar.

The evolution of the distribution of P_1 over time (Fig. 4.13) shows that the *MindS-bots* with high P_1 appear later than those with low P_1 (the former at 1500 s, the latter at 1000 s). All *MindS-bots* start with the same P_1, as can be seen from the black stripe for $t = 0$ and $0.03 \leq P_1 \leq 0.035$. After some time, some *MindS-bots* reduce their P_1 because they are not successful (see the black stripes that start at 250 s and 500 s for $P_1 = 0.025$ and $P_1 = 0.015$), while the others alternate successes with failures (indicated by the region in the middle of y range that remains dark till 1500 s). The number of *MindS-bots* in the arena decreases, that is, there are less competitors for those which managed to keep their P_1 high enough. Less competitors implies more and easier retrievals, which increase the P_1 of the remaining foragers. It may be hypothesised that the presence of robots with low P_1 is necessary for the group with high P_1 to appear. However, Fig. 4.14 shows that this is true only for some particular conditions. When the environment becomes richer and richer, the foragers tend to appear earlier and earlier, even before the loafers.

As we expected, the ratio of foragers in the *s-bot* group increases with higher prey density for fixed group size. More interestingly for most prey densities, the proportions of foragers for group of six and

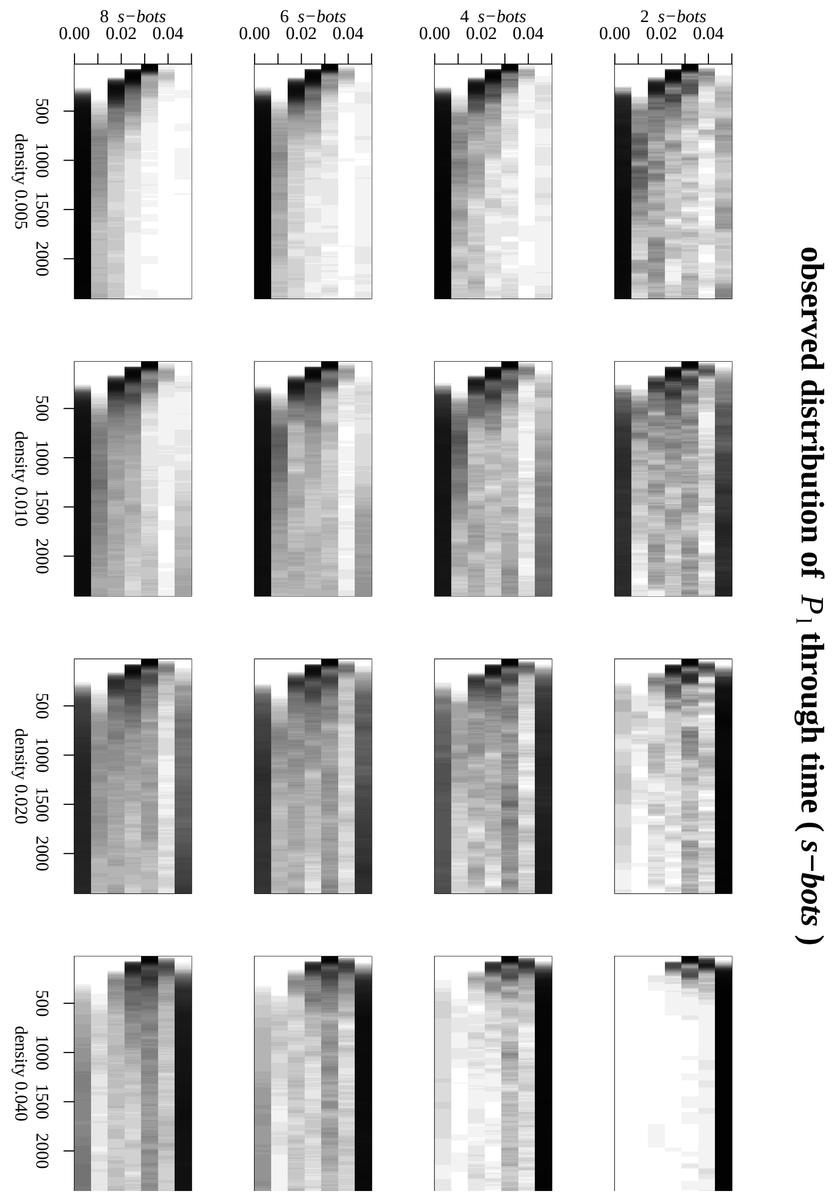
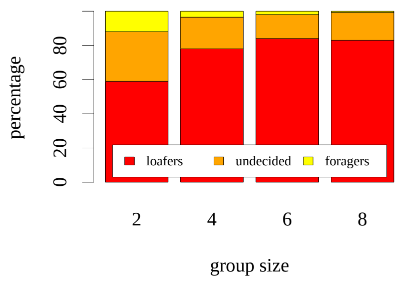
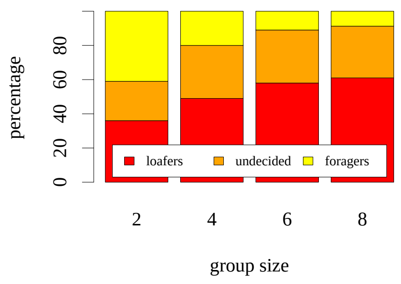


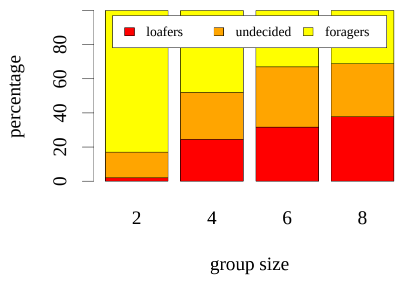
Figure 4.14: Dynamics of the observed frequency of P_1 in the *s-bots*. Each row of the array of plots refers to different group sizes, each column to different prey densities. For the description of the plot, see Fig. 4.13.



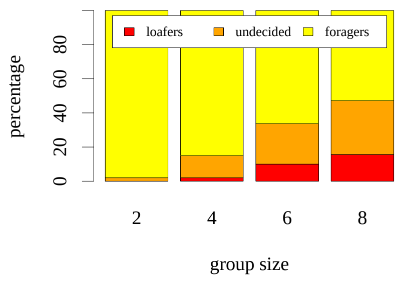
(a) prey density = 0.005 s^{-1}



(b) prey density = 0.01 s^{-1}



(c) prey density = 0.02 s^{-1}



(d) prey density = 0.04 s^{-1}

Figure 4.15: Division of labour in the *s-bots*. Each group of four columns refers to different environments with increasing prey density. Each bar refers to a group size (see the legend). Each bar is divided into three parts whose height is proportional to the ratio of robots belonging to the following groups: *foragers* ($P_l > 0.043$) on the top, *loafers* ($P_l < 0.007$) at the bottom, and *undecided* ($0.007 \leq P_l \leq 0.043$) in between. For example, if the top part is 25% of the total height of the bar for a group of 8 robots, it means that on average 2 robots were foragers.

eight *s-bots* are nearly the same, and thus there are in average more foragers in the latter group.[11] This phenomenon could explains the loss in efficiency when increasing the group size even when *s-bots* learn (Sec. 4.8.3). Because of the particular set-up used for the experiments, we could not test colonies with more than eight individuals: if more robots had been used, there would not have been room enough in the nest for them all. As a mere speculation, it can be said that the VD algorithm, or the set of parameters chosen for it, can be effective only to a certain extent and that other rules or other parameter settings could work better in such conditions.

4.10 Experiments: Selection of the Best Individuals

The VD algorithm is based only on individual successes or failures. If one robot, for any reason, is better than the others for the task of retrieving, then it will be more successful and therefore it will more likely become a forager. This might seem an obvious conclusion, but it needs anyway to be validated.

Generally speaking, the differences can be artificially created or intrinsic in the robots. In the first case, for instance, some robots can be intentionally designed for the task of retrieving while the others are designed to explore the environment to find and mark dangerous spots. In the second case, the differences come from the imperfections of the robots' components, which can never be identical (e.g., one motor that is less powerful than another). Any mechanism for division of labour should take into account this type of heterogeneity of the group.

This section shows that individual learning can be effective to select the best suited individuals for the retrieving task. We want to stress the fact that the VD algorithm does not care of the other robots. In fact, a robot neither knows how many nest-mates are present, nor whether it is working in a group or alone. There is no model either of the environment or of the robot's own capabilities.

For what concerns the experiments with real robots, the *MindS-bots* are built identically and the only differences come from the components. For the experiments with the *s-bots*, we artificially introduced some heterogeneity. In the following, we are interested in those robots whose P_l is greater than 0.025. Abusing of the definition given in Sec. 4.9.3, we refer to them as *foragers*. Note that both with *MindS-bots* and *s-bots* the robots are not aware of such concepts as "being a forager" or "being undecided". These are categories defined *a posteriori*, that is, they are arbitrary definitions, whose purpose is to help in the discussion. Therefore, it is possible to modify them, if this helps to explain better the results of the experiments.

[11]This is true also for prey density 0.04 s^{-1}, where there are on average 4.23 foragers in a group of 8 robots vs. 3.98 in a colony of 6.

Table 4.5: For each *MindS-bot*, identified by an unique name, the total number of experiments in which it was used and the number of times it was a forager ($P_1 > 0.025$) are reported. Data refers to ten experiment, four *MindS-bots* per experiment.

ID	Tot. Exp.	$\#_{foragers}$
*MindS-bot*1	6	5
*MindS-bot*2	3	2
*MindS-bot*3	9	1
*MindS-bot*4	9	4
*MindS-bot*5	3	0
*MindS-bot*6	10	4

4.10.1 Real Robots

We used the data from the experiments of Sec. 4.8.1, where we use groups of 4 robots selected out of a pool of $N = 6$ robots. Table 4.5 reports the number of times each *MindS-bot* was observed to be a forager at the end of the experiments.

4.10.2 Simulation

We created six *s-bots* that differ in their maximum speed. More precisely, the maximum speed of the first *s-bot* was equal to the one in the previous experiments multiplied by 0.5. The speed of each of the other five robots was scaled respectively by 0.7, 0.9, 1.1, 1.3, 1.5. The six robots were combined into all possible colonies of four robots, forming fifteen different groups. Each group was tested in the same fifty instances randomly created with prey density 0.01 s^{-1}. The groups were simulated for 2400 s and we counted how many times each *s-bot* in each group ended up being a forager are reported in Table 4.6.

4.10.3 Discussion

Given the stochastic nature of the experiments, we can model the fact that a robot i is a forager at the end of an experiment as a random event. For both the *MindS-bots* and the *s-bots*, we used groups of 4 robots out of 6. This makes a total of $\binom{6}{4} = 15$ combinations of robots. The probability of robot i to be a forager at the end of the experiment may depend on the specific group G_k, $k \in \{1, \ldots, 15\}$, to which i belongs: we denote this probability by $P_{\mathrm{f}}(i|k)$.

There are two possibilities, depending on whether the following condition is true or not:

$$\exists\, i, k, j : \quad P_{\mathrm{f}}(i|k) \neq P_{\mathrm{f}}(i|j), \quad k \neq j \; . \tag{4.7}$$

If (4.7) is true, there is at least one robot for which the probability to be a forager depends on which nest-mates are present, that is, division of labour exploits mechanical heterogeneity of the robots, which is what

Table 4.6: Number of times that each *s-bot* was a forager. Each column refers to one of the six *s-bots*. Columns are ordered by increasing maximum speed. Each row refers to one of all the possible combinations of four robots taken from a group of six and are sorted by ascending mean group speed. Each group of robots was tested fifty times. The number in the cells reports how many times an *s-bot* became a forager ($P_1 > 0.025$), when used in the given group, after 2400 s.

Group	*s-bot*1	*s-bot*2	*s-bot*3	*s-bot*4	*s-bot*5	*s-bot*6
1	6	19	14	26	–	–
2	5	10	21	–	20	–
3	7	11	–	26	23	–
4	10	12	19	–	–	24
5	5	8	–	19	–	26
6	6	–	16	12	25	–
7	4	8	–	–	31	24
8	4	–	11	18	–	19
9	–	14	17	18	17	–
10	3	–	16	–	17	21
11	–	7	16	11	–	25
12	3	–	–	12	20	19
13	–	13	12	–	17	21
14	–	8	–	17	15	20
15	–	–	11	9	11	23

we want to prove. On the contrary, if (4.7) is false, then $P_{\mathrm{f}}(i|k) = P_{\mathrm{f}}(i)$ (that is, the probability of i being a forager is not a function of the group G_k to which it belongs and is independent of other robots) and we have that either the following condition is true:

$$\exists\, i, j : \quad i \neq j, P_{\mathrm{f}}(i) \neq P_{\mathrm{f}}(j)\ , \tag{4.8a}$$

in which case, once again, division of labour exploits mechanical differences, or the following equations are true:

$$P_{\mathrm{f}}(i) = P_{\mathrm{f}}(j) \quad \forall\, i, j \in \{1, \ldots, N\}\ , \tag{4.8b}$$

in which case there is no exploitation of differences (note that (4.8a) and (4.8b) are mutually exclusive).

For what concern the *MindS-bots*, if we assume that (4.7) is false, we can show that also (4.8b) is false considering the data in Table 4.5, which reports the number of times each *MindS-bot* was observed to be a forager at the end of the experiments. In fact, a statistical analysis of this data shows that (4.8b) can be rejected with confidence 95%.[12] Thus, we are in the case where either (4.7) or (4.8a) is true. It is not necessary which one because both of them prove that VD considers individual differences for the division of labour.

This way of proving our thesis might seem complex but it has a great advantage: it allows us to reach significant conclusions with only 10 experiments. Consider the simpler approach of testing all possible groups with different instances: it would require 15 experiments (all the combinations of 4 robots out of 6) for each instance.

For what concerns the *s-bots*, there is a statistically relevant difference both among the groups and the individuals (confidence level 95%).[13]. Therefore we can conclude that both individual characteristics and average abilities of the group are crucial to select the best individuals to become foragers. This suggests that the situation more likely to have occurred with the *MindS-bots* is (4.7). Finally, $P_{\mathrm{f}}(i)$ increases with the speed of *s-bot* i (confidence level 95%).[14]

4.11 Experiments: Dynamic Environments

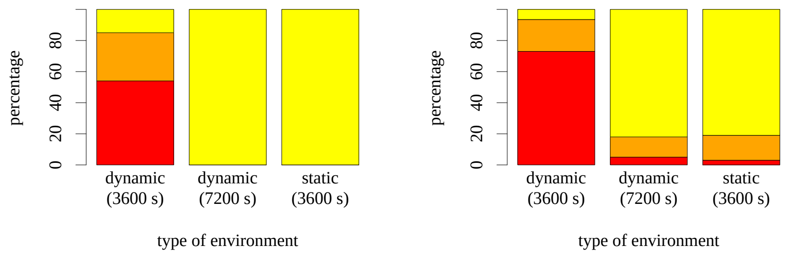
The last series of experiments shows the ability of the group that uses the VD algorithm to adapt to changing environments. The experiments were done only using the *s-bots* because the *MindS-bots* could not reliably work for the amount of time required.

We simulated 2, 4, 6 and 8 *s-bots* in fifty instances of an environment with prey density initially set to 0.005 s^{-1}. The robot were initialised as in the previous experiments. The prey density was changed

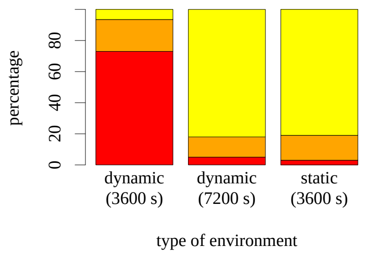
[12] χ^2 test. Null hypothesis: (4.8b). Alternative hypothesis: (4.8a). This test can be used only if the data sets are independent, which is granted by assuming that (4.7) is false.

[13] Two-way ANOVA test. Null hypothesis: no difference between groups and robots. Alternative hypothesis: there are differences among the groups and the robots.

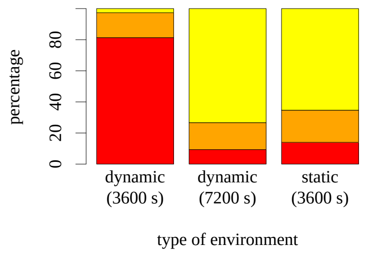
[14] Paired Wilcoxon test. Null hypothesis: $P_{\mathrm{f}}(i) = P_{\mathrm{f}}(j)\ \forall\, i, j \in [1, 6]$. Alternative hypothesis: $P_{\mathrm{f}}(1) < P_{\mathrm{f}}(2) < P_{\mathrm{f}}(3) < P_{\mathrm{f}}(4) < P_{\mathrm{f}}(5) < P_{\mathrm{f}}(6)$. Bonferroni correction was applied to account for multiple tests.



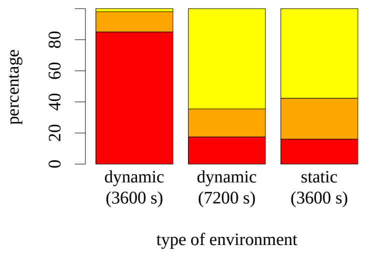
(a) 2 *s-bots*



(b) 4 *s-bots*



(c) 6 *s-bots*



(d) 8 *s-bots*

Figure 4.16: Effects of changing environments on the proportion of foragers. Each graph refers to a different group size. The first bar in each graph shows the division of labour after 3600 s, just before the prey density changes from 0.005 s^{-1} to 0.04 s^{-1} (see Fig. 4.15 for the meaning of the bar). The second bar shows the final proportions at 7200 s from the beginning, and the third one refers to a similar group of *s-bots* that works from the beginning with prey density 0.04 s^{-1} and measured at 3600 s from the beginning.

to 0.04 s^{-1} at 3600 s, later than the overall duration of the previous simulations in order to assure that the group had reached a stable working regime. We then estimated the distribution of P_1 3600 s after the change, that is 7200 s from the beginning of the experiments.

Figure 4.16, which uses the same definitions of *loafers*, *undecided* and *foragers* as in Fig. 4.15 and Sec. 4.9.2, reports the results. The first two bars in each graph refer to the proportions of the three classes after 3600 s and 7200 s in the dynamic environment and show how all the groups can effectively adapt to the new situation. In all cases, the number of foragers increases.

The distribution of the *s-bots* at 3600 s and 7200 s corresponds with the one reported in Fig. 4.15 for similar prey densities. While this was quite obvious for the distribution at 3600 s, it was not so for that at 7200 s since the initial conditions of the group in the second half were different than in Sec. 4.9.2.

To cross check, we performed another set of experiments where the robots were initialised all with $P_{\text{init}} = 0.033$, that is, as in Sec. 4.9. We took from the fifty instances of the previous experiments only the events from 3600 s to 7200 s, and translated back in time so that the robots work from the beginning in an environment with prey density 0.04 s^{-1}. Then we recorded the distribution of P_l in the groups after one simulated hour. The result is shown in the third bars of the plots in Fig. 4.16. We use the words *static environment* to refer to this setup and *dynamic environment* for the one where the prey density changes.

The end distribution in the static environment shows no relevant difference with the one at 7200 s in the dynamic environment.[15] This result shows that the final distribution of foragers in the environment is a function only of the prey density and the group size, thus it is not influenced by the initial conditions.

4.12 Conclusions

The very simple algorithm proposed to model ants' learning can be successfully implemented also on robots. This chapter shows how the simple learning rule of Alg. 2 can improve the efficiency of a group of robots. What is more interesting, is that the robots do not need to explicitly communicate with each other, but they exploit only the local information available in the environment for co-ordination.

The group is more efficient because the robots achieve division of labour. Reducing the number of active robots decreases the interferences among members of the group and increases the quality of retrieval. The algorithm implemented and discussed in the previous pages, the Variable Delta algorithm, is also able to autonomously select the best robots for retrieving. To obtain similar results, most approaches in mainstream robotics would have first estimated the "fitness" of each robot for retrieval, using some well defined metric. The "fitness" values would then be communicated to other robots, and an arbitration would occur. In our case, the robots do not know anything about themselves and how they were built. They even do not know whether there are other robots in the arena.

Our analysis of the VD algorithm shows indeed that a robotic system using it has all the qualities that one wishes for a group of robots: efficiency, scalability and robustness. Our work is however important not only for the robotics community, but for biologists as well. Not many biologists agree that simple rules might be at the origin of group-level complex behaviours. Not many agree that learning could explain the interesting patterns and features observed in ants' (and other species') behaviour. Our work is in fact a strong argument in favour of learning, because we show that it works not only in simple and numerical worlds (like the simulations by Deneubourg et al. [1987]), but also in reality by means of real robots.

The next chapter focuses on different learning strategies. The purpose is to use the knowledge gained in this chapter to compare them and to gain some insight about their applicability in prey retrieval.

[15]Kolmogorov-Smirnoff test. Null hypothesis: The distributions of P_l are the same.

Chapter 5

Other Algorithms for Division of Labour

In Chap. 4, we showed that there is a way for a group of robots to efficiently co-operate without using explicit communication. The solution is based on individual learning, meaning that the robots use only local information to modify their behaviour. We analysed one particular form of learning, inspired by ants' behaviour. The literature offers however solutions similar to ours. Would other learning algorithms prove to be better than Variable Delta (Alg. 2)? In this chapter, we try to answer this question.

Comparisons between different algorithms in robotics is however not straightforward. There are a number of issues that have to be taken into account in order to obtain scientifically sound results. We first introduce the algorithms on which we focus our analysis. Then we describe the issues related to the comparison and detail the way we addressed them. Subsequently, we detail the modifications we did to the algorithms and the results of our analysis.

5.1 Two Learning Algorithms

Among the several learning algorithms proposed in the literature for groups of robots, we decided to focus our attention on two in particular: ALLIANCE by Parker [1998] and the one described by Li, Martinoli, and Abu-Mostafa [2004].[1]

Our choice was done according to the following criteria:

- the adaptation mechanisms used by the two algorithms resemble the VD (Alg. 2), although there are some important differences that might play an important role;
- one algorithm, ALLIANCE, is well known and tested in mainstream robotics, the other was developed for Swarm Robotics (SR),

[1]The authors do not provide a name for their algorithms. For the sake of simplicity, we name it "Li" in the rest of this chapter.

thus they are represent to some degree a broad range of research areas.

5.1.1 ALLIANCE

ALLIANCE, designed by Parker [1998], is a fully distributed, behaviour-based software architecture, developed to obtain fault tolerant co-operative control of teams of mobile robots. Robots learn and adapt their behaviour in an automatic way, even when a centralised knowledge is absent. It is a well known and widely used learning strategy in mainstream robotics.

ALLIANCE was not specifically designed for SR. Robots explicitly communicate their current activities to the other team-mates. Robots use this information to decide the actions to perform, and this is contrary to what is normally seen in SR.

ALLIANCE does not assume or require availability of the communication medium, neither does it require the agents to be fully reliable. ALLIANCE was designed to work also in case of such problems arise. Communication-less robots as in SR are thus seen as a worst-case scenario. It is also worth to note that Parker developed ALLIANCE considering much more complex objectives than prey retrieval. The work of the group of robots is usually divided into a number of tasks that can be executed concurrently or in a given order.

ALLIANCE is based on the following assumptions [Parker, 1998]:

1. the robots of the team can detect the effect of their own actions, with some probability greater than 0;
2. robot i can detect the actions of other team members for which 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 can not necessarily communicate its failure to its team mates;
8. a centralised store of complete world knowledge is not available.

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

The whole mechanism is implemented using *impatience* and *acquiescence*. They are modelled with two mathematical functions, described below. Each task of the team corresponds in the robots to a behaviour set. Impatience and acquiescence are used to evaluate the *motivation* of a robot to execute one 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 in which the robot itself fails to accomplish its task. Each behaviour is assigned a parameter, the threshold of activation (θ), that determines the level of motivation beyond which the behaviour set will become active.

We follow a bottom-up approach to describe ALLIANCE: first we describe some functions that are used as the fundamental building blocks and then how to calculate impatience, acquiescence and motivation. Thereafter, we describe how parameter learning takes place. $R = \{r_1, r_2, \ldots, r_n\}$ is the set of n heterogeneous robots composing the team. $T = \{task_1, task_2, \ldots, task_m\}$ is the set of m independent tasks and $A_i = \{a_{i1}, a_{i2}, \ldots\}$ the set of behaviours for robot r_i. Robots may have different ways of performing a task. ALLIANCE uses a set of n functions $\{h_1(a_{1k}), h_2(a_{2k}), \ldots, h_n(a_{nk})\}$, where $h_i(a_{ik})$ returns the task in T that robot r_i is working on when it activates behaviour set a_{ik}.

The robots use sensors to know whether a behaviour set can be activated. ALLIANCE models this with a simple function:

$$sensory\_feedback_{ij}(t) = \begin{cases} 1 & \text{if the sensory feedback in robot } r_i \text{ at time } t \text{ indicates that the behaviour set } a_{ij} \text{ is applicable, and} \\ 0 & \text{otherwise.} \end{cases}$$

The communication between robots is modelled by the function

$$comm\_received(i, k, j, t_1, t_2) = \begin{cases} 1 & \text{if robot } r_i \text{ has received message from robot } r_k \text{ concerning task } h_i(a_{ij}) \text{ in the time span } [t_1, t_2], \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

In ALLIANCE, when the controller is performing a set of behaviours related to a task, it inhibits the behaviour sets of other tasks. This is to say, that robots perform only one task at a time. To model the task selection, ALLIANCE uses the function

$$activity\_suppression_{ij}(t) = \begin{cases} 0 & \text{if another behaviour set } a_{ik} \text{ is active, } k \neq j, \text{ on robot } r_i \text{ at time } t, \text{ and} \\ 1 & \text{otherwise.} \end{cases}$$

Before formally defining the impatience of a robot, we still need to define three parameters. The first one, $\phi_{ij}(k, t)$, gives the amount of time during which robot r_i is willing to allow robot r_k's communication message to affect the motivation of behaviour set a_{ij}. The second and third ones, $\delta\_slow_{ij}(k, t)$ and $\delta\_fast_{ij}(t)$, give the rates of impatience of robot r_i concerning behaviour set a_{ij} either while robot r_k is performing the task corresponding to behaviour set a_{ij} (i.e., $h_i(a_{ij})$) or in absence of other robots performing the task $h_i(a_{ij})$, respectively. The impatience of the robot is given by

$$impatience_{ij}(t) =$$
$$\begin{cases} \min_k(\delta\_slow_{ij}(k,t)) & \text{if } comm\_received(i,k,j,t-\tau_i,t) = 1 \text{ and } \\ & comm\_received(i,k,j,0,t-\phi_{ij}(k,t)) = 0, \text{ and} \\ \delta\_fast_{ij}(t) & \text{otherwise.} \end{cases} .$$

The impatience rate will be the minimum slow rate, $\delta\_slow_{ij}(k,t)$, if robot r_i has received communication indicating that robot r_k has been performing the task $h_i(a_{ij})$ for the past τ_i time units, but not for longer than $\phi_{ij}(t)$ time units. Otherwise, the impatience rate is set to $\delta\_fast_{ij}(t)$.

The motivation of a robot to perform a task is reset the first time it hears about another robot performing the same task. Using δt to indicate the last time that the robot communicated with the others, the reset is modelled by the function

$$impatience\_reset_{ij}(t) =$$
$$\begin{cases} 0 & \text{if } \exists k : comm\_received(i,k,j,t-\delta t,t) = 1 \text{ and} \\ & comm\_received(i,k,j,0,t-\delta t) = 0, \text{ and} \\ 1 & \text{otherwise.} \end{cases} .$$

This reset function causes the motivation to be reset to 0 if r_i has just received its first message from robot r_k indicating that r_k is performing task $h_i(a_{ij})$.

The robot acquiescence is implemented using two parameters, $\psi_{ij}(t)$ and $\lambda_{ij}(t)$. The first one, $\psi_{ij}(t)$, gives the time that robot r_i wants to perform the task before yielding to another robot. The second parameter, $\lambda_{ij}(t)$ gives the time robot r_i wants to maintain the task before giving up to possibly try another one. The following function indicates when the robot has decided to acquiesce its task:

$$acquiescence_{ij}(t) =$$
$$\begin{cases} 0 & \text{if behaviour set } a_{ij} \text{ of robot } r_i \text{ has been active for more than } \psi_{ij}(t) \text{ time units at time } t \text{ and } \exists x : comm\_received(i,x,j,t-\tau_i,t) = 1 \text{ or behaviour set } a_{ij} \text{ of robot } r_i \text{ has been active for more than } \lambda_{ij}(t) \text{ time units at time } t, \text{ and} \\ 1 & \text{otherwise.} \end{cases} .$$

At this point, each robot evaluates its motivation as follows:

$$\begin{aligned}
m_{ij}(0) &= 0 \\
m_{ij}(t) &= [m_{ij}(t-1) + impatience_{ij}(t)] \\
&\quad \times sensory\_feedback_{ij}(t) \\
&\quad \times activity\_suppression_{ij}(t) \\
&\quad \times impatience\_reset_{ij}(t) \\
&\quad \times acquiescence_{ij}(t)\ .
\end{aligned}$$

The corresponding behaviour set is activated whenever $m_{ij}(t) > \theta$. Robots broadcast their current activities at a rate ρ_i.

Parker [1997] extended the original architecture to allow the robots to learn the parameters used in ALLIANCE: $\delta\_fast_{ij}(t)$, $\delta\_slow_{ij}(t)$ and $\psi_{ij}(t)$. The extended architecture is called L-ALLIANCE, but for the sake of simplicity we refer to it using still the original name, ALLIANCE. Learning uses a performance metric defined by:

$task\_time_i(k, j, t) =$

average time over last μ trials of r_j's performance of task $h_i(a_{ij})$ + one standard deviation of these μ attempts, as measured by r_i.

The parameters are then updated as follows:

$\phi_{ij}(k, t) =$ time during which robot r_i is willing to allow robot r_k's communication message to affect the motivation behaviour set a_{ij}

$= task\_time_i(i, j, t)$

$\delta\_slow_{ij}(k, t) =$ rate of impatience of robot r_i concerning behaviour set a_{ij} after discovering robot r_k performing the task corresponding to this behaviour set

$$= \frac{\theta}{\phi_{ij}(k, t)}$$

$min\_delay =$ minimum allowed delay

$max\_delay =$ maximum allowed delay

$$\begin{aligned}
high &= \max_{k,j}\ task\_time_i(k, j, t) \\
low &= \min_{k,j}\ task\_time_i(k, j, t) \\
scale\_factor &= \frac{max\_delay - min\_delay}{high - low} \\
z_{case_1} &= \frac{\theta}{min\_delay - (task\_time_i(i, j, t) - low) \cdot scale\_factor} \\
z_{case_2} &= \frac{\theta}{min\_delay + (task\_time_i(i, j, t) - low) \cdot scale\_factor}
\end{aligned}$$

$\delta\_fast_{ij} =$ rate of impatience of robot r_i concerning behaviour set a_{ij} in the absence of other robots performing a similar behaviour set

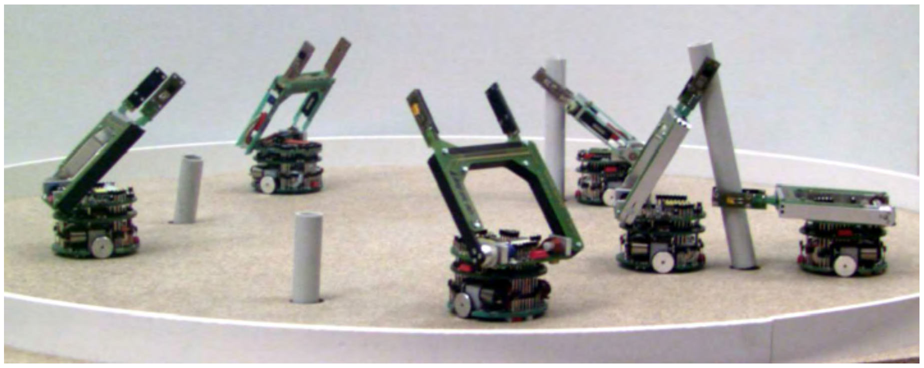


Figure 5.1: Example of collaborative stick pulling. Two robots have to collaborate in order to pull a stick out of the floor [Ijspeert et al., 2001].

$$
\begin{aligned}
&= \begin{cases} z_{case_1} & \text{if the robot expects to perform the task better than any other team member and no robot is currently performing it, and} \\ z_{case_2} & \text{otherwise.} \end{cases} \\
\psi_{ij}(t) &= \text{time robot } r_i \text{ wants to maintain behaviour set } a_{ij}\text{'s activity before yielding to another robot} \\
&= task\_time_i(i, j, t)
\end{aligned}
$$

The reader can find more details about ALLIANCE, L-ALLIANCE and motivations of the algorithm in Parker [1998, 1997].

5.1.2 Li

The second candidate for comparison is an adaptive line-search algorithm presented by Li et al. [2004, 2003, 2002]. The goal of the algorithm is to offer a method to improve the performance of an artificial swarm on a given task.

The task addressed by the authors is distributed stick pulling, as described and analysed by Martinoli and Mondada [1995] and Ijspeert et al. [2001]. 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 can not 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 (Fig. 5.1).

The control system of the agents used by Ijspeert et al. was relatively simple. Each robot randomly moved in the arena. When it found a stick, it tried to pull it out of the ground. The agent could recognise by the speed of the elevation arm whether another team mate was already holding the same stick or not. In the first case, the task is completed; the robots left the stick out of the ground and started searching for new ones. In the second case, the robot held the stick for a given time, waiting for another agent to come and to complete the task.

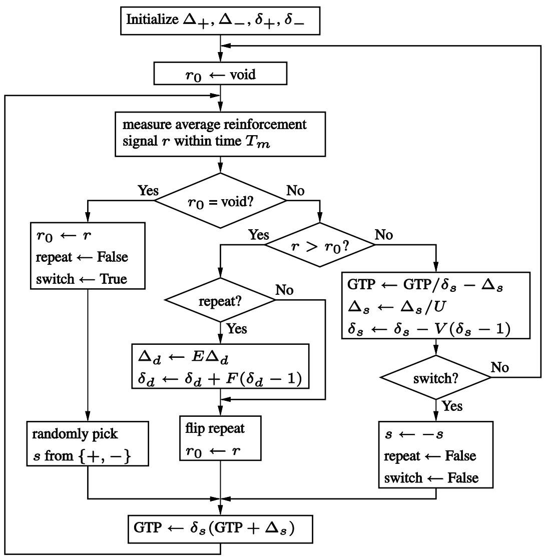

Initialize $\Delta_+, \Delta_-, \delta_+, \delta_-$
$r_0 \leftarrow$ void
measure average reinforcement signal r within time T_m
r_0 = void?
Yes
No
$r_0 \leftarrow r$
repeat ← False
switch ← True
$r > r_0$?
Yes
No
repeat?
No
Yes
$\Delta_d \leftarrow E\Delta_d$
$\delta_d \leftarrow \delta_d + F(\delta_d - 1)$
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)

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]. The meaning of the variables are described in Table 5.1.

The maximum delay that a robot waits holding the stick is called Gripping Time Parameter (GTP). The experiments by Martinoli and Mondada and Ijspeert et al. showed how the GTP can improve the performance of the swarm. The number of robots in the swarm is another important factor that influences the performance.

Li et al. [2004] introduced two generalised versions of the stick pulling experiments. They have the same structure of the original experiment, but they extend its purposes focusing on different aspects of collaboration. The authors explore issues related to sequential collaboration and to parallel collaboration, by using longer or heavier sticks. In case of longer sticks, k robots have to sequentially lift the stick. For heavier sticks, k agents have to lift it at the same time.

The authors then proposed a learning algorithm to adapt the GTP in order to maximise the overall performance of the colony and tested it in simulation. 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 broadcast to all the agents and contains the performance of the group. The global reinforcement signal is particularly important in the generalised version of the stick pulling experiment that requires sequential collaboration, because in this case only the robot that makes the final grip knows if the collaboration has been successful.

The algorithm is illustrated in Fig. 5.2. It starts by initialising the variables that hold offset and multiplication factors of the GTP (δ_-,

Table 5.1: Values of the parameters used in the original work of Li et al. [2004].

(a) Algorithmic variables

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

(b) Algorithmic parameters

	Value	Description
T_m	2400	average period for reinforcement signal s
E	1.9	GTP offset enlarge factor
F	0.3	GTP factor enlarge ratio
U	2	GTP offset shrink factor
V	0.5	GTP offset shrink factor

δ_+, Δ_- and Δ_+). Then, each agent first updates its GTP in a random chosen direction $s \in \{+,-\}$. In the stick pulling experiment is not clear for the robots if an increase or a decrease of the GTP improves the performance. Low GTP would not give time to other robots to come and help. If the robots had high GTP, they would all spend most of the time waiting and never helping the others, except when there are more robots than sticks to pull.

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 updates the GTP 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 (δ_-, δ_+, Δ_- and Δ_+) 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.

In the stick pulling experiment, as well as in prey retrieval, every agent has the same capabilities of the others. In the first set-up, 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. Also the shape of the arena used in the original stick pulling experiments is circular, as the one we prepared for our experimental setup.

5.2 On the Comparison of Algorithms for Swarm Robotics

The two algorithms explained in Sec. 5.1 work similarly to VD (Alg. 2), in the sense that they increase or reduce some parameters to modify the robots' behaviour. But what does "*similar*" exactly mean? According to an English dictionary[2] "*similar*" refers to "objects that have characteristics in common and that are strictly comparable". "*Comparable*" means that the "objects are suitable for a comparison".[3] A "*comparison*" is the "examination of two or more items to establish similarities and dissimilarities".[4]

Just the definition of the word poses us a difficult problem: how do we perform a comparison? At a closer look, the three algorithms (VD, ALLIANCE and Li) are not strictly comparable. They were developed for different applications and they aim at modifying different parameters: probability to exit the nest for VD, time to wait before executing a task and duration of the task for ALLIANCE, time to wait before giving up a task for Li.

Let us focus first on the difference of application domains, and more generally of the contexts in which these algorithms have been developed. A context comprises the goal the robots have to achieve, the environmental set-up and also the robots used for the experiments. If we want to compare objects, we first have to define a common context. It would make no sense, for instance, to compare the colour of a car with the number of apples in a barrel: in the first case we look at the colour because we are interested in the aesthetics of the car, thus is a visual context; in the second case, we are interested, for instance, in the productivity of a farm, thus it is an economical context.

A look at the robotics literature shows that we often lack a common context. Most of the papers propose particular solution for *ad hoc* set-ups. Our work in Chap. 4 is no exception: the VD is tested in robots which work in an environment modelled after ants' foraging. To make things even more difficult, there is no such thing as a common robotic platform which is used throughout the literature. Researchers often use prototypes or robots that were developed by their institutions. Only recently, a set of common robotic platforms seems to emerge (most noticeably, the ones from MobileRobots Inc.[5] and K-Team[6]). Finally, the algorithms, being for particular purposes, might exploit features typical of the application taken into consideration but that might not be found elsewhere. These factors do not allow, for instance, to compare our results directly with the data published by other researchers.

Here comes the need to define first a common context for the experiments. Which one?

It would make sense to test the different algorithms in different ap-

[2]http://www.m-w.com/dictionary/similar
[3]http://www.m-w.com/dictionary/comparable
[4]http://www.m-w.com/dictionary/comparison
[5]http://www.mobilerobots.com/
[6]http://www.k-team.com/

plications, set-ups and with different robotic platforms. Unfortunately, experiments in robotics are costly. Any experiment requires time, people, space if done with real robots, and obviously money to buy/build the robots. Thus, a researcher is forced to limit herself/himself to a limited number of experiments.

This is indeed the normal way to proceed in most sciences. In medicine, for instance, subjects with the same illness are given different (or no) treatments. This allows to find out whether a medicament is effective against a particular pathology. Tests on different drugs or on different pathologies are either done later by the same team, or in parallel by other researchers.

For what concerns our case, we decided to focus only on the two algorithms presented in Sec. 5.1 and, given the hardware and the resources available, in a prey retrieval domain. It is not however an easy task. The robotics literature does not offer, to the best of our knowledge, a formal method to compare algorithms. Indeed, "there are no generally accepted global criteria to evaluate a swarm system's performance" [Rybski et al., 2003]. There are however authors that propose analytical models of their systems, e.g., Lerman et al. [2001]. It would be possible to compare the analytical models derived from different algorithms to find out their differences. This approach has one interesting benefit: in order to write down a model, researchers have usually to identify the main elements of their system and how they interact. In other words: in order to write the model, one has to *understand* the phenomena. There is however one major drawback: one usually needs to make a number of assumptions and simplifications in order to be able to obtain the model. Given the important role that the complexity of interactions plays in fields like Swarm Robotics, it might happen that some important features of the system are overlooked and that the results of the model diverge from that of reality. We experienced this problem in previous research [Trianni et al., 2002].

We show in this chapter a complementary approach. Instead of comparing the algorithms from an abstract point of view, we do it empirically, "*on the field*", basing our results on experiments with real robots and detailed simulations. The experiments take place in well defined laboratory conditions. The advantage of this approach is that we need many fewer simplifications and assumptions than in the previous case. The major drawback is that the experiments require more time and are quite resource-hungry. Thus, we had to plan and carefully design our experiments in order to get sound results with little effort. We give, in the following sections, enough details about our methodology so that other researchers could build on and continue our work. If more and more comparisons like the one in this chapter were carried out in the robotics community (and in the swarm robotics community in particular), they would eventually create a common knowledge-base among the researchers in robotics that would allow for more advances in the field.

There is still one question that needs to be addressed. Having fixed the application, we have then to modify the algorithms and to "port" them to the new task. How much can we change of an algorithm and still call it with its original name? Or, rephrasing, can we relate the results of a modified algorithm with its original one and say, for instance, that “our modified version of algorithm ‘A’ was the best in the experiments, thus ‘A’ is the best algorithm”? The answer is quite easy: no. We cannot consider a modified algorithm as the original one. We can not use the original algorithm either, because its assumptions might not hold true in our domain or because it was developed for different contexts.

The answer to this problem comes from the observation that each learning algorithm in the literature, and those in Sec. 5.1 in particular, proposes a way of adapting some parameters of the robot’s behaviour. What we do, is to implement and/or modify the same adaptation mechanism for *our* parameter, the time the robots spend in the nest (T_l), and then we test the modified versions of the algorithms. Section 5.1 describes our modifications. We warn the reader that in this chapter we refer for the sake of simplicity to our modified version of the algorithms using the original names. Every time the reader reads “algorithm A”, she/he should understand “our algorithm based on the adaptation mechanism of algorithm A”.

5.3 Test Application

There is little to say about the test application used in the experiments: it is the same task and arena used to analyse VD in Chap. 4. We used the same real robots, the *MindS-bots* (Sec. 3.1), and an *ad hoc MindS-bot* simulator, called “MindS-miss” (Sec. 3.2.1). The environment is still as described in Sec. 4.7.1. The value of the parameters for VD did not change, those of the other algorithms are given in Sec. 5.4.

The robots run the same control program as described in Sec. 4.5. The only thing that changes is the way in which robots go from **Rest** to **Search**. This transition is triggered differently according to the learning algorithm under study.

For what concerns the purpose of the experiments, described from Sec. 5.6 to Sec. 5.8, we repeat some of the experiments of Chap. 4: efficiency, division of labour and dynamic environments.

5.4 Modified Algorithms for Prey Retrieval

The algorithms described in Sec. 5.1 could not directly be used in our test application. In this section we describe our modifications to port them to prey retrieval.

5.4.1 ALLIANCE

Our modifications to ALLIANCE address the work hypotheses of the original algorithms that do not hold true in the prey retrieval domain and that we listed in Sec. 5.1.1.

In our test domain, the robots can not directly communicate and are homogeneous, even if run-time problems or mechanical diversities could lead to different efficiency. 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 timeout. The original assumptions 2,4 and 7 have to be modified in the following way:

2. robot i **can not** directly detect the actions of other team members (the robots we used do not have sensor capability to distinguish team mates and their actions);
4. the communication medium is **never available**;
7. if a robot fails, it **can not directly communicate** its failure to its teammates;

Taking into consideration this situation, we can simplify motivation, acquiescence and impatience as we describe below.

Motivation The motivation value is simplified into:

$$m(t) = [m(t-1) + impatience(t)] \cdot acquiescence(t)\ . \qquad (5.1)$$

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

This function is evaluated only when the robot is in the nest. The moment it enters the nest, say t', $acquiescence(t') = 0$ (see below), and thus $m(t') = 0$. For $t > t'$, $acquiescence(t)$ is constant, and thus $m(t)$ depends only on $impatience(t)$.

We eliminated the other components that are based on communication and on the presence of other behaviour sets. Robots can always perform the retrieval task, thus $sensory\_feedback_{ij}(t)$ is always 1.

Acquiescence This function is 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 in searching exceeds the timeout, the function value is set to 0, thus

$$acquiescence(t) = \begin{cases} 0 & \text{if timeout expired and the robot} \\ & \text{is not in the nest} \\ 1 & \text{otherwise.} \end{cases} \quad . \qquad (5.2)$$

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

The value for the timeout is the same used for the experiments in Chap. 4.

As we did for motivation, we had to simplify the original acquiescence function, to eliminate all the parameters dependent on direct communication among robots and on the presence of more than one high level behaviour set. The original formulation of acquiescence was based on the parameter $\psi_{ij}(t)$ that is adapted during the experiments. We did not adapt the acquiescence for two reasons: first, we are interested only in the adaptation of the time to rest in the nest; second, changing the time spent out of the nest, which is equal for the other algorithms, would introduce too much diversity among the algorithms and make our analysis more difficult.

Impatience Considering that in our case robots are not aware of the state of the nest-mates, their impatience values grow independently of each other, as it is specified by

$$impatience(t) = \delta_{\text{fast}}(t)\ , \tag{5.3}$$

$$\delta_{\text{fast}}(t) = \frac{\theta}{min\_delay + (task\_time(t) - low) \cdot scale\_factor}\ , \tag{5.4}$$

$$scale\_factor = \frac{max\_delay - min\_delay}{high - low}\ , \tag{5.5}$$

where $task\_time(t)$ is the average time to retrieve a prey over robot's trials on the prey retrieval task plus one standard deviation.

The constant max_delay is set equal to the timeout. The constant min_delay represents the minimum possible delay employed by a robot to exit the nest, to grab a prey and to retrieve it. We experimentally estimated it by placing a prey just outside the nest and in front of the robot.

In the original description of the algorithms, $high$ and low represent the maximum and minimum value of $task\_time(t)$ for all the robots and all the tasks. Given that the robots can not know the performance of their nest-mates, we set these values to:

$$\begin{aligned} low &= min\_delay \\ high &= max\_delay \end{aligned}$$

and thus $scale\_factor = 1$.

In the experiments with VD in Chap. 4, the initial probability to leave the nest was such that the robots waited on average 30 s in the nest. We initialised the motivation value at time $t = 0$ to $0.87 \cdot \theta$. With this value, the first exit from the nest is after about 30 s from the beginning of the experiment. It should also be noted that, after our simplifications, the actual value of θ is not so important as it was in the original

Algorithm 3 Modified version of ALLIANCE used for prey retrieval.

```
$m(0) = 0.87 \cdot \theta$
**proc** $update\_when\_returning\_to\_nest \equiv$
  $task\_time(t) =$ average searching time+
                    searching time standard deviation
  $\delta_{\text{fast}}(t) = \frac{\theta}{min\_delay+(task\_time(t)-low)}$
  $impatience(t) = \delta_{\text{fast}}(t)$
.
**proc** $rest\_in\_the\_nest \equiv$
  **while** $m(t) < \theta$ **do**
        $sleep(1\text{ s})$
        $m(t) = m(t-1) + impatience(t)$
        **if** $resting\_time > T_{\max}$ **then**
           $m(t) = \theta$
        **fi**
  **od**
.
```

work, thus we set it to 1. The robot can rest in the nest for a maximum of $T_{\max} = \frac{1}{P_{\min}}$ seconds, where $P_{\min}$ is the value used for VD and thus $T_{\max} = 667$ s. This is the maximum average time spent in the nest by robots that use VD.[7]

The algorithm we used for the experiments is summarised in Alg. 3. The two most important features are that this algorithm is time based (we recall that VD is probability based) and that the adaptation occurs on the base of the whole history of the system, summarised by the mean time to retrieve a prey and its standard deviation. ALLIANCE does not make any difference between a successful and an unsuccessful retrieval. In the second case, the timeout value is used when evaluating the new $task\_time(t)$.

5.4.2 Li

In our application, robots do not use communication. Thus, we have to get rid of the global reinforcement signal used by Li et al. when porting their algorithm. We limit our reinforcement signal only 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 search timeout must be considered as a failure). The global reinforcement signal was mostly useful for situations where local feedback was not readily available, such as in the case of sequential collaboration.

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

[7]Notice that VD does not set a maximum amount of time to spend in the nest. For every time t, no matter how big, the probability that a robot rests for more than t seconds is never null, and this for every $P_l < 1$.

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

	Range	Description
Δ_+	[2, 30]	
Δ_-	[-30,-2]	
δ_+	[1.1, 5]	
δ_-	[0.2,0.9]	
r	{`success`, `failure`} ∪ {`void`}	current performance
$repeat$	Boolean	reinforcement flag
$switch$	Boolean	direction flag

Table 5.3: Initialisation values of the enlarge and shrink factors and offsets in our version of the algorithm by Li et al. [2004].

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

decreased to improve the performance of the system. In our case, we need to adapt T_l according to the amount of prey found by the agents in the environment. It is quite obvious that it is better to decrease the time spent in the nest after a success, and to increase it after a failure. It would be pointless to change our parameter in a random way, since the algorithm would waste time in order to find the correct direction for update.

As for ALLIANCE, we set $T_{\max}$ to $\frac{1}{P_{\min}}$ (667 s) and T_{init} to 30 seconds. We also had to find new values and bounds for the other parameters used by Li et al.. They are summarised in Table 5.2. Table 5.3 shows the initialisation values of the parameters. Table 5.4 lists the constants we used in our adaptation of the algorithm, which are unchanged from the original. The values were tuned so that four successful retrievals bring the time to rest in the nest from its lower value (1 s) to its maximum (667 s). This is the same number of iterations that VD requires to go from $P_{\min}$ to $P_{\max}$.

The T_m parameter (average period for reinforcement signal) is useless for our purposes, because the reinforcement is assigned to each robot when it comes back to the nest after a trial (successful or unsuccessful).

Algorithm 4 summarises the structure of our modified version of the original Li's learning strategy. As ALLIANCE, the main difference with respect to VD is that it is time based. As VD, it uses varying increments to change T_l, but it uses a more complex evaluation.

Table 5.4: Constants used in our version of the algorithm, but not changed from the original algorithm by Li et al. [2004].

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

Algorithm 4 Li's algorithm for prey retrieval. T_l is the time an agent has to stay in the nest. The function $restrict\_to\_range()$ checks if its parameters are out of their bounds, and modifies them accordingly.

```
initialisation:
$initialise(\Delta_+, \Delta_-, \delta_+, \delta_-)$
$r \leftarrow$ void;   $repeat \leftarrow$ false ;   $T_l \leftarrow T_{\text{init}}$

if prey retrieved then
  if $r =$ void then
    $repeat \leftarrow$ false
  elsif $r =$ success then
      if $repeat$ then
        $\Delta_- \leftarrow E \cdot \Delta_-$
        $\delta_- \leftarrow \delta_- + F \cdot (\delta_- - 1)$
        $restrict\_to\_range(\Delta_-, \delta_-)$
      fi
      $repeat \leftarrow \neg repeat$
  elsif $r =$ failure then
      $T_l \leftarrow \frac{1}{\delta_+} T_l - \Delta_+$
      $restrict\_to\_range(T_l)$
      $\Delta_+ \leftarrow \frac{\Delta_+}{U}$
      $\delta_+ \leftarrow \delta_+ - V \cdot (\delta_+ - 1)$
      $restrict\_to\_range(\Delta_+, \delta_+)$
      $repeat \leftarrow$ false
  fi
  $T_l \leftarrow \delta_- \cdot (T_l + \Delta_-)$
  $restrict\_to\_range(T_l)$
  $r \leftarrow$ success
fi

if timeout then
  if $r =$ void then
    $repeat \leftarrow$ false
  elsif $r =$ failure then
      if $repeat$ then
        $\Delta_+ \leftarrow E \cdot \Delta_+$
        $\delta_+ \leftarrow \delta_+ + F \cdot (\delta_+ - 1)$
        $restrict\_to\_range(\Delta_+, \delta_+)$
      fi
      $repeat \leftarrow \neg repeat$
  elsif $r =$ success then
      $T_l \leftarrow \frac{1}{\delta_-} T_l - \Delta_-$
      $restrict\_to\_range(T_l)$
      $\Delta_- \leftarrow \frac{\Delta_-}{U}$
      $\delta_- \leftarrow \delta_- - V \cdot (\delta_- - 1)$
      $restrict\_to\_range(\Delta_-, \delta_-)$
      $repeat \leftarrow$ false
  fi
  $T_l \leftarrow \delta_+ \cdot (T_l + \Delta_+)$
  $restrict\_to\_range(T_l)$
  $r \leftarrow$ failure
fi
```

5.5 Experiments: Methodology

The methodology used during the experiments follows the one of Chap. 4. Position and probability of prey appearance, i.e., the *prey density*, were used to generate a set of instances for the experiments. An instance was then used for groups of different sizes and using different learning algorithms. For the experiments with real robots, the instances of the experiments included also which robots from our pool had to be used.

If compared with Chap. 4, we invert here the order of showing the results: first we show the results in simulation and then those with real robots. In Chap. 4, our priority was to validate a theoretical model, that is, to show that it works also in the reality. This is the reason why the experiments with real robots, which are more important in this case, were shown first. The results in simulation were then used for a better understanding of the results.

In this chapter, we do not have to test theoretical models anymore. The algorithms that we study here were already shown to work (although the one by Li et al. [2004] only in simulation). Our favourite experimental tool is now simulation, because it allows to test more conditions than real robots. Simulation needs to be validated though: this is the reason why we experimented also with the real robots and why we show also these results.

5.6 Experiments: Efficiency

5.6.1 Simulation

We tested groups of 2, 4 and 6 simulated *MindS-bots*, using 40 instances obtained with prey density 0.005 s^{-1}, 0.01 s^{-1} and 0.02 s^{-1} (120 instances in total). Each experiment lasted 3600 s.

Figure 5.3 shows the resulting efficiencies for each algorithm in each environmental condition. Figure 5.4 reports the performances of the groups.

5.6.2 Real Robots

The experiments with real robots were also run for 3600 s. Repeating the experiments for all combinations of prey density/group size was not realistic,[8] therefore we focused only on special cases. Validation of the results in simulation and parsimonious use of the resources (i.e., the time required for the experiments) were the rationale for the choice of the cases to test.

We can roughly distinguish the results of Fig. 5.3 in two fuzzy categories: experiments that show a big difference in the efficiencies, and

[8]Three algorithms time three colony sizes time three prey densities means 27 h *per instance*. Considering the time it takes to recharge the batteries and the probability that robots break (thus the probability to have to repeat the experiment, which in our experience was a value next to 0.5), this time can be multiplied by a factor of three.

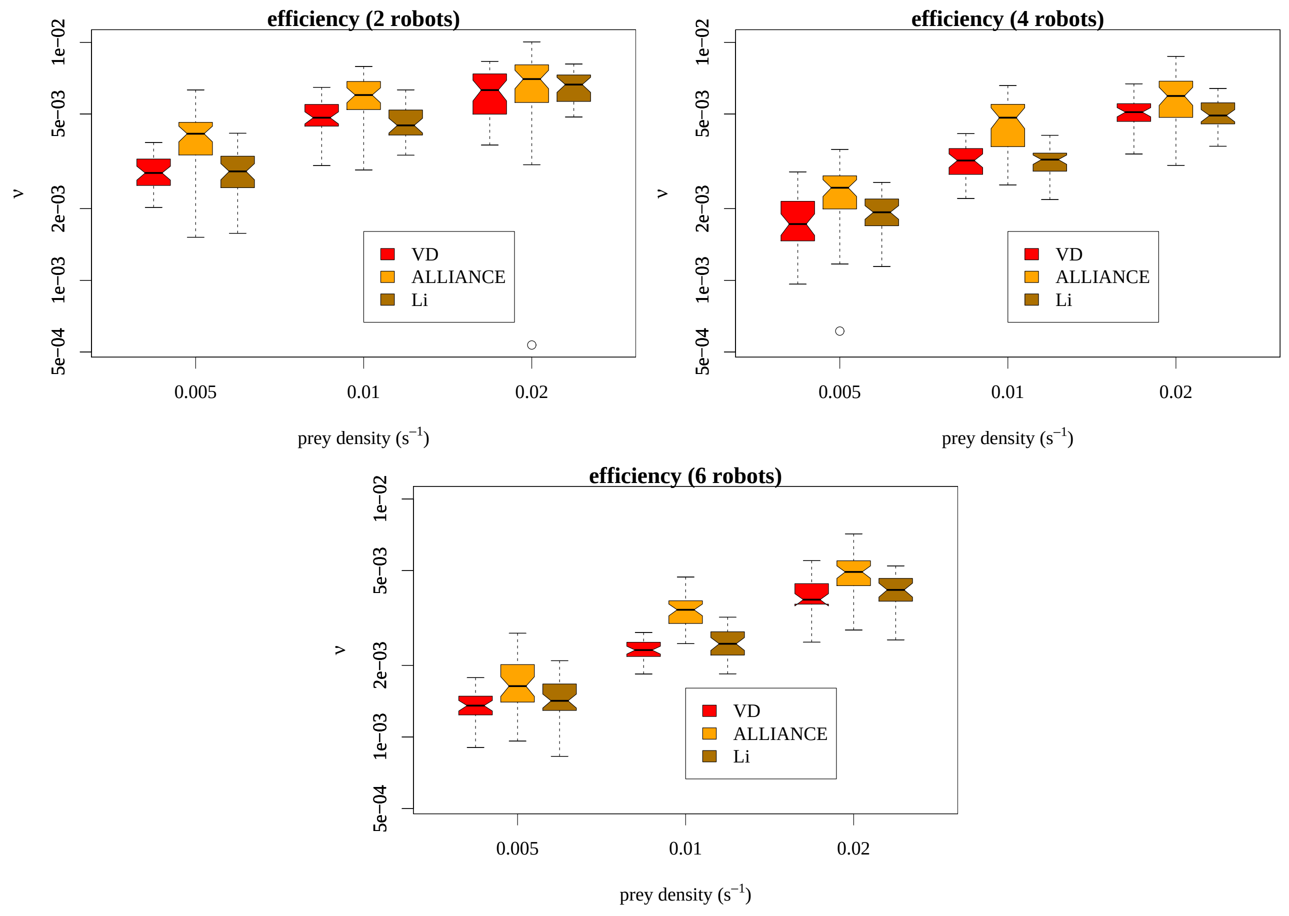


Figure 5.3: Comparison of the efficiencies of the three learning algorithms in simulation. Each plot refers to different group sizes. In each plot, we show the results of the three algorithms for each prey density. For the meaning of the bars, see Fig. 4.6. Data refers to 40 runs.

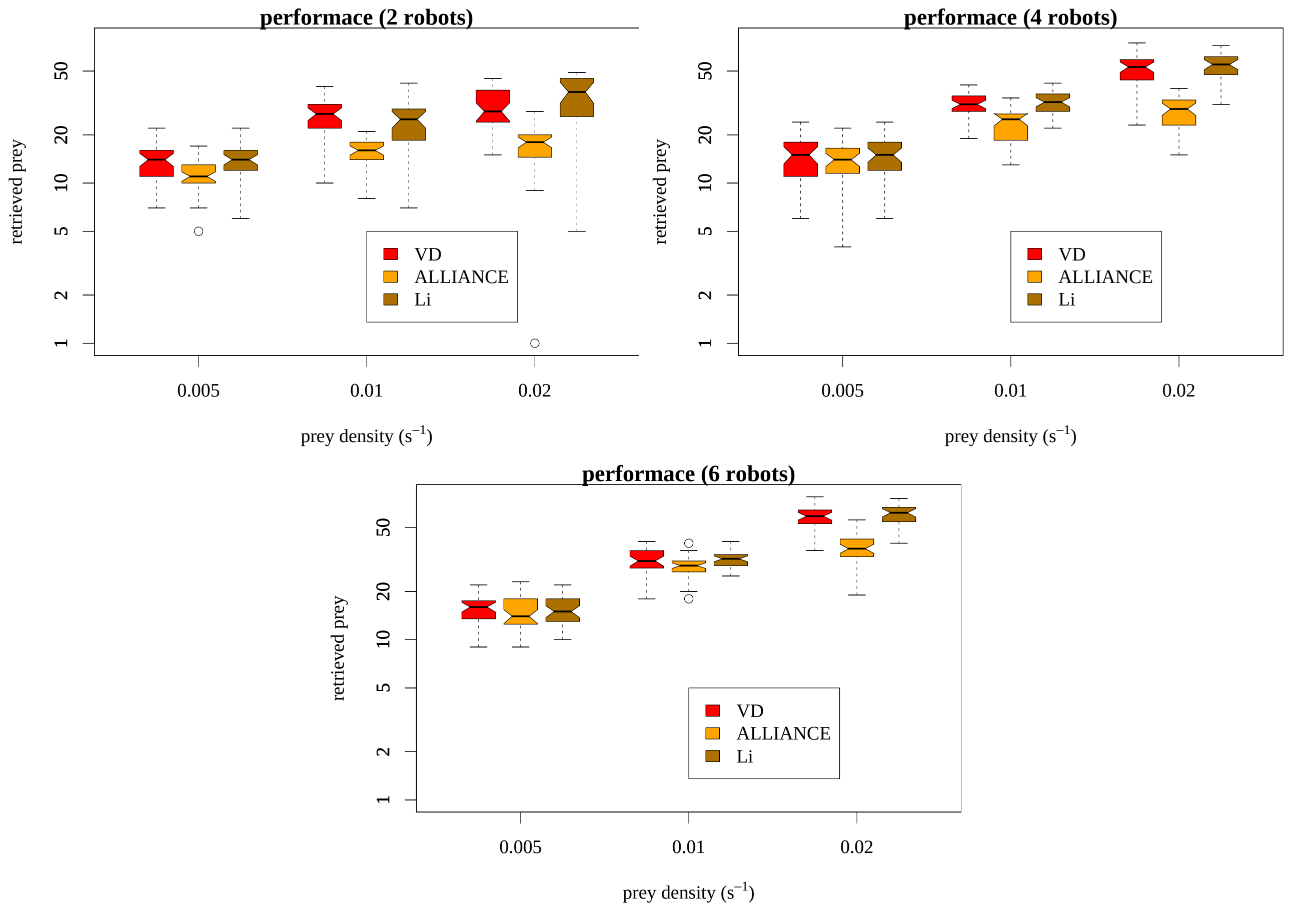


Figure 5.4: Comparison of the performances of the three learning algorithms in simulation. See Fig 5.3 for a description of the plot. Data refers to 40 experiments.

experiments that show a small difference in the efficiencies. We selected one set-up for each category in order to validate the simulation. We have then the following cases:

1. we use prey density D and group size G so that the experiments done with $\langle D, G\rangle$ in simulation show a big difference in the efficiencies, with algorithm 'X' being the best;

2. we use a tuple $\langle D, G\rangle$ that shows a small difference in the efficiencies, and thus nearly no differences between algorithms.

In choosing the values to use for D and G, we had to take care of the following problems. A value of $G = 6$ is not desirable because there is a high probability that robots break and that we have to rerun the experiment (we recall that Lego, while being good for fast prototyping, is not robust enough for long experiments). Moreover, we had in total only 6 *MindS-bots* available, thus one broken robot or one empty battery would block all the experiments, since we would not have had any spare robot to use while fixing the other one. High prey density, $D = 0.02$, had to be avoided too, since it means a high rate of prey appearance. The experimenter could not follow the rate of appearance correctly in placing the prey. Moreover, preliminary experiments showed us that with such rate we would have pretty soon exhausted our prey reserve.

Our final choice was to test the real robots using $D = 0.01$ and group size $G = 4$, as an example of case 1 (big difference in the efficiencies), and $D = 0.005$ and $G = 4$ for case 2. Each set-up was tested with five instances. Figure 5.5 shows the efficiencies of the groups, and Fig. 5.6 the final performances.

5.6.3 Discussion

We first analyse the results in simulation. The efficiencies of the three algorithms are statistically different with 95% confidence.[9] More precisely, ALLIANCE is more efficient than the other two algorithms,[10] but there is no significant difference between VD and Li.[11] It should be noted however that ALLIANCE shows also the bigger variance of the results.

Looking at the performances of the groups, we see that the situation changes. There is also here an important difference among algorithms,[12] but this time ALLIANCE is the *worst* among the three algo-

[9]Friedman two-way test. Null hypothesis: the median efficiency of the tree algorithms is the same. Alternative hypothesis: there is at least one algorithm with different median efficiency.

[10]For both comparisons we used the random permutation test. Null hypothesis: median efficiency of ALLIANCE is equal to the other algorithms. Alternative hypothesis: median efficiency of ALLIANCE is bigger.

[11]Random permutation test. Null hypothesis: median efficiencies are the same. Alternative hypothesis: median efficiencies are different. p-value: 0.6426.

[12]Same test as footnote 9, using performances instead of efficiencies.

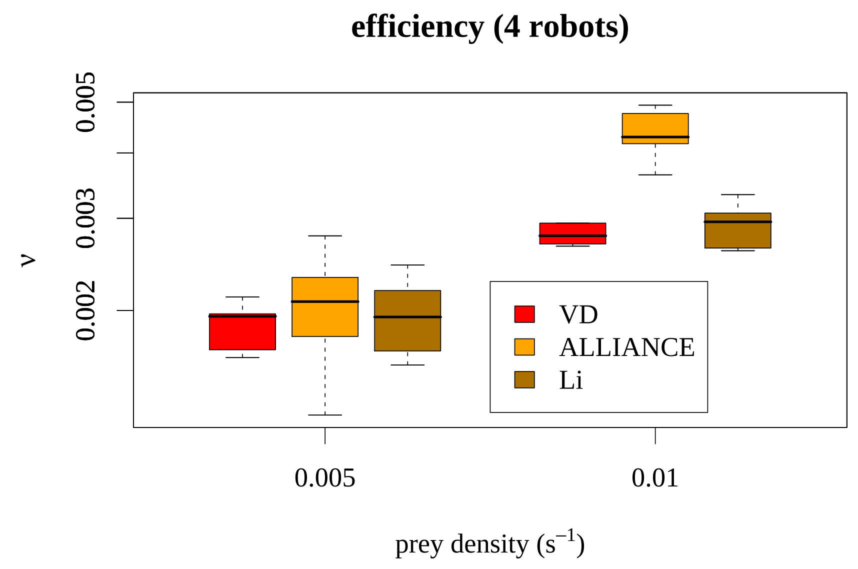


Figure 5.5: Comparison of the efficiency of the three learning algorithms using real robots. The algorithms were tested with a group of four robots, and prey density of 0.005 s^{-1} and 0.01 s^{-1}. Data refers to 5 runs.

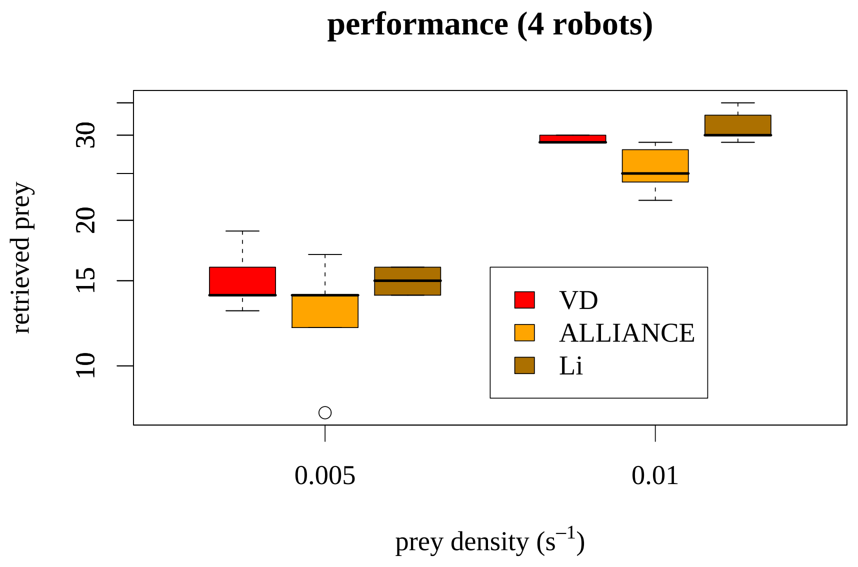


Figure 5.6: Comparison of the performance of the three learning algorithms using real robots. The algorithms were tested with a group of four robots, and prey density of 0.005 s^{-1} and 0.01 s^{-1}. Data refers to 5 runs.

rithms.[13] Again, there is no difference between VD and Li.[14]

We recall that the efficiency is the ratio between the performance of the group and its group duty time (see Eq. (4.4)). Low performance can lead to high efficiency only if the group duty time decreases more than the performance. Thus, it seems that ALLIANCE implements as a matter of fact a very conservative strategy for prey retrieval. The robots rest longer in the nest and exit more seldom than with the other two algorithms. This implies that when they exit, the environment is also richer in prey than with the VD and Li (because less prey have been removed by other robots), thus they are able to retrieve immediately a prey. Moreover, there are less nest-mates that could disturb their retrieval. The drawback of this strategy is that the robots retrieve less prey.

The results with real robots are similar to simulation. If we look at the efficiencies with four robots and prey density 0.01 s^{-1}, we have a statistically relevant difference among the algorithms[15] (and this with only five instances). ALLIANCE is more efficient than the other two,[16] which do not show any difference among them.[17] For prey density 0.005 s^{-1}, there is no statistical difference.[18] Finally, ALLIANCE is the worst performing algorithm.[19]

Can we say that the experiments with real robots validated the simulation? In Sec. 5.6.2 we listed the two outcomes of the experiments in simulation. The possible outcomes with real robots for case 1 in simulation could have been:

1-I. a big difference in the efficiencies, with algorithm 'X' being still the best;

1-II. a big difference in the efficiencies, with another algorithm, say 'Y', being the best;

1-III. a small difference in the efficiencies, which does not give any statistically relevant difference.

In cases 1-I and 1-II a few instances are enough to obtain a significant result. If either case 1-II or 1-III occurs, they are both against the result obtained in simulation, thus we can assert that the simulation is not valid. If case 1-I occurs, simulation and reality give the same results, but we can not draw any conclusion, because we can not be sure that the simulator *always* agrees with reality.

A similar reasoning can be applied to case 2 for simulation. The possible outcomes of the experiments with the real robots are:

[13] Same tests as footnote 10, using performances instead of efficiencies.

[14] Random permutation test. Null hypothesis: median performances are the same. Alternative hypothesis: median performances are different. p-value: 0.8858.

[15] Same test as footnote 9.

[16] Same tests as footnote 10.

[17] Random permutation test. Null hypothesis: median efficiencies are the same. Alternative hypothesis: median efficiencies are different. p-value: 0.2201.

[18] Same test as footnote 9.

[19] Same tests as footnote 12 and 13.

Table 5.5: Linear regression model parameters for the efficiencies of the three learning algorithms. For each algorithm and parameter, we report the nominal value of the fitted model and its 95%-confidence interval.

parameter	*VD*	*ALLIANCE*	*Li*
α	0.70	0.60	0.69
	$[0.66, 0.74]$	$[0.55, 0.65]$	$[0.65, 0.72]$
β	0.58	0.52	0.55
	$[0.52, 0.63]$	$[0.46, 0.58]$	$[0.50, 0.60]$
γ	0.16	0.13	0.15
	$[0.13, 0.20]$	$[0.01, 0.16]$	$[0.13, 0.19]$

2-I. a small difference in the efficiencies, which does not give any statistically relevant difference;

2-II. a big difference in the efficiencies (it does not matter now which algorithm is the best one).

Case 2-II would invalidate the simulator.

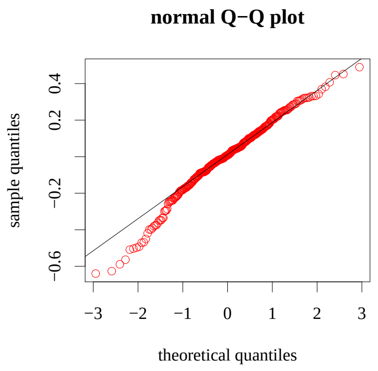
The results with real robots correspond to the only two cases that do not allow us to invalidate the simulator: case 1-I and 2-I. We could have done more experiments with prey density 0.005 s^{-1} in order to have a statistical relevance, but we think that it would have been only a waste of resources. The results in simulation agree so far with those with real robots, both qualitatively and quantitatively (compare Fig. 5.3 with Fig. 5.5 and Fig. 5.4 with Fig. 5.6). Therefore, we thought it would not have been useful to make more experiments, given that at this point it would have been very likely to obtain the same results as in simulation.

We can now try to use the analytical model of the efficiency that we derived in Sec. 4.8.4 in order to measure the differences among the algorithms. The model (4.5) is given by

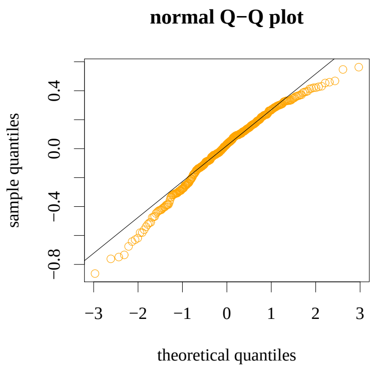
$$\nu = \gamma \frac{D^\alpha}{G^\beta} ,$$

which we fit with the data from the simulation. The values for $\langle \alpha, \beta, \gamma \rangle$ that we obtained for each algorithm are listed in Table 5.5, together with their 95%-confidence intervals. The goodness of fit is shown by the Q-Q plots of Fig. 5.7 (the meaning of these graphs is explained in Sec. 4.8.4).

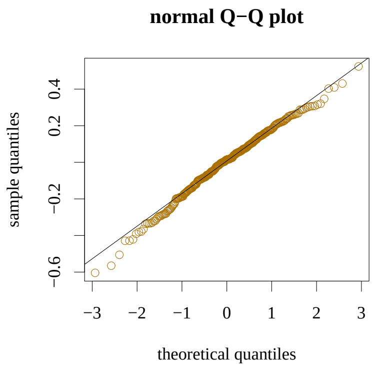
The values of the parameters confirm what we previously observed. Because its α is the lowest, we deduce that ALLIANCE is better than the others in exploiting high prey densities ($\alpha_{\text{ALLIANCE}} < \alpha_{\text{Li}} \equiv \alpha_{\text{VD}}$, but $D \leq 1\ \text{s}^{-1}$), by using a conservative strategy. All three algorithms react in the same way to the group size ($\beta_{\text{ALLIANCE}} \equiv \beta_{\text{Li}} \equiv \beta_{\text{VD}}$).



(a) Q-Q plot for VD



(b) Q-Q plot for ALLIANCE



(c) Q-Q plot for Li

Figure 5.7: Q-Q plot (Quantile-Quantile plot) of the residuals of the linear models for each learning algorithm. See Fig. 4.10 for the description of the plots.

5.7 Experiments: Division of Labour

5.7.1 Simulation

We used the data of the experiments in Sec. 5.6.1. At the end of the experiments, we collected the value of T_l for ALLIANCE and Li, and P_l for VD. In order to have comparable results, we do not use for VD the estimated distribution of P_l, but that of $\frac{1}{P_l}$, that is, the *mean* time to spend in the nest. The results, for different prey densities, group sizes and algorithms are shown in Fig. 5.8.

5.7.2 Real Robots

As in simulation, we used the data from the experiments in Sec. 5.6.2. The results are depicted in Fig. 5.9.

5.7.3 Discussion

The most interesting result is that ALLIANCE decreases the group duty time (Sec. 5.6.3) by *not* differentiating the individuals of the colony. Its distribution of T_l has in fact only one peak, with the majority of the individuals between 200 s and 300 s, in simulation as well as with real robots. Adaptation to different conditions is achieved by modifying the tails of the distribution, but the peak remains nearly always in the same area.

Li's distribution shows mostly one peak too. There are some cases where it is possible to see a small peak for high T_l (i.e., prey density $0.005\,\mathrm{s}^{-1}$, $0.01\,\mathrm{s}^{-1}$ and group size 4 and 6) but they are negligible when compared to those of VD in the same conditions. This is true for both simulation and real robots.

We think that these differences can be explained by two factors. Firstly, given the conservative strategy of ALLIANCE, the competition between the robots is reduced. With ALLIANCE, robots exit more seldom, thus there is time for a conspicuous number of prey to appear in the environment. It is true that one successful robot reduces the number of prey in the environment, and thus the probability that other robots are successful, but the decrease in case of ALLIANCE is negligible.

Secondly, VD introduces a stronger non-linearity than Li and ALLIANCE. Li uses variable increments and decrements of T_l. VD does the same but with probabilities, which are in inverse relationship with time. A decrement of low probabilities corresponds to a huge increment of times, while a decrement of high probabilities corresponds to a small increment of time (see Fig. 4.3). Therefore, VD pushes more easily the individuals toward $T_{\max}$ than Li and ALLIANCE.

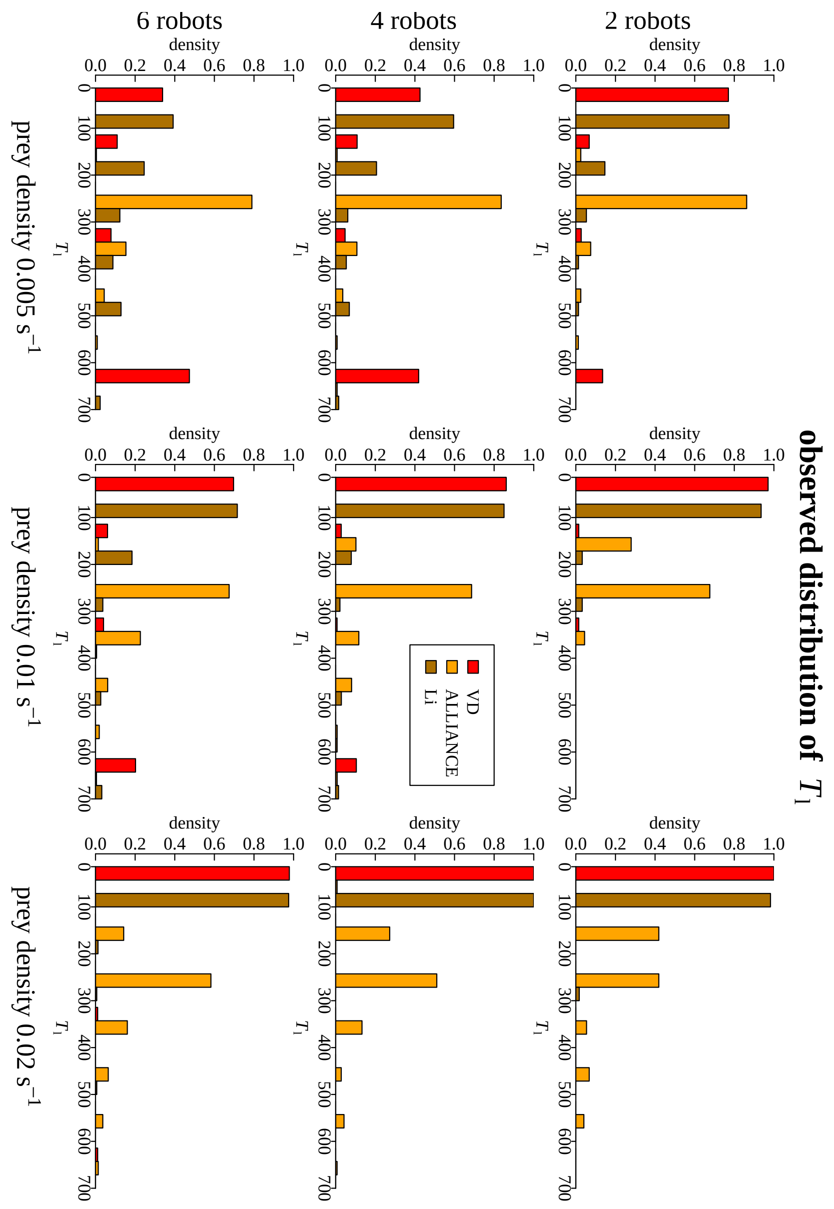
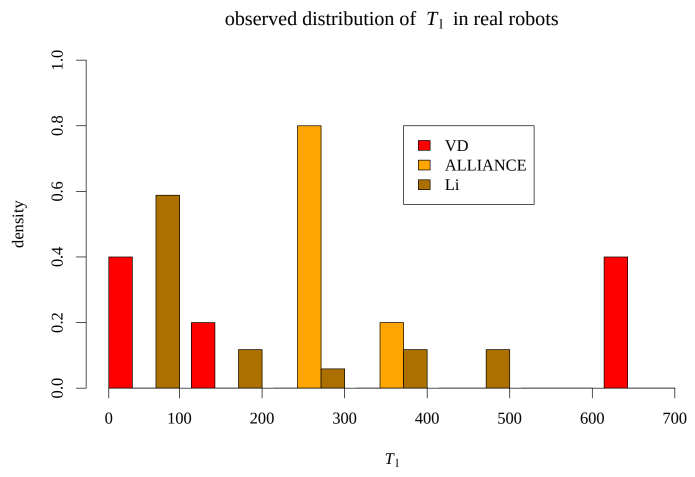
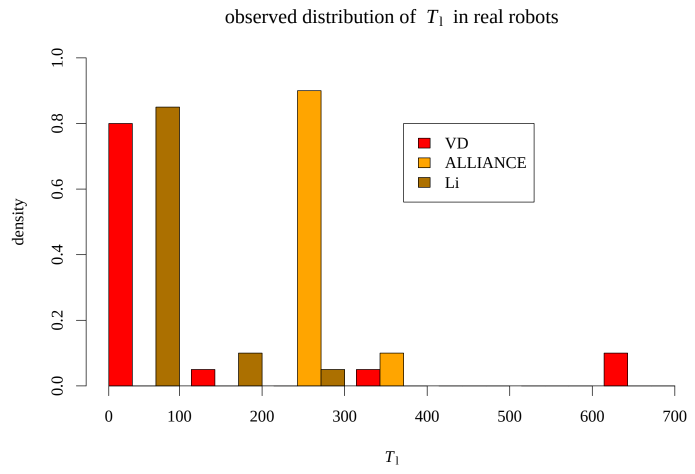


Figure 5.8: Estimated distribution of T_l for each algorithm. Each graph in a row refers to the same group size, specified on the left. Each graph in a column refers to the same prey density, specified on the bottom. The x-axis of each graph is divided into seven intervals, each hundred-seconds long. Each interval contains three bars, one per algorithm. The height of each bar represents the ratio of robots that have been observed to have T_l in the respective interval at the end of the experiments. For VD, we use $T_l = \frac{1}{P_l}$, that is, the mean time spent in the nest. For instance, the rightmost triplet of bars in a plot, refers to the ratio of robots that had $T_l \in (600, 700]$.



(a) Distribution of T_1 for 4 robots and prey density 0.005 s^{-1}.



(b) Distribution of T_1 for 4 robots and prey density 0.01 s^{-1}.

Figure 5.9: Comparison with real robots of the estimated distribution of T_1 of the three algorithms. For a description of the plot, see Fig. 5.8.

5.8 Experiments: Dynamic Environments

We run the same type of experiments as in Sec. 4.11, but only in simulation. Real robots could not in fact reliably work for the time required for the following experiments. The previous sections showed however that the matching between simulation and real robots is really good. We briefly recall in what the experiments consist: a group of robots works initially with a low prey density, 0.005 s^{-1}; after 3600 s the prey density changes to a higher value, 0.02 s^{-1}, and the group works for another 3600 s. The prey densities used in this case represent the minimum and maximum used for the previous experiments. Accordingly, we tested groups of 2, 4 and 6 robots. In all cases, the robots are initialised as in the experiments of Sec. 5.6 and 5.7.

The results are summarised in Fig. 5.10. Both VD and Li adapt the group to the new environment. The number of active robots, those with low T_l, decreases in fact at the beginning of the experiments for all group sizes, and increases after 3600 s. ALLIANCE instead shows not adaptation at all: the number of active robots continue to decrease. It would be fairer to say that ALLIANCE does not show any adaptation in this time-frame (two simulated hours). We hypothesise that the conservative strategy brought forth by ALLIANCE makes the dynamics of the group much slower than the change that we tested in these experiments, and a bigger simulation time might show indeed some adaptation also with ALLIANCE.

The lack of adaptation in ALLIANCE is better seen in Fig. 5.11, where we plot how the percentage of the most active robots changes through time. The "most active" robots are those with $0 \text{ s} \leq T_l \leq 100 \text{ s}$ for VD and Li, and with $200 \text{ s} < T_l \leq 300 \text{ s}$ for ALLIANCE. These ranges correspond to the bottom darkest bands of Fig. 5.10.

We think that the "slowness" of ALLIANCE might be explained by its use of the mean and standard deviation of the time to find a prey (see Alg. 3). Such values are calculated on the base of the whole history of the group, and thus past events play an important role also in the future events of the system.

5.9 Conclusions

We compared different algorithms for learning the time to rest in the nest in a group of robots involved in prey retrieval. Comparisons among different algorithms is not commonly seen in the robotic literature, even less in SR, which is much younger. There are several practical reasons for this: differences in the application domains and in the hardware make difficult for two algorithms to be comparable.

We think that comparisons among different solutions are indeed necessary for the development of robotics. They might be done analytically, but the more important results are those obtained with empirical experiments. In fact, also theoretical results need to be validated.

This chapter shows how we can conduct empirical tests to assess the differences between algorithms. Our way to proceed, mixing simu-

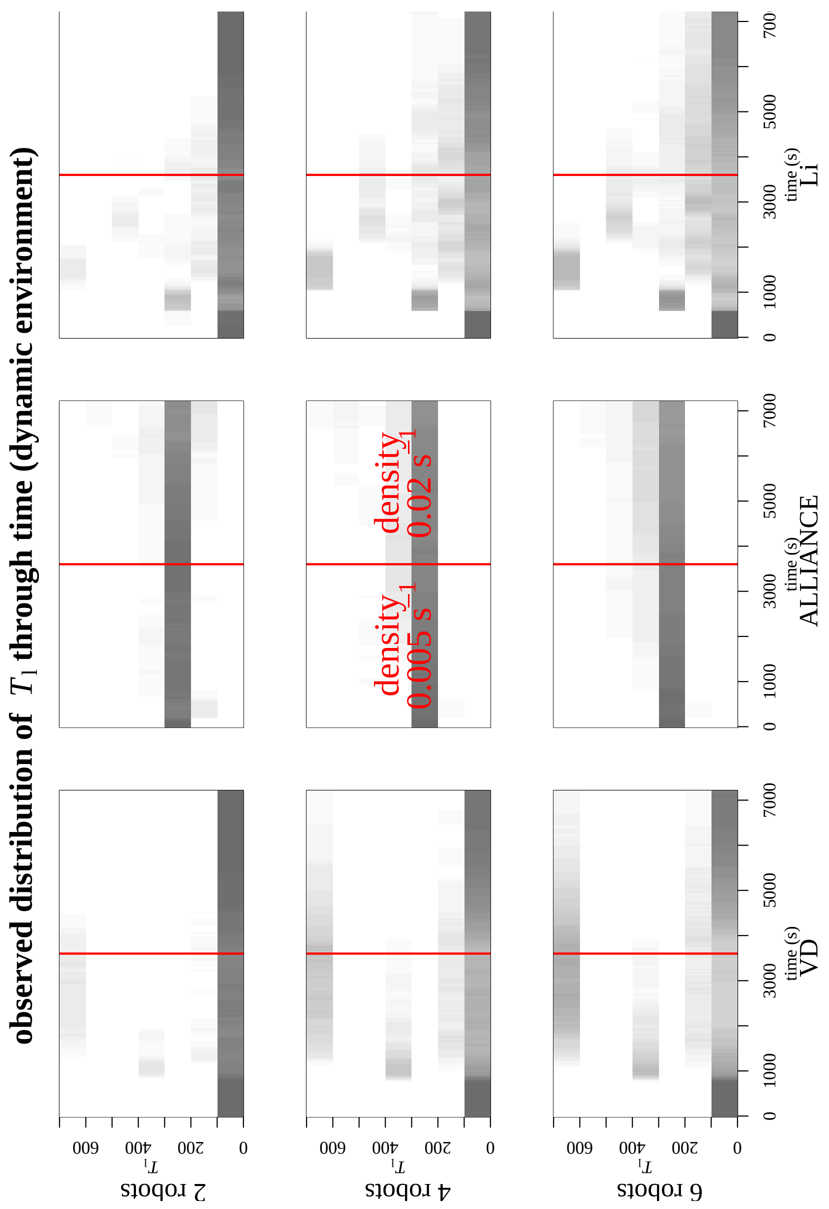
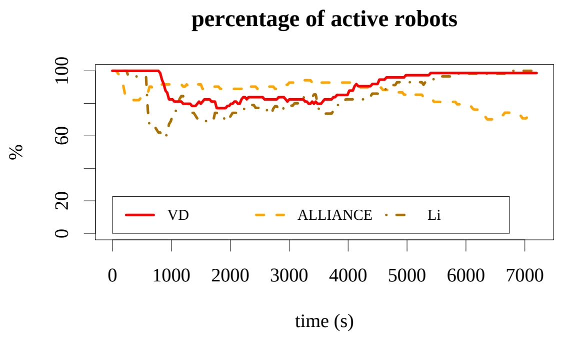
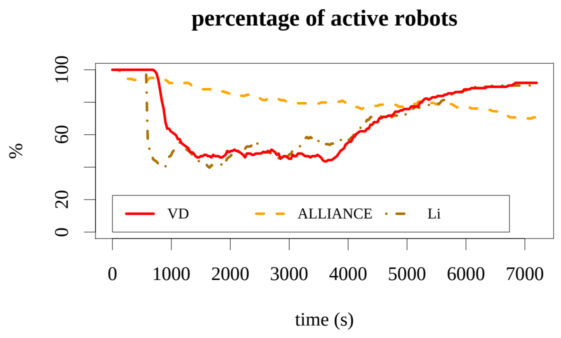


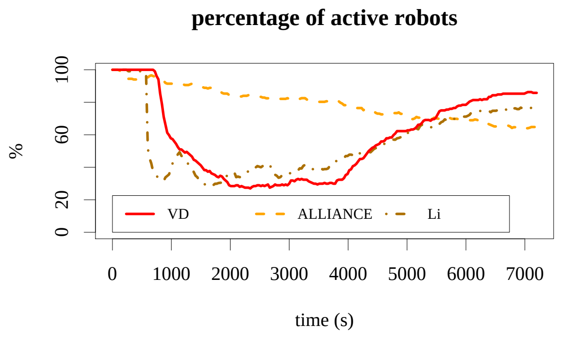
Figure 5.10: Dynamics of T_1 in a dynamic environment. Each algorithm was tested in an environment with prey density initially at 0.005 s^{-1}. After 3600 s, the prey density changes to 0.02 s^{-1}. Each plot in a row refers to experiments with a given group size (specified on the left). Each plot in a column refers to one algorithm (specified on the bottom). The content of the plot is explained in Fig. 4.13. The vertical line in the middle of each plot marks the moment when the prey density changes. Data refers to forty experiments for each combination of algorithm/group size.



(a) 2 robots



(b) 4 robots



(c) 6 robots

Figure 5.11: Percentage of most active robots in dynamic environments. For each colony size, we plot the percentage of most active robots through time. The "most active" robots are those with 0 s $\leq T_1 \leq$ 100 s for VD and Li, and with 200 s $< T_1 \leq$ 300 s for ALLIANCE. Such intervals correspond to the bottom dark lines of Fig. 5.10.

lation and real robots and using appropriate experiment design, allows us to obtain valid results while sparing important resources. Recalling that prey retrieval is used as a model for other real-life applications, we obtained the following practical results:

- ALLIANCE is a very conservative learning algorithm, thus it is best suited for applications in which energy is a major concern (like exploration and collection of samples on a planet, where robots should work as long as possible);
- we can measure the diversities among the algorithms using the formula (4.5) and the estimated values of its parameters, as in Table 5.5;
- VD and Li are not distinguishable for what concerns the outcome;[20] both perform better than ALLIANCE and thus are best suited for applications where the number of retrieved items is more important (like search and rescue);
- ALLIANCE does not adapt well to changes in the environments, thus it is better to choose VD or Li for applications where adaptivity is important (like again search and rescue, or all other scenarios where the environment might change suddenly).

[20]VD is much simpler to describe and implement (compare Alg. 2 vs. Alg. 4), thus it might be a better choice for the designer of a group of robots. It must be noted however that the complexity of Li comes from the fact that it was originally developed for a different application domain.

Chapter 6

Division of Labour in Sensor/Actuator Networks

One of the possible ways of extending the work presented in the previous chapters is by increasing the number of tasks that the robots have to perform. We implemented an architecture for division of labour over several task in a Sensor/Actuator Network (SANET). SANETs are a new research field, where mobile robots co-operate with sensor networks (SN) for the achievement of a goal. The objects composing the SN are referred to as “motes”. We give an overview of the field in Sec. 6.1, and describe which are the common research issues in the area. In order to better understand the work presented in this chapter, we propose in Sec. 6.2 a fictitious application which might require a solution like the one we present in this chapter. This scenario requires that robots and motes perform different tasks concurrently.

Chapter 4 and 5 showed how division of labour can occur thanks to the interactions among the robots and with the environment. More specifically, division of labour is based on positive feedback and on competition among the robots. A successful robot rests less in the nest and thus increases the probability to be more successful (positive feedback). A successful robot decreases the probability of success of the other robots (competition). We tried to introduce the same types of interactions in a SANET to solve scenarios like the one in Sec. 6.2. Sections 6.3 to Sec. 6.6 describe a control architecture for robots and motes that induces division of labour exploiting the same interactions described above.

The second part of the chapter, Sec. 6.7 to Sec. 6.9, shows some of the results that we obtained. We ran experiments in simulation. We took particular attention in implementing a realistic simulator (Sec. 3.3), which is, to the best of our knowledge, something not yet seen in the literature.

SANETs are still a pretty new object of research and their peculiarity is in the high heterogeneity between their components, the motes and the robots. Research both in networking and in robotics has dealt with heterogeneous systems, therefore SANETs should be no problem. We think however that SANETs are a different issue. Let us take for example a heterogeneous group of robots. The individuals are different, but only to a given extent. Some might roll, others might fly, but they are all robots, that is, they can sense, interact with and modify the environment. The difference between robots and motes is more profound and qualitatively different: no matter how one improves or modifies a mote, it will never become a robot. The dualism of SANETs reflects into the approaches that are usually followed in the literature, as we show in the next section.

Robots and motes have different requirements, can do different things and, finally, are usually analysed in different ways. It is therefore not straightforward to find a way to analyse a SANET as a whole. Our experiments were meant to understand the behaviour of the whole group and thus we limit our observations only to the particular aspect of interest for this thesis, i.e., division of labour. There are however several other aspects that are interesting and on which we will base our future work: end-to-end delays, self-organised congestion control, energy aware routing or Quality of Service, just to name a few.

6.1 Sensor/Actuator Networks

Recent advances in electrical engineering have enabled the development of low-cost, low-power, multifunctional “sensor nodes” or “motes”. Motes are small embedded systems, provided with their own power supplies (Fig. 2.4), with the ability of sensing the environment, processing data and communicating with the neighbours, forming a SN.

The research issues connected to SN span over a broad range of topics. Being embedded systems, the electronics of a mote should be designed in order to be compact and to be energy efficient. Motes have also to communicate often with a destination that is farther than the communication range of a single mote. Thus, researchers have focused on the implementation of *ad hoc routing* protocols to allow the SN to work with unknown and dynamic topologies. Trakadas et al. [2004] and Abolhasan et al. [2004] classified and reviewed most of the known routing mechanism for SN, and more generally for mobile *ad hoc* networks: distributed spanning tree, geographic routing, dynamic addressing techniques, cluster routing protocols, and others. Researchers have developed also bio-inspired protocols, most noticeably *AntHocNet* [Di Caro et al., 2005]. We discuss about it more in detail in Sec. 6.4.

The routing algorithms can roughly be classified in three categories: *proactive*, *reactive* and *hybrid*. Proactive algorithms keep up-to-date routes for each destination during all the lifetime of the node. Reactive algorithms search routes to a specific destination only when the upper layer wants to establish a new data session. When the session is closed, the route is usually deleted. The advantage of proactive algorithms is that the data transmission delay is usually smaller, because nodes do not need to find routes to a destination, but at a cost of a high overhead in the network. A lot of packets flow in the network only to maintain the routing information up-to-date. Reactive algorithms reduce the overhead and are also inherently more robust because they do not need to track changes in the network. The disadvantage is that a route discovery takes place every time a node wants to start a transmission. For this reason, hybrid methods have recently appeared. Their purpose is to mix the two routing strategies, proactive and reactive, in order to obtain the best out of them.

This little digression[1] about SNs is justified by the fact that they have recently been coupled with Multi Robot Systems. The result is called "Sensor/Actuator Networks (SANETs)".[2]

There are typically two approaches to SANETs. Researchers in the field of networking usually envisage applications where the robots are "helpers" of the SN, for instance by replacing out-of-work pieces [Corke et al., 2004] or, since they have room for bigger batteries and they can afford more powerful transmission, they work as gateways for long distance communication [Butler, 2003]. Roboticists tend to see the relationship between robots and motes the other way round, using motes as intelligent landmarks for robot navigation [Batalin et al., 2004b, Poduri and Sukhatme, 2004, Corke et al., 2003, Sinopoli et al., 2003]. Navigation is a fundamental problem, because it is required for every activity that the robots should perform. The approach is usually to create a sort of potential field among the motes that the robots follow in order to reach their destination.

One of the reasons for the increasing attention to SANETs in the last years is that there are a number of challenging research issues connected to these heterogeneous systems [Akyildiz and Kasimoglu, 2004]. They have usually strong real-time requirements (think, for instance, to an application like intrusion detection or forest monitoring). Robots and motes have to co-ordinate differently within themselves and between each others. They have moreover conflicting design requirements: on the one hand, the latency of the SANET should be low, so that the robots can react promptly to new situations; on the other hand, energy consumption and network lifetime usually imply a reduced use of the transmission media, and thus higher time to send a message and higher latency.

The protocols used for communication are spread on several, mostly orthogonal, "planes". The *management* plane takes care of the power consumption, the mobility of the nodes and the repair in case of failure. The *co-ordination* plane handles the behaviours of the nodes (both motes and robots) according to the information received from the other planes. The *communication* plane deals with the construction of physical channels, the access into the medium (MAC) and the selection of paths between one node and the other. The challenging aspect is that

[1]If the reader wants to know more about SN, we suggest the works by Chong and Kumar [2003], Akyildiz et al. [2002], Estrin et al. [2002].

[2]The literature offers several other names for the same systems, such as "Robot and Sensor Networks" [Kumar et al., 2004], or "Wireless Sensor and Actor Networks" [Akyildiz and Kasimoglu, 2004].

this three planes have to strictly interact. Data received by a node at the communication plane should be submitted to the co-ordination plane which decides how the node should act on the data. The management plane is responsible for monitoring and controlling the actions decided by the co-ordination plane so that the node operates properly. It can also provide information needed by the co-ordination layer to make decisions.

We could find no work about division of labour in SANETs. There are however a few works about task allocation. As an example, Batalin and Sukhatme [2003] proposed a task-allocation method for alarm response based on the interactions among SN and robots. Their solution makes use of implicit assumptions that can not be granted in case of dynamic environments and/or a huge number of robots in the system. For instance, the motes know how many robots are present and which are their positions,[3] which might be hard to keep in the motes' memory if there are many robots. In case of highly dynamic environments, this solution would require the propagation of the information about each robot in a very short time, or at least faster than the dynamics of changes, but the high data rate might congestion the network. Gerkey and Matarić [2002] proposed a market-based task allocation schema. The agents bid to acquire a task based on the estimation of their capabilities. The authors' auction involves the whole SANET. They are aware that this might be a problem and envisage to solve it by running only localised actions. Clustering methods provide the inherent capability of performing operations in a local context. Younis et al. [2003] exploited this behaviour for task allocation in SNs. Batalin and Sukhatme [2004] used a greedy policy to allocate tasks. Every task is allocated to the best available agent. "It has been shown that greedy algorithms provide good approximate solutions to" the task allocation problem [Batalin et al., 2004a]. Low et al. [2006] used a bio-inspired task allocation mechanism. It is based on the threshold model [Bonabeau et al., 1996] and is used to allocate the task of tracking objects in the network. As in the architecture we describe below, their agents adapt during the network lifetime.

Possible applications for SANETs include intrusion detection (motes detect an intrusion, and robots move to intercept the intruder) and forest monitoring (motes detect a fire and robots shall go to the place and extinguish the fire). SANETs might also be useful for precision agriculture (that is, the ability to intervene in differentiated ways on different zones of a field). Issues in rescue applications have been only recently investigated [Kumar et al., 2004].

6.2 A Scenario for SANETs

To better understand what comes in the next pages, we prefer to start by giving an example of a possible application for SANETs. We refer to

[3]The problem of localisation in SN has also been extensively studied. See, for instance, Galstyan et al. [2004], Corke et al. [2003], Bulusu et al. [2001].

this example in the next sections in order to clearly explain concepts that might remain otherwise too abstract.

The scenario is about search and rescue. The uncertainty of the environment after a disaster, its sudden changes, and the need to act as fast as possible, make this application quite challenging.

Suppose that a SANET is used to help the rescue team after, for instance, an earthquake. As very first step, the rescue team deploys the motes.[4] The task of the motes is to sense the environment and send data to the headquarter. Interesting information to send might be the temperature (to discover fire), sound (in case of some victims are calling for help), pictures and videos (so that the rescue team sees the situation). Robots are used to support the operations of the rescuers. In addition to capturing information, robots can also operate on the field. In the future, they might be able to give first aid to victims, extinguish fire, or secure the place before the rescue team arrives. Motes decide by analysing their readings that a robot is required in a particular area. Robots listen for "help calls" by motes and, in case, reach the problematic spot.

This scenario requires robots and motes (henceforth commonly referred to as *agents*) to perform different tasks. The agents performing a task should be spread in the environment, so that the measurements cover homogeneously the area. Additionally, two neighbouring agents should not perform the same task. It would make no sense, for instance, if all neighbouring motes measure the temperature, because this information is likely not to be different. It is not useful too if motes in another cluster record sound at the same time. In this case, the rescue team would receive redundant information about a part of the environment and nothing about the rest.

To avoid the aforementioned problem, agents need to homogeneously distribute the task execution on the area. This could be done in different ways:

1. the headquarter could send instruction to the agents about what they have to do;

2. agents might be pre-programmed and then deployed on the area;

3. agents distribute the load autonomously using direct communication (e.g., one of the methods described in Sec. 6.1;

4. agents work autonomously but using communication-less co-ordination, that is, without explicitly communicating their activity to the other agents.

Solutions 1 and 3 require the transmission of packets on the network, that is, explicit communication. In opposition to Chap. 4 and Chap. 5, the agents have now a communication medium. In such scenario however, the network is already heavily used to transmit huge

[4]Deployment of motes for a Sensor Network is a broad research area. For the moment, the reader can imagine that motes are dropped from flying vehicles, either autonomous or manned.

amounts of data (videos, sounds, etc.)[5] and thus the medium might be *de facto* useless for co-ordination. Solution 2 shifts the problem to the deployment phase and might require reprogramming in case of changes in the environment. Reprogramming might occur via network or through mobile robots that reach a node and reprogramme it locally. The first solution would stress the network even more, the second would take away resources from the SANET.

In this chapter we present a control architecture for SANETs that aims at solution 4. There are other issues that might be worth investigating in this scenario, like network load sharing, energy consumption, redundancy and robustness. For the time being, we focus only at the implementation of a communication-less co-ordination mechanism.

The following pages describe our architecture, which includes the implementation of the network layer and of the application layer of each agent. The main characteristics of our architecture are:

1. tight interaction between the two layers, which are not seen anymore as two independent modules;
2. probabilistic decisions;
3. exploitation of inter-agent interactions;
4. adaptation.

The previous chapters already showed that exploitation of interactions, adaptation during individual lifetime and probabilistic decision are a valid mean for division of labour. We are going now to test if this holds true also for SANETs.

6.3 Agents' Control: Application Layer

We first describe the application layer, how it chooses the tasks that the individuals have to perform, and its adaptation mechanism. Section 6.4 proceeds by describing the network layer and how it deals with routing of messages to the destination. Finally in Sec. 6.5 and Sec. 6.6, we point out the interactions within and between the agents.

Robots and motes know *a priori* the list of possible tasks that they can perform. Both have in common, for instance, "measurement of temperature", "video recording", but only robots can explore the environment or answer to help requests by other nodes.

Robots and motes have generally different sets of tasks. However, we used for our experiments four tasks, both for motes and robots. We refer to the set of tasks as $T_{\text{agent}} = \{\mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3, \mathrm{T}_4\}$. The robots' and motes' four tasks are described below.

Each agent k associates real number τ_i^k to each task i, with $i \in T_{\text{agent}}$. At the moment of selecting a task to perform, agent k chooses randomly

[5]It should also be pointed out that the current transmission rate of real motes is only of some kB/s.

between the tasks. The probability of choosing task i is

$$P(i) = \frac{(\tau_{\mathrm{i}}^{k})^{\beta_{\mathrm{task}}}}{\sum_{j \in T_{\mathrm{agent}}} (\tau_{\mathrm{j}}^{k})^{\beta_{\mathrm{task}}}} \ , \tag{6.1}$$

with $\beta_{\mathrm{task}} \geq 1$. This equation is similar to the one used to model how ants choose one path among those that bring to a food source [Bonabeau et al., 1999]. In the original formulation, τ_{i}^{k} was the concentration of pheromone on path i. The parameter β_{task} was introduced to increase the exploitation of good paths. If $\beta_{\mathrm{task}} = 1$, then $P(i)$ is proportional to τ_{i}^{k}. If $\beta_{\mathrm{task}} > 1$, then tasks with high τ_{i}^{k} have higher probability to be selected than in the previous case.

Agents have the capability to perceive whether their actions are successful or not, either by directly sensing the environment, or by receiving a feedback from other nodes (we discuss more of this in Sec. 6.5). The agent first initialises $\tau_{\mathrm{i}}^{k} = \tau_{\mathrm{init}}$, $\forall i \in T_{\mathrm{agent}}$. If the agent is successful in performing task i, then

$$\tau_{\mathrm{i}}^{k} = min\{\tau_{\mathrm{max}}, \tau_{\mathrm{i}}^{k} + \Delta\tau\} \tag{6.2}$$

and

$$\tau_{\mathrm{i}}^{k} = max\{\tau_{\mathrm{min}}, \tau_{\mathrm{i}}^{k} - \Delta\tau\} \tag{6.3}$$

if it is unsuccessful.

We use four tasks for the experiments in Sec. 6.9. For three of them, the behaviours of robots and motes are the same:

T_1. measure the temperature locally and send it to the rescue team headquarter;

T_2. record the sound in the surroundings and send it to the headquarter;

T_3. record a video of the place and send it to the headquarter.

Task T_1 is terminated immediately because it creates and sends only one small packet. Tasks T_2 and T_3 occupy the node for more time, because they generate a stream of packets, which is usually a big load for the network. T_2 differs from T_3 because it generates less packets than the latter.

Motes' and robots' behaviours are different in the case of the fourth task:

T_4. motes broadcast a help requests, robots answer to them and travel where they are needed.

Generally, motes might decide that they need help by analysing their data, or after instructions from the base. In our experiments, motes' help needs are modelled with a stochastic process. Robots listen for incoming help requests and answer to them by first travelling where it is required and then by completing the task (for instance, by taking care of a victim). It is assumed that the robots can perceive the direction and the distance of their neighbours, motes and robots. It is not the

Table 6.1: Type of packets generated by the application layer.

Type	Description
BASEHELLO	packet regularly generated from the base host to broadcast its address
TEMPERATURE	packet containing a temperature reading (task T_1)
SOUND	type associated to a stream of packets which contains the sound recorded by the agent (task T_2)
VIDEO	type associated to a stream of packets which contains the video recorded by the agent (task T_3)
HELPREQUEST	packet used during the execution of task T_4 for mote-robot co-ordination
ERROR	used by the network layer to signal an error to the application layer; it contains the information about the packet which generated the error

purpose of our work to address this issue, but it might be done, e.g., by triangulating the signal emitted by a node, by using a directional antenna, or by means of a vision system.

Task T_4 requires co-ordination between robots and motes. The agents exchange a number of messages between them to perform the task. The robot can use the route discovery capability of the network layer to find a path to its destination. In SN, the topology of the network mostly corresponds to the topology of the environment. The network topology can be seen then as a simple map of the environment, which the robot can exploit to travel through it.

All the tasks generate and send different types of packets. They are listed in Table 6.1.

6.4 Agents' Control: Network Layer

The communication range of robots and motes is not enough for them to send messages directly to every other agent in the environment. For this reason they need to relay messages to their neighbours, which then relay again to some other node, till the messages reach their destination. This process is called *routing*. The traffic generated by the nodes due to their activity is a uniform one. This means that we do not expect, *a priori*, bursts of packets to be always generated from a part of the area. Nodes might generate high traffic (e.g., when performing task T_3), but the probability that it occurs is generally uniformly distributed on the area.

We already discussed about routing protocols in Sec. 6.1. We chose to focus on *AntHocNet* [Di Caro et al., 2005] for our application. *An-*

tHocNet is a bio-inspired, self-organising, routing algorithm, inspired by the food searching behaviour of ants. There are two main reasons why we chose this algorithm: firstly, it is inspired by ants' behaviour and perfectly fits in the context of our work; second, and more important, Di Caro et al. showed that it performs better than AODV [Perkins et al., 2003, Perkins and Royer, 1999], the state of the art for routing algorithms in *ad hoc* networks.

AntHocNet is a hybrid multi-path algorithm. It is reactive in the sense that it does not keep up-to-date information on all possible destinations. It builds routes to a destination only when the upper layer wants to communicate with it. Once one or more paths have been found to the desired destination, *AntHocNet* proactively maintains them till the data session is over.

The routing information of node i is kept in a table $\mathcal{R}^i$. Each entry $\mathcal{R}^i_{nd} \in \mathbb{R}$ of the table represents an estimation of the goodness to go from node i to node d using node n as next hop. The higher $\mathcal{R}^i_{nd}$ is, the better the path to d via n is. $\mathcal{R}^i$ is called *pheromone* table, because its content simulates the pheromone traces that ants lay in the environment to find the shortest path to food sources.

When the application layer wants to send a message to a host for which there is no known path, *AntHocNet* starts a route discovery process. When one or more routes have been found, *AntHocNet* routes the data on the newly discovered paths. To describe *AntHocNet* we need to describe the two phases separately. We do this in Sec. 6.4.1 and Sec. 6.4.2. In Sec. 6.4.3, we describe our modifications and extensions to the original algorithm in order to fit it in our work with SANETs.

6.4.1 *AntHocNet*: Route Discovery

If a node wants to send packets to a destination for which it does not know any route, it broadcasts a route discovery packet, containing the address of the desired destination d. In the original formulation, this packet is called a *forward ant*. For consistency with the other types of packets we describe later, we call it ROUTEDISCOVERY. The request is treated by the other nodes as a normal data packet.

The packet is transmitted to the node's neighbours. They might know a route that reaches d, or not. If the node knows how to reach the destination, it randomly chooses a next hop n to relay the request to. The probability $\mathcal{P}^i_{nd}$ at node i to choose n as next hop to reach d is:

$$\mathcal{P}^i_{nd} = \frac{(\mathcal{R}^i_{nd})^{\beta_{\text{disc}}}}{\sum\limits_{j \in N^i_d} (\mathcal{R}^i_{jd})^{\beta_{\text{disc}}}} \ , \tag{6.4}$$

where N^i_d is the set of neighbours for which a path to d is known. β_{disc} is a parameter that can control the exploratory behaviour of the algorithm, in the same fashion as for task selection.

If a neighbour does not know anything about d, it broadcasts the incoming request again. Due to broadcasting, the discovery messages can proliferate quickly and follow different paths in the network.

The ROUTEDISCOVERY packet stores the path travelled so far. If a node receives several requests originating from the same node, it compares the path of each packet with the shortest known path, the distance being measured in number of hops. The packet is let through if the new incoming packet is not worse than a_1 times the shortest number of hops. This filter reduces the overhead by killing packets that have travelled over bad paths. There is a less restrictive acceptance factor a_2 in the case the first hop of an incoming packet is different from the first hop of the best packet. This strategy is used also in other works [Marina and Das, 2001] in order to obtain a uniformly spread set of paths.

AntHocNet assumes that the paths are symmetric: if node A can directly communicate with node B, then node B can directly communicate with node A. This is a reasonable assumption also in our set-up. On arrival to destination, the receiving node generates a ROUTEDISCOVERYRESPONSE packet (*backward ant* in the original description), which is sent back with high priority along the same path $\mathcal{P}$ of the incoming packet. Its travel back is used to estimate $\hat{T}_{\mathcal{P}}$, the time it would take for a packet to travel over $\mathcal{P}$ to the destination. The estimation is iterative:

$$\hat{T}_{\mathcal{P}} = \sum_{i \in \mathcal{P}} \hat{T}^i \ .$$

$\hat{T}^i$ is calculated by each node i as:

$$\hat{T}^i = (Q^i_{\text{MAC}} + 1)\hat{T}^i_{\text{MAC}} \ ,$$

where Q^i_{MAC} is the number of waiting messages at the MAC layer of node i, and $\hat{T}^i_{\text{MAC}}$ is the moving average waiting time at the MAC layer.

The ROUTEDISCOVERYRESPONSE sets up a path to destination d in each node $i \in \mathcal{P}$. Each node first calculates the value ρ^i_d to use to update the pheromone table:

$$\rho^i_d = \left(\frac{\hat{T}^i_d + hT_{\text{hop}}}{2} \right)^{-1} \ ,$$

where $\hat{T}^i_d$ is the estimated travelling time to go from i to d, h is the number of hops and T_{hop} is a parameter representing the time to take one hop in unloaded conditions. And finally,

$$\mathcal{R}^i_{nd} = \gamma \mathcal{R}^i_{nd} + (1-\gamma)\rho^i_d \ . \tag{6.5}$$

Once a node starts the route discovering process, it waits for a response for some time and buffers the data to send. If it did not receive any after some time (1 s in the original work), it starts the discovering process again. The node repeats the process for a maximum number of times (originally 3) before giving up.

6.4.2 *AntHocNet*: Routing

Once one or more routes have been found, *AntHocNet* can start the data session. For each packet, the node chooses randomly the next hop. The probability for each hop is calculated with (6.4), only using a different exponent, β_{rout}, higher than β_{disc}. The higher exponent results in a greedier behaviour with respect to good paths. The probabilistic data routing leads to data load spreading, relieving congested paths.

To maintain the routing information up-to-date, *AntHocNet* sends special packets, called *proactive forward ants*, every n data packets. Proactive forward ants are normally unicast, sent to the next hop as ROUTEDISCOVERY packets using pheromone values, but they also have a small probability p to be broadcast. These packets serve a double purpose: they probe and keep up-to-date existing paths, and they explore new paths that might have come into existence.

6.4.3 Modifications to *AntHocNet* and Additional Features

The most important extension in our implementation of *AntHocNet* is the definition of different message classes. Each class is associated to one task of the application layer, and each node keeps a different routing table for each class. We elaborate more on this point in Sec. 6.5.

Both *AntHocNet* and our modified version use HELLOMESSAGE packets, which are periodically broadcast from the nodes to their neighbours. After a node receives a HELLOMESSAGE, it expects to receive it regularly. If it does not occur within a given amount of time, the neighbour and all its associated routes are deleted from the routing table. In our case, the HELLOMESSAGE rates are different for robots and motes. The robots use a higher sending rate than the motes. The nodes use also two different timeouts in case the node is mobile (shorter timeout) or not (longer timeout). The information about the mobility of the nodes is contained in the HELLOMESSAGE. The list of all packet types used by the network layer is listed in Table 6.2.

If a node does not find a route to the destination (because there is no path, or because the discovery message was lost), it takes one of two actions according to who originated the message: if the message comes from the application layer, the network layer notifies it about the failure; if the message came from someone else, it sends a ROUTEFAILURE packet to the origin. The packet is used at the origin to assess the failure in performing a task.

In a SANET, most of the packets are likely to contain data coming from sensor readings. In order to reduce the congestion of the network, a node might decide, instead of routing a packet, to drop it. The decision is probabilistic and is described in Sec. 6.6.1.

We do not collect only statistics about the waiting time at the MAC layer, but also different other information, like the signal-to-noise ratio and the energy required for transmission. This allows the nodes to choose the next hop using several criteria. Every node might decide to route its data trying to maximise different objectives. It might choose, for instance, the route with higher minimal signal-to-noise ratio, increasing the reliability of the message delivery. The node might also choose to use different criterion in different moments. If it detects important information to be sent, it might decide to use a route that minimises the end-to-end delay, short routes with low traffic. If the node has low power level, it might decide to send the message to a very near node.[6]

Statistics about the paths are not stored only in the ROUTEDISCOVERYRESPONSE packets, but in all the packets that the network layer receives. In this way, other nodes can passively set up routes to other hosts when they receive a message from it. This avoids the need of reinforcing the paths during transmission, at the cost of slightly bigger network packets. The reader should remember that in our scenario we envisage the transmission of heavy data (images, sound, videos), thus a couple of bytes more should not influence much the performance of the SANET.

The different information about routes with respect to *AntHocNet* requires also a small change in the way the pheromone table is updated. The content of our routing tables is not single values, but tuples $\mathbf{R}^i_{nd} = \langle t, h, e, s, m, v \rangle$, where t is the estimation of the transmission time, h is the number of hops in the route, e is the energy required for transmission, s is the minimal signal-to-noise ratio of all the links in the path, $m \in \{\mathrm{true}, \mathrm{false}\}$ denotes if the next hop n is mobile, and $v \in \{\mathrm{true}, \mathrm{false}\}$ whether it is still valid for routing. $\mathbf{R}^i_{nd} \in \mathbf{R} = \mathbb{R}^3 \times \mathbb{N} \times \{\mathrm{true}, \mathrm{false}\}^2$.

At the arrival of a new packet with data about the route to d through n, we update the routing table using a custom function $\oplus$. If $\mathbf{r}^i_{nd}$ is the information obtained from the incoming packet,

$$\mathbf{R}^i_{nd} = \mathbf{R}^i_{nd} \oplus \mathbf{r}^i_{nd}$$

performs a weighted sum of the real values t, h, e, s (like (6.5)) and sets m to the new value. All occurs only if both the previous information and the new information are valid.[7]

To find the next hop for routing, we use a function $r : \mathbf{R} \to \mathbb{R}^+$ and calculate the probability of using n as next hop to d with

$$\mathcal{P}^i_{nd} = \frac{r(\mathbf{R}^i_{nd})^\beta}{\sum\limits_{j \in N^i_d} r(\mathbf{R}^i_{jd})^\beta} \,. \tag{6.6}$$

The higher $r(\mathbf{R}^i_{nd})$ is, the more likely node n will be chosen as next hop. To change the strategy of routing, a node would simply change the function $r(\,\cdot\,)$. If the node wants to be sure that a packet reaches its destination, it might choose routes with good signal-to-noise ratio, and use $r(\mathbf{R}^i_{nd}) = s$ (s is the estimated signal-to-noise ratio contained in

[6]The distance of a node can be estimated by the receiving power of the message, as measured by the antenna, that is, by the physical layer. It should be noted however that it is not really important in this case to know the real distance, but the power required for the transmission.

[7]In our case it can not happen that motes become mobile, and thus changing m might seem useless. It can happen, in applications different from our scenario and experiments, that some robot decides to "become" a mote, that is, not to move. The robot could decide it because the network in that point is particularly under load, or because there was a failure of one mote and the network lost connectivity.

Table 6.2: Type of packets generated by the network layer.

Type	Description
HELLOMESSAGE	generated and broadcast by every node to inform the neighbours about one node's presence; it contains information about node's mobility
ROUTEDISCOVERY	used to discover routes to a specified destination
ROUTEDISCOVERYRESPONSE	used by nodes to reply to ROUTEDISCOVERY requests
ROUTEFAILURE	generated when no route for a destination is found or when a node rejects and incoming packet
DATA	used at the network layer to transport data from the application layer

$\mathbf{R}^i_{nd}$). If the node prefers that the message arrives as soon as possible, it might choose routes with short travel time and use $r(\mathbf{R}^i_{nd}) = \frac{1}{t}$ (t is the estimated transmission time).

It is not our purpose in these pages to test different routing strategies, thus $r(\cdot)$ is mostly unchanged in our experiments (there is just one exception, explained later). The actual function is:

$$r_{\text{time}}(\mathbf{R}^i_{nd}) = \begin{cases} \frac{1}{t} & \text{if } t \neq 0, \\ H & \text{if } t = 0. \end{cases},$$

$$r(\mathbf{R}^i_{nd}) = \begin{cases} 0 & \text{if } n \text{ is not a valid hop}, \\ H & \text{if } n \text{ is a valid hop and } n = d, \\ \frac{r_{\text{time}}(\mathbf{R}^i_{nd})}{2} & \text{if } n \text{ is a valid hop, } n \neq d \\ & \text{and } n \text{ is a mobile hop}, \\ r_{\text{time}}(\mathbf{R}^i_{nd}) & \text{otherwise.} \end{cases},$$

where H is a high-value constant. As it can be seen, this function increases the probability to route packets through nodes which are not mobile, i.e., the motes. This is because a link to a mobile node is likely to break soon, while the motes can form a sort of stable backbone.

If a node is not able to find a route to the destination, it generates a ROUTEFAILURE packet which is sent to the source of the packet. If the source is the node itself, the network layer sends a ERROR packet to the application layer.

6.5 Agents' Control: Cross-layer Interactions

The application layer interacts with the network layer by two means.

1. Every message from the application layer is associated to a task. The network layer uses different routing tables for each task.

When it receives a message from the application layer, it routes the packet using the information contained in the table associated to the message's task. This means that the node starts a new route discovery for the destination if there is no known route for the task, even though routes are known for other tasks.

2. When performing task T_4 for driving a robot, the application layer uses the information stored in the routing table in order to reach its destination. The route is evaluated on the base of the hop count contained in the routing table (Sec. 6.6.2) and the validity of neighbours. The application layer modifies the content of the routing table, by invalidating the nodes that the robot has reached. In this way the robot can proceed to the following node in the path.

Different tables might be useful to have different routes emerging for different tasks. The probabilistic routing mechanism coupled with competition among packets for the routes might be enough to autonomously choose different routes for different packets. If there were more than one possible destination, like in the case of more than one rescue team base, these mechanisms would be sufficient for *anycast* communication[8] in the SANET. Although we designed our SANET with this idea in mind, it is not our purpose in these pages to investigate this topic further.

The network layer is mostly responsible to assess the success or the failure of tasks T_1, T_2 and T_3. It is the network layer that informs the application layer to update the task table.

The application layer considers the above tasks successful by default. The update for success (6.2) takes place immediately after the task has been selected. In the case the network layer received a ROUTE-FAILURE packet related to the task, the application updates the task table using (6.3), the update for failure. It might seem that this update schema is unbalanced toward reinforcing the tasks, and that τ_i^k for task i can never become less than τ_{init}. We show in the next section that there are other cases in which the update for failure takes place.

6.6 Agents' Control: Inter-agent Interactions

One of the main interactions between the agents in our SANET occurs at the network layer. A node, upon arrival of a packet to route, might decide to drop it and then inform the source. We describe this mechanism in the following section. Section 6.6.2 illustrates the second important interaction: the robot-mote co-ordination in case of help requests (task T_4).

[8]In anycast communication, the destination is neither one node (unicast), nor all the nodes (broadcast), nor a group of nodes (multicast), but one of a group of node (it does not matter which one).

6.6.1 Packet Filtering

The rejection of others' messages plays an important role in the division of labour. It is the source of the negative feedback which is required by the agents to specialise. Agents do not need to signal to their neighbours the task they are currently executing, because their output is likely to be read anyway by nearby agents because of routing. This is why it is reasonable to directly use this 'free' source of information as base mechanism for agents' adaptation.

To understand better this mechanism, we can make a parallel with the prey retrieval task shown in Chap. 4. The robots compete among themselves to retrieve prey. If one robot is successful, the other robots have lower probability to retrieve a prey, because of the reduced number of prey. The more successful a robot is, the more it will repeat the successful task. In our SANET application, the agents compete to send a message to a common destination. An agent that can send a message is considered successful and increases the probability to repeat the same action (that is, to send the same type of message), while its neighbours' probability decreases. The opposite occurs when the agent's packets are rejected, according to the mechanism described below.

Each node remembers the last packet it received of a given type and for a given destination.[9] Upon arrival of a new packet to route, each node compares it with the one previously stored. It then probabilistically decides whether the packet should be routed or not. In case the node decides to route the packet, it increases the probability to route it again later and decreases the τ_{i}^{k} related to the packet type. If the node rejects the packet, it decreases the probability to route it later, increases its τ_{i}^{k} and sends a ROUTEFAILURE message to the source.

This mechanism does not take place in the following cases:

1. the packet is broadcast (it might be some important message to spread in the network);
2. the packet is not the first packet of a stream generated by T_2 or T_3 (streams are interrupted at the beginning, but not when the connection with the destination has already been established);
3. it is a packet belonging to T_4 (this task requires a strict co-ordination between robots and motes, thus it should not be interrupted);
4. the packet comes from a source further than a given hop-count threshold D (packets from near sources have correlated content, and thus they can be dropped without losing much information).

Each node keeps a table $\mathcal{Q}$ of values $\mathcal{Q}_d^i \in [\mathcal{Q}_{\min}, \mathcal{Q}_{\max}]$ for known destinations d and packet class i. The probability $\mathcal{P}_d^i$ to route a packet

[9]This implementation might require much memory and might not scale well. Other solutions can be used on devices with limited memory. We could use a limited array for recording the last messages. If the array is full and a new message should be stored, then the oldest element can be deleted. This system would have only the effect to weaken the interaction between nodes, which could however be compensated by some other means.

is

$$\mathcal{P}_d^i = \begin{cases} \mathcal{Q}_d^i & \text{if this is the first packet} \\ & \text{of class } i \text{ for } d \text{ seen, or} \\ \mathcal{Q}_{\min} + \alpha_1\alpha_2(\mathcal{Q}_d^i - \mathcal{Q}_{\min}) & \text{otherwise.} \end{cases} ,$$

where

$$\alpha_1 = (1 - e^{-\frac{5h}{D}}) , \qquad \alpha_2 = (1 - e^{-\gamma_1 \Delta t}) ,$$

h is the number of hop travelled from the incoming packet, Δt is the elapsed time from the previous known message, $0 < \mathcal{Q}_{\min} < \mathcal{Q}_{\max} \leq 1$, $\gamma_1 > 0$. The coefficients α_1 and α_2 decrease the probability to route a packet for nearer sources and for similar information recently transmitted. It it known that $\alpha_1 \approx 1$ when the exponent $\frac{5h}{D} \approx 5$, that is, $h \approx D$. Therefore, the effect of α_1 smoothly decreases for h approaching the threshold D.

Every $\mathcal{Q}_d^i$ is first initialised to $\mathcal{Q}_{\text{init}}$. If a node decides to reject a packet, it updates $\mathcal{Q}_d^i$ using

$$\mathcal{Q}_d^i = max\{\mathcal{Q}_d^i - \Delta\mathcal{Q}, \mathcal{Q}_{\min}\} ,$$

and uses

$$\mathcal{Q}_d^i = min\{\mathcal{Q}_d^i + \Delta\mathcal{Q}, \mathcal{Q}_{\max}\} ,$$

if it decides to route the packet. Additionally, if the node rejects a packet of class i, it increases its own τ_{i}^k with (6.2), and decreases it with (6.3) for each packet that is routed.

Note that the threshold D grants that this interaction between agents is localised, that is, does not involve all the agents in the area.

6.6.2 Help-request Behaviours

In the scenario described in Sec. 6.2, motes can detect whether they require additional help from robots. In a real application, the mote could decide this either by analysing the sensor data, or by direct instruction from the rescue team.

When a mote's application layer chooses to perform task T_4, it first checks if there are any local help requests waiting to be fulfilled. The controller then works as represented in Fig. 6.1:

start If there is no need to call for help, the mote considers it as a failure, and thus updates $\tau_{\mathrm{T}_4}^k$[10] accordingly using (6.3). In the opposite case, it broadcasts a help request, and updates $\tau_{\mathrm{T}_4}^k$ for a success.

request sent The mote waits for some robots to reply. The mote receives usually many packets from the robots when it is in this state. Most of the packets are only used by the robots to estimate the distance from the mote (see later, when we discuss the robots' behaviour) and do not contain an acceptance of the task. The first robot that sends an acceptance packet is assigned to the task. If

[10]According to our enumeration of the task listed in Sec. 6.3 and to the convention used in the same section, 4 is the number associated to the help-request task.

the mote does not receive a positive answer before MT_{req} seconds, it broadcasts the request again. It repeats this for a maximum of M_{retr} times and then gives up. Note that in this case the mote does not decrease $\tau^k_{\text{T}_4}$.

waiting for robot The mote waits for the robot that was assigned the task. During this period, the robot is travelling the network to reach its destination. It regularly sends messages to the mote which are both used as 'keep alive' messages and to update the robot's routing table. These messages contain also the expected maximum time the robot requires to travel one hop. If the mote does not receive again a message from the robot within this time, the mote 'drops' the robot, broadcasts again the request and returns to the **request sent** state. Upon arrival, the robot sends a message to signal it is on the place, and the mote considers the request fulfilled. If the mote receives a message from other robots accepting the request while the task is still assigned to the previous robot, the mote sends back a refusal packet (the actual communication protocol is described below).

end The mote selects another task to perform.

The robots store every help request they receive in a temporary buffer. They increase their $\tau^k_{\text{T}_4}$ for each incoming packet in order to improve the likelihood of robots to perform the task. The reason is that it makes sense in applications like our scenario to push the robots to perform more a task which requires mobility than to 'waste' them for tasks which could be done by the motes. Upon arrival, the robots always send a message to the mote. The purpose of this message is to activate the route discovery process of the network layer in order to obtain the estimated number of hops required to reach the destination.

When the application layer selects the help-request task, the control system of the robot works as follows (Fig. 6.2):

start The robot checks whether there is any request pending.

select destination The robot probabilistically chooses one host among the set HD of possible destinations. The probability for each host $d \in HD$ is given by:

$$\mathcal{P}(d) = \frac{h_d^{-0.5}}{\sum\limits_{i \in HD} h_i^{-0.5}} \;,$$

where h_j is the distance in number of hops of host j, obtained from the routing table. The exponent -0.5 favours near destinations, but leaves a fair chance to further destinations to be chosen.

ask destination The robot informs the destination that it is willing to take the request. The mote decides whether the robot can continue or not. The mote may reject because the request was already fulfilled, because another robot is working on it, or because the mote is currently busy with another task. If the robot does not

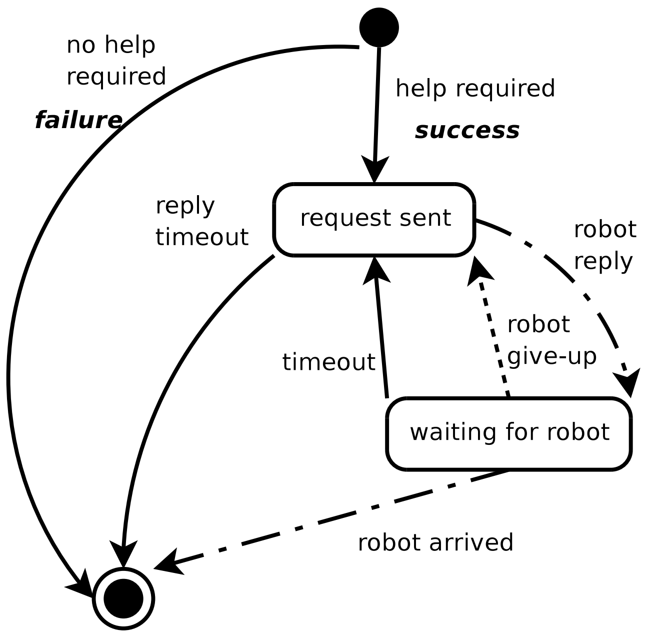


Figure 6.1: Motes' behaviour for help-request task. The dash-dotted arrow represents a transition that occurs thanks to an incoming packet, in this case a robot that answers the mote's request or that signals its arrival. The dotted arrow stands for an incoming packet from the robot that gave up the task. Continuous-line arrows are internal events that the mote evaluates at each control step. See the text for the description of the states.

receive an answer after RT_{req} seconds, it sends the request again for a maximum of R_{retr} times, then it gives up.

find next hop The robot looks into the routing table to select the next hop of the route to its destination. The next hop is used as described in Sec. 6.4.3, but using:

$$r(\mathbf{R}^i_{nd}) = \begin{cases} 0 & \text{if } n \text{ is not valid,} \\ H & \text{if } n \text{ is valid and } n = d, \\ \frac{1}{h} & \text{otherwise.} \end{cases} .$$

H is the same constant used for packet routing (see Sec. 6.4.3). The different function $r(\cdot)$ used in this case accounts for the fact that robots are not like packets that flow in the network. The function used for packets takes into account the transmission time, which involves the waiting time at the MAC layer, the congestion of the network, and so on. Robots care mainly about the physical distance, which is approximated by the hop count h.

request next hop If no next hop is found, the robot sends a message to the destination and waits for a reply. This message is used to start the route discovery process at the network layer. As in **ask destination**, the robot waits RT_{req} seconds before sending another request, for a maximum of R_{retr} times, then it gives up.

go to next hop The robot proceeds towards the next hop. When it reaches a distance RD_{appr}, it invalidates the hop's entry in the routing table, and sends a message to the destination, in order to start a new route discovery process. When it reaches the distance RD_{reach}, it considers the hop reached and searches for a new one. If the previous message did not get lost, the routing table should already contain the information to find immediately the new hop. If the robot has been trying to reach the hop for more than RT_{trav}, it gives up.

go to destination In this state, the robot behaves mostly as in **go to next hop**, only the robot does not need to send a message to the destination when it is at RD_{appr} from it. When the robot is at less that RD_{reach} from destination, it signals that it has arrived, and updates its $\tau^k_{\mathrm{T}_4}$ for a success.

failure The robot is in this state after a rejection from the mote or a timeout. It updates $\tau^k_{\mathrm{T}_4}$ for a failure.

end The robot ends the execution of this task and the application layer chooses a new one.

The messages exchanged between robots and motes consist of three Boolean fields: `ra`, `ma` and `a`. The robots set `ra` (it stands for "`r`obot `a`cknowledgement") to `true` when they ask to be assigned the help request, or when they send messages to find the next hops in the route. Motes set `ma` ("`m`ote `a`cknowledgement") to `true` to inform the robots

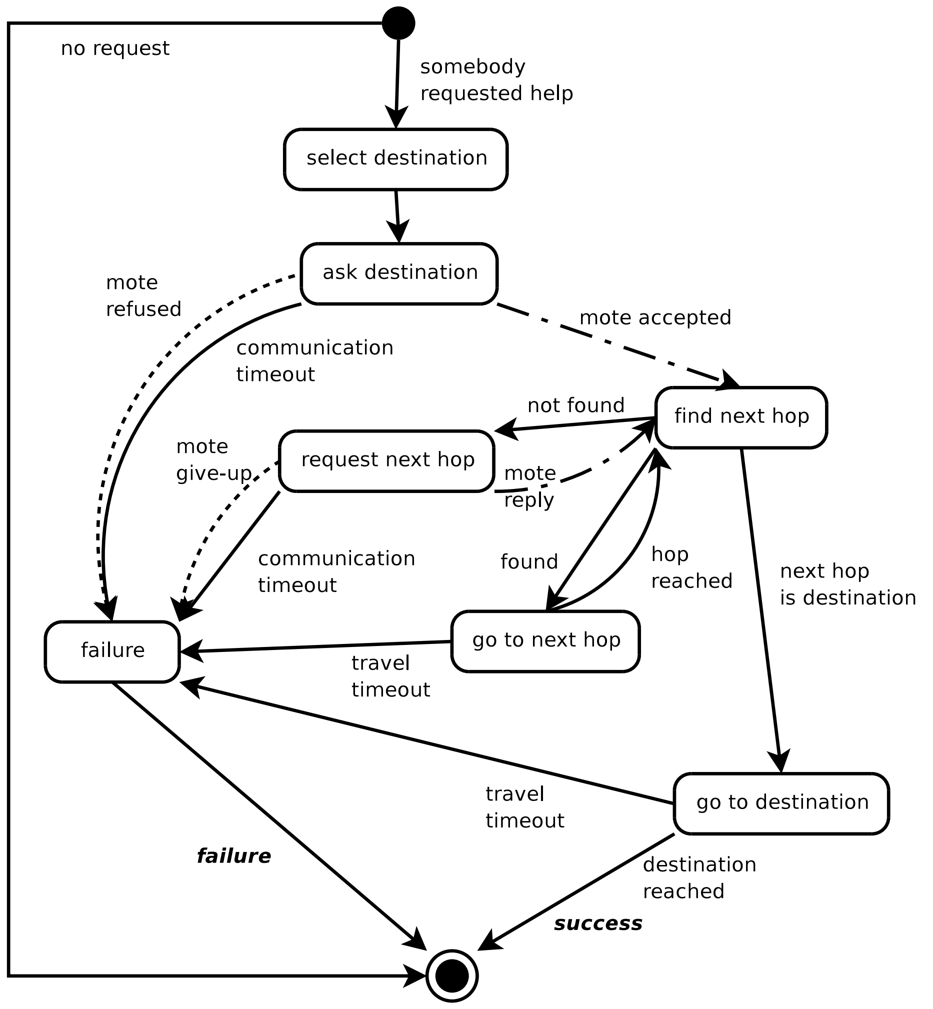


Figure 6.2: Robots' behaviour for help-request task. The dash-dotted arrow represents a transition that occurs thanks to an incoming packet, in this case a robot that answers the mote's request or that signals its arrival. The dotted arrow stands for an incoming packet from the robot that gave up the task. Continuous-line arrows are internal events that the mote evaluates at each control step. See the text for the description of the states.

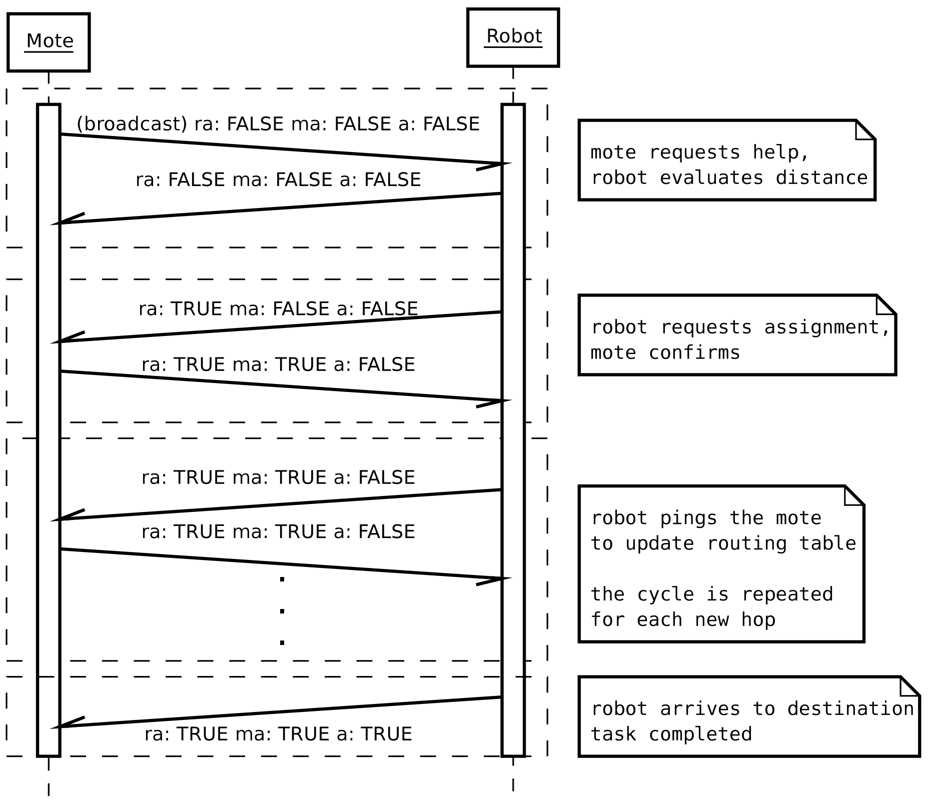

Mote
Robot
(broadcast) ra: FALSE ma: FALSE a: FALSE
ra: FALSE ma: FALSE a: FALSE
mote requests help, robot evaluates distance
ra: TRUE ma: FALSE a: FALSE
ra: TRUE ma: TRUE a: FALSE
robot requests assignment, mote confirms
ra: TRUE ma: TRUE a: FALSE
ra: TRUE ma: TRUE a: FALSE
robot pings the mote to update routing table
the cycle is repeated for each new hop
ra: TRUE ma: TRUE a: TRUE
robot arrives to destination task completed

Figure 6.3: Protocol for mote-robot communication: normal case. This graph represents the normal communication between a robot and a mote for the achievement of task T_4. The mote is represented by the left bar, the robot by the right bar. Time flows downward. Arrows represent the messages exchanged between the agents. The text above each arrow describes which of the three Boolean fields of the message are set in the message.

that they are in charge of the request. Motes set it to `false` if the mote gives up the task or if other robots ask for a request already assigned. Finally, robots set a ("arrived") to true to inform the motes that they arrived to destination. Figure 6.3 summarises the communication between a robot and a mote in the normal case, that is, where no rejection takes place. Figure 6.4 shows how robots and motes communicate failures and timeouts.

6.7 Experiments: Set-up

We run experiments only in simulation, as we explained in the introduction. We simulated SANETs in a rectangular area, whose side is 500 m. Twenty-five motes are placed in a grid that covers the environment. This is a likely placement for the scenario of Sec. 6.2. To

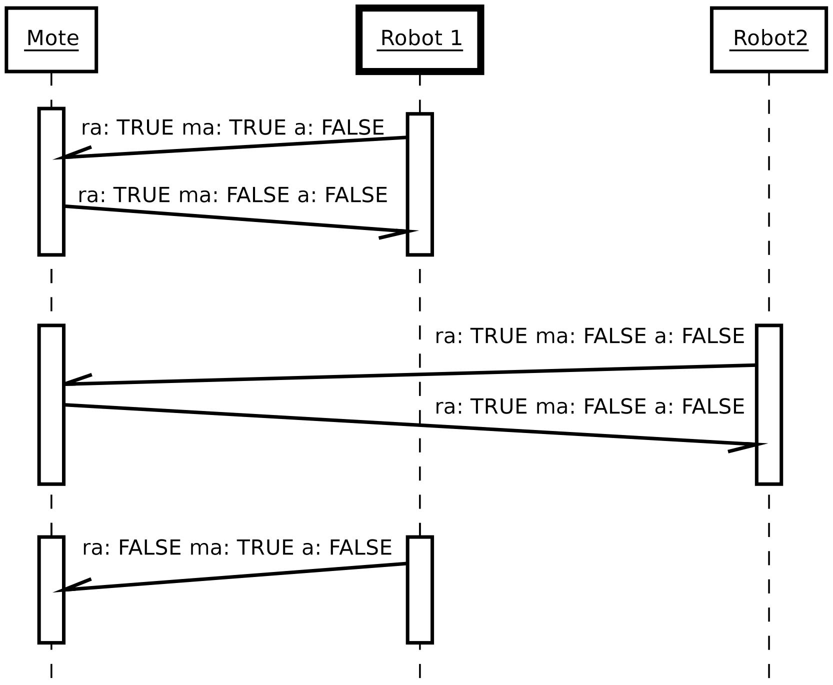

Mote
Robot 1
Robot2
ra: TRUE ma: TRUE a: FALSE
ra: TRUE ma: FALSE a: FALSE
ra: TRUE ma: FALSE a: FALSE
ra: TRUE ma: FALSE a: FALSE
ra: FALSE ma: TRUE a: FALSE

Figure 6.4: Protocol for mote-robot communication: error messages. The meaning of the bars and of the arrows is as described in Fig. 6.3. The robot in the middle, with a bold frame, is the one that previously negotiated and obtained the assignment of the request. The first block of messages, at the top, describes the case in which the mote gives up the task, and signals it to the robot. The second case describes the situation when another robot is asking for the assignment of the request but this was already given to Robot 1. Finally, the bottom block illustrates how the robot signals to the mote that it gives up the task.

simulate however a real deployment process, the motes are randomly placed in an area 5 m around the actual grid point. Robots are placed at the corners of the environment. We tested 1, 2 and 3 robots at each corner, for total group sizes of 4, 8 and 12 robots. Figure 6.5 shows the set-up, from the point of view both of the network and of the simulated world.

Three of the tasks in which the agents are occupied involve the transmission of the data to a common destination: the rescue team base, according to our scenario, or the *sink*, according to the networking jargon. The base is placed in one of the corner of the environment (top left corner of Fig. 6.5(a)). The area is thus symmetrical along one of the diagonals.

Robots and motes do not know the address of the base. The application layer of the base broadcasts regularly a packet which contains its address. The agents start working only after the arrival of this message. Given the importance of this information, it is replicated and broadcast also by each node's HELLOMESSAGE, generated by the network layer. Apart from broadcasting its address, the application layer of the base only receives and records the messages it received. The network layer of the base is the same as the other nodes.

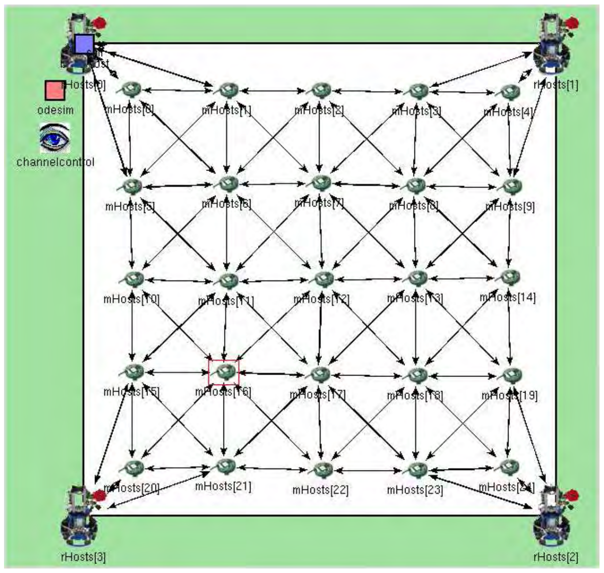
The help requests are randomly generated by the motes, with a variable probability per second, called *help density*. For the experiments, we used help density $12.5\ 10^{-6}\text{s}^{-1}$, $25\ 10^{-6}\text{s}^{-1}$ and $50\ 10^{-6}\text{s}^{-1}$. Each combination of robot group size/help density was tested in forty different runs. Each run is described by a seed for the random number generator and the misplacement of the motes. Another random number generator, initialised with a different seed, is used to generate the help-request events during the simulation.

The values of the agents' parameters are summarised in Table 6.3.

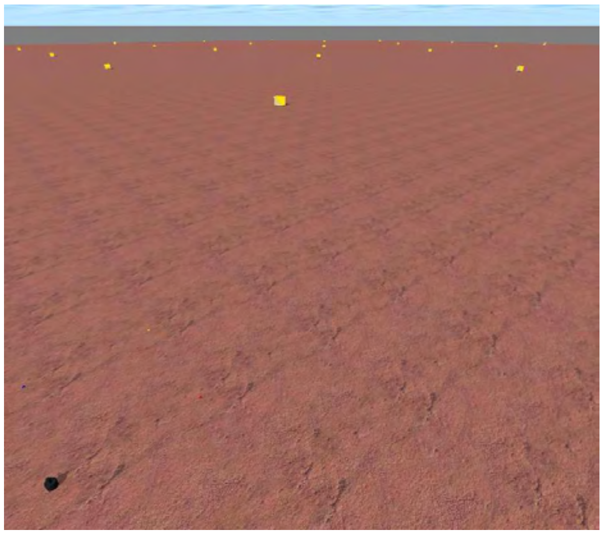
6.8 On the Measurement of the Division of Labour in SANETs

Before showing the actual results of our simulations, we have to spend a few words to introduce how we measure the division of labour in the context of SANETs.

We could just show the distributions of the probability of performing each task among the agents, as we did in the previous chapters (e.g., Fig. 4.14 and Fig. 5.8). This information is not enough for SANETs because it does not consider the position of the agents. Take for instance Fig. 6.6. The two figures show two cases with the same distribution among the agents, but with two different spatial distribution. In Fig. 6.6(a), the agents actively performing the task are separated, and thus the task is distributed over the environment. If the task were 'temperature reading', the base host would receive in this case information about unrelated zones of the environment. In Fig. 6.6(b), the agents are neighbours: the base host would receive redundant information about one area, and nothing about the rest of the environment.



(a) Network view



(b) Simulated world

Figure 6.5: Set-up of the experiments. Views of the network (a) and of the simulated world (b). (a) Motes are placed on a grid that covers the environment. Robots, in this case four, are placed on the corners. The base host, the one which receives the information that the agents collect, is on the top left corner (over the icon of the robot). The purpose of the other objects (*odesim* and *channelControl*) is described in Sec. 3.3.2. (b) The real dimension of the motes, usually few centimetres, was increased to make them visible. By comparison, the robot in the bottom left corner in the simulated world view is 42 cm tall.

Table 6.3: Parameter values of robots and motes.

(a) Network layer

Parameter	Value
β_{disc}	1.0
β_{rout}	2.0
a_1	0.4
a_2	0.9
γ_2	0.01 s^{-1}
$\mathcal{Q}_{\text{min}}$	0.01
$\mathcal{Q}_{\text{max}}$	1.0
$\mathcal{Q}_{\text{init}}$	1.0
$\Delta\mathcal{Q}$	0.02
hello period - robots	30 s
hello period - motes	120 s
hello timeout - robots	180 s
hello timeout - motes	600 s
D	2

(b) Application layer

Parameter	Value
N_{robot}	4
N_{mote}	4
β_{task} 3.0	
τ_{min}	0.1
τ_{max}	10
$\Delta\tau$	0.5
τ_{init}	3.0
MT_{req}	30 s
M_{retr}	3
RT_{req}	30 s
R_{retr}	3
RT_{trav}	300 s
RD_{appr}	10 m
RD_{reach}	4 m
sound stream size	40 kB
video stream size	400 kB

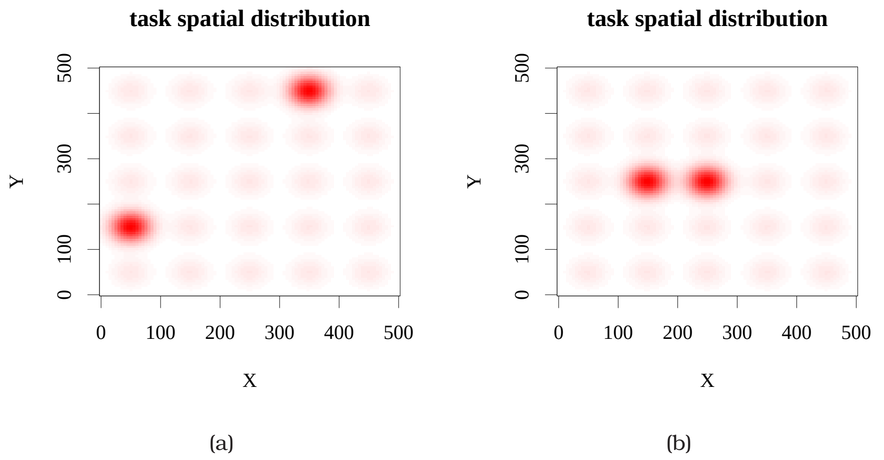


Figure 6.6: Examples of task distribution among the agents. The two plots show the distribution of the probability to perform a task among the agents. Agents are distributed on a grid that covers the environment and are represented as circles. The darker and the bigger the circle, the higher the probability for that agent to perform the task. In both graphs, most agents have low probability associated to the task, except for two of them. Thus, the distribution among the agents of the probability of performing a task is the same in the two cases. On the left side, the agents are far from each other. On the right side, the two agents are neighbours, and thus the task is concentrated only in one part of the environment.

To overcome this problem, we use the *hierarchic social entropy* of the group. The hierarchic social entropy is a measure of diversity first introduced by Balch [2000]. It allows to estimate how heterogeneous a system is, and also to compare the level of heterogeneity of different systems. In the following we describe how we calculate the hierarchic social entropy for our system.

6.8.1 Hierarchic Social Entropy

The first step for the measurement of the hierarchic social entropy is to cluster neighbouring agents that have the same probability distribution on the tasks. We represent each agent as a point in a $(N_{\texttt{agent}} + 1)$-dimensional space. The co-ordinates of the points are the probabilities for each agent to perform each task except one,[11] and its x and y positions in the environment. The co-ordinates are all normalised so that the points lie in the hypercube with side 1.[12]

The cluster algorithm works simply by joining those points which are no more distant than a given parameter d. The distance is usually the Euclidean distance. It is obviously critical to choose a good value for d. If $d = 0$, the number of clusters C is equal to the number of agents A. If $d = \sqrt{N_{\texttt{agent}} + 1}$, there would be only one cluster that includes the whole hypercube. For this reason, it is common use to show how C changes with d, thus $C = C(d)$, and which agents are clustered together for a given d. This is well summarised by a *dendrogram*, as those shown in Fig. 6.7.

After the clustering algorithm has been applied to the agents of our SANET, a single cluster includes those agents which are "alike". Agents in a cluster have similar probabilities to perform each task and are physically near. Two agents with the same distribution of probabilities but physically away from each other, or two agents near but with totally different probabilities require high values of d to be clustered. A good division of labour algorithm for a SANET will show a dendrogram like Fig. 6.7(b), a bad one will look like Fig. 6.7(a).

For a given d, we know the number of clusters $C(d)$ and the number $I(i, d)$ of agents included in each cluster $i \in \{1, 2, \ldots C(d)\}$. Picking up an agent randomly, the probability that it belongs to cluster i is then $p_i = \frac{I(i,d)}{A}$. We can measure then the diversity of the group using Shannon's *information entropy* [Shannon, 1949]:

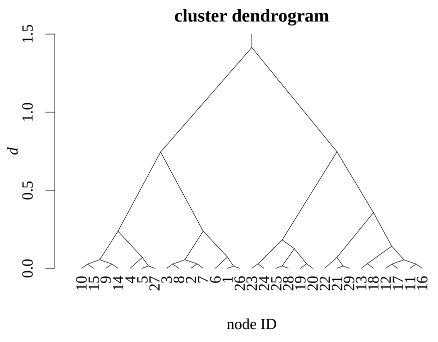
$$H = -K \sum_{i=1}^{C(d)} p_i \log_2 p_i \tag{6.7}$$

where K is a constant. It merely amounts to the choice of a unit of measure, thus it is commonly set to 1.

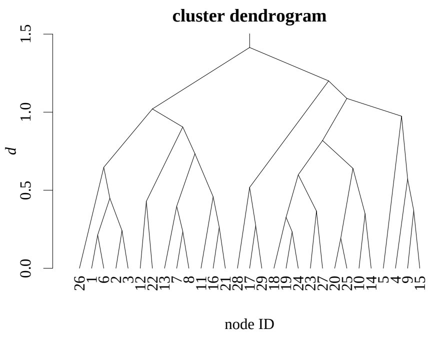
The information entropy H has several important features. For our purpose, we point out that $H \geq 0$, and $H = 0$ only if $\exists i : p_i = 1$, that

[11] The probabilities are obviously not independent, since they all sum to 1.

[12] The probabilities are, by definition, already in $[0, 1]$. The x and y co-ordinates of the agents are divided by the size of the environment.



(a)



(b)

Figure 6.7: Examples of dendrograms. The y axis represents the clustering parameter d. The agents (robots and motes) lay on the x axis. Two agents are connected to a node which is at the minimum height required to cluster the agents together. This is repeated recursively for each new formed cluster. A horizontal line at height d crosses the tree in a number of points equal to the number of clusters present for that particular value of d. The left picture shows a group of agents which are very similar, and thus can immediately be clustered with small value for d. The right plot shows a group of heterogeneous agents, which require higher d to be clustered together.

is, if there is only one cluster. H is maximised if $p_j = p_k \ \forall j, k$, that is, if $p_i = \frac{1}{C(d)}$, and in this case H is monotonically increasing with $C(d)$. Given that the number of clusters decreases with d, we can deduce that H is a monotonically decreasing function of d.

Balch [2000] defines *hierarchic social entropy* as:

$$S = \int_0^\infty H(d)\ dd = -\int_0^\infty \sum_{i=1}^{C(d)} p_i \log_2 p_i \ dd \ . \tag{6.8}$$

The usefulness of the hierarchic social entropy is clearly illustrated in Fig. 6.8: the more diverse a group, the higher its hierarchic social entropy. The hierarchic social entropy is thus a valid means to measure the level of specialisation in a group of robots. Its strength is in the fact that it can include both spatial information and probability distribution over the tasks.

It is not enough, though. Let us consider the case in which all the robots have high probability of performing T_4. In this case the hierarchic social entropy decreases, although this might not be considered a bad situation. It is reasonable that most mobile agents perform this task, because it is the main reason why robot were introduced in an SN in the first place.

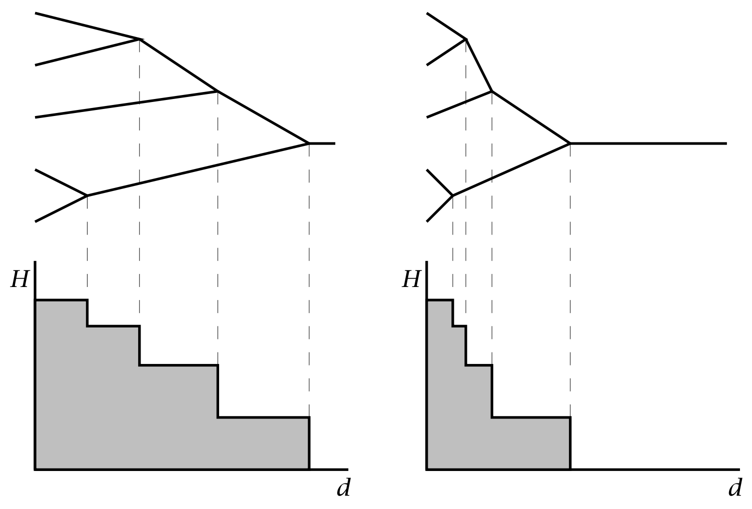


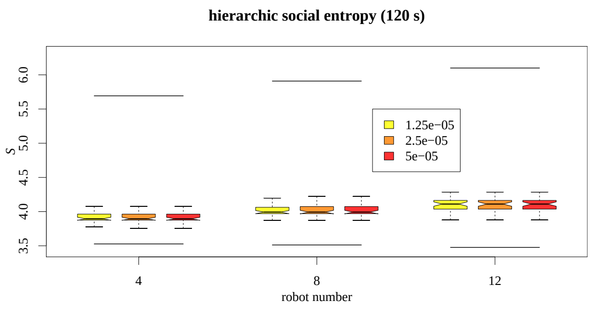
Figure 6.8: Relationship between dendrograms and hierarchic social entropy. The dendrogram on the left refers to a more heterogeneous group than the one on the right. The bottom plots show how the entropy H changes with the clustering parameters d. The area beneath the curve is the hierarchic social entropy. The later the clustering of two nodes occurs, that is, the more heterogeneous a group is, the bigger is the hierarchic social entropy.

6.9 Results

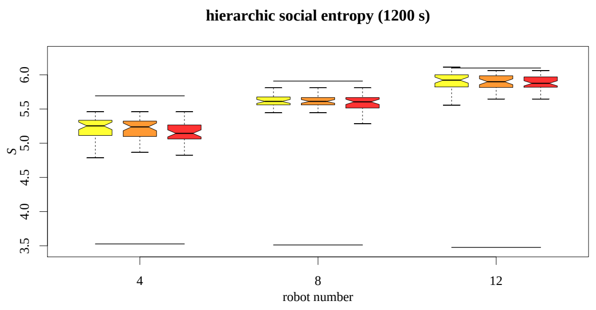
The results of the forty replications for each pair robot number/help density are summarised in Fig. 6.9. The plots show the distribution of the hierarchic social entropy at different time steps. The bottom and upper line in the plots represent approximated lower and upper bounds. The lower bound corresponds to the hierarchic social entropy calculated at the beginning of an experiment, that is, when the robots and the motes have uniform probability of executing each task. The only component which is taken into consideration when clustering is thus the physical distance between the nodes, since it is the only source of diversity.

The upper bound was estimated by finding the combination of robot positions and agents' task probabilities that maximises S. During the search for the maximum, the position of the motes was never changed from the initial one. Given the complexity of the function S and the dimensions of the search space, we could not afford to find the optimum. We calculated the local maxima reached from a random initial position by an optimisation algorithm. We repeated the process for several initial positions. The upper lines in Fig. 6.9 are the medians of the results. The lines are therefore not real upper bounds, but help the reader to understand how well our SANET is able to differentiate its members.

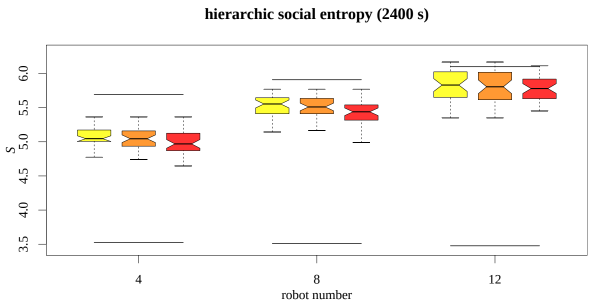
Figure 6.9 shows the value of hierarchic social entropy for different time steps. We see that already after 120 s the agents are able to



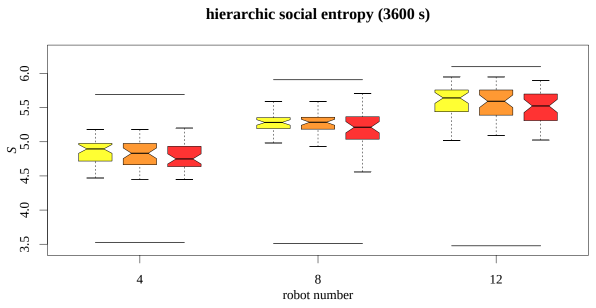
(a) 120 s



(b) 1200 s



(c) 2400 s



(d) 3600 s

Figure 6.9: Hierarchic social entropy of the SANET. The three graphs show snapshots at different time steps of the distribution of values of S, the hierarchic social entropy, for different group sizes and help densities. S is plotted on the y axis. On the x axis there is the number of robots used. Each bar in the triplet corresponding to a number of robots refers to different help densities. The meaning of the boxes is as in Fig. 4.7. The lines at the top and at the bottom represent approximated upper and lower bounds for the entropy.

differentiate. At 1200 s, the agents reach the maximum value of hierarchic social entropy, which they keep till approximately 2400 s. The hierarchic social entropy slowly decreases afterwards, till the situation depicted for 3600 s.

The hierarchic social entropy can increase because the agents are more apart in the environment, because they have different task probabilities, or a combination of both. Given that most of the agents are motes, and that the robots start from outside and move inward the network, the physical distance component can only be reduced. Therefore the hierarchic social entropy increases because of a more heterogeneous distribution of probabilities.

The help density has practically no influence on the hierarchic social entropy. This is in fact not a surprise. Probably, higher help density would decrease the hierarchic social entropy. High help density would push more and more agents to perform task T_4. Their task probability would be thus very similar, and the only source of differentiation would be the position in the environment.

The distribution of task probabilities among the agents can be seen from Fig. 6.10 to Fig. 6.12. Except for the distributions regarding T_4, most of the plots show that agents have probabilities spread all over the $[0, 1]$ interval. Nevertheless, all distributions show a multimodal profile, T_4 included. This is a sign that the agents are distributing the tasks among them. The peaks of the distributions might be due to a few clusters of agents placed only in particular areas of the environment. The results about the hierarchic social entropy however reassure us that this is not the case, and that agents with the same (high) probability to perform a task are also spatially separated.

This last claim is supported also by the observation of the evolution of the task probability distributions over time (from Fig. 6.13 to Fig. 6.15). What struck our attention is that, while the hierarchic social entropy does not change between 1200 s and 2400 s, the distributions still do. There are new peaks arising (corresponding to the stripes for $P \in (0.4, 0.5]$ that become darker and darker in Fig. 6.13 to Fig. 6.15) even after 1500 s. If more and more agents get similar probability of performing a task, they become more “alike” and thus the hierarchic social entropy should decrease. This does not occur only in the case the nodes become more “alike” in the task probability space but they are further away in the physical space. It is like the nodes move on a hypersphere in the clustering space. That is, if two nodes have the same high probability of performing a task, they are far one from the other in order to keep the same distance in the clustering space.

It is also possible to note that more and more agents set their probability of performing a task next to the right peak value. This is shown by the stripes that become darker and darker beside the main one ($P \in (0.3, 0.4]$ and $P \in (0.5, 0.6]$), starting at about 2300 s. We think that this phenomenon could explain the slow decrease of the hierarchic social entropy at the end of the experiments. At 2300 s in fact, a consistent number of agents has probability in $(0.4 \in 0.5]$, but they are relatively far from each other. When other agents get probability in $(0.3, 0.4]$ and $(0.5, 0.6]$ (and become similar in the task space), they fill the

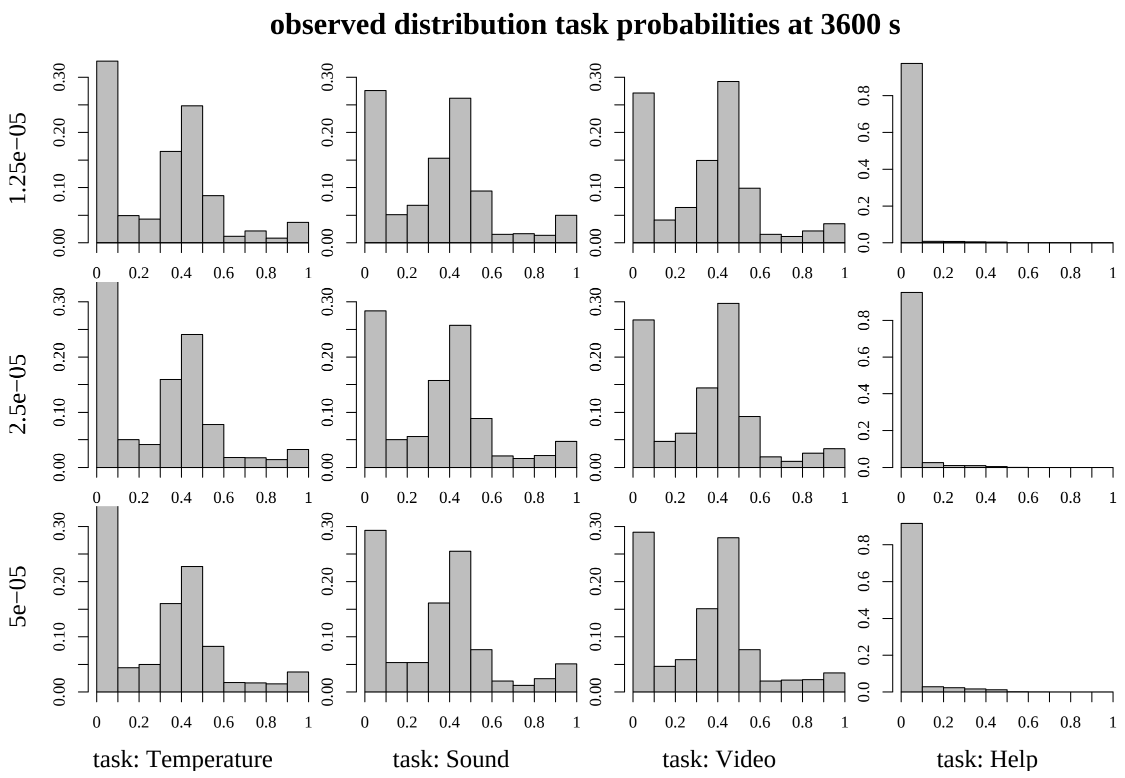


Figure 6.10: Final distribution of task probabilities (four robots). Each histogram in the plot array refers to a combination of tasks (one per column) and help densities (one per row). The histograms show the estimated density of task probability among the agents of the SANET. The x axes of the plots refer to the probability of performing a task, The y axes to the ratio of agents that were observed having probability of performing a task in the relative x range after 3600 s (note the different y scale for the last task).

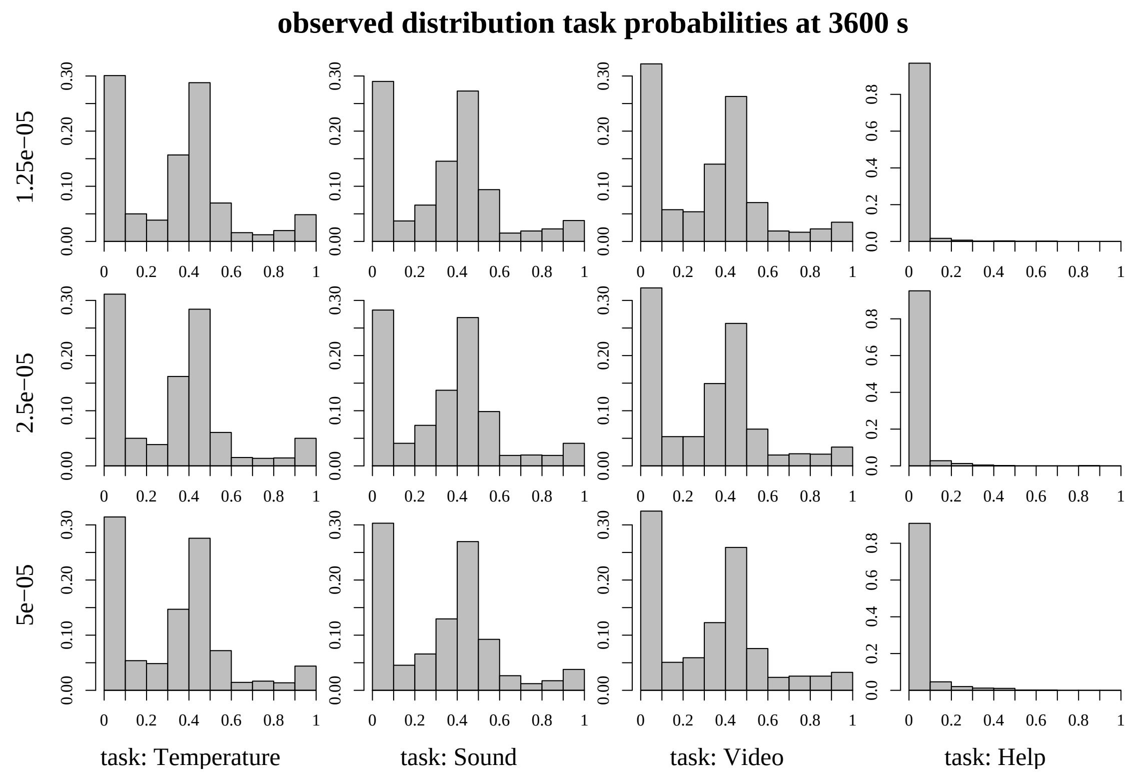


Figure 6.11: Final distribution of task probabilities (eight robots). See Fig. 6.10 for the meaning of the plots.

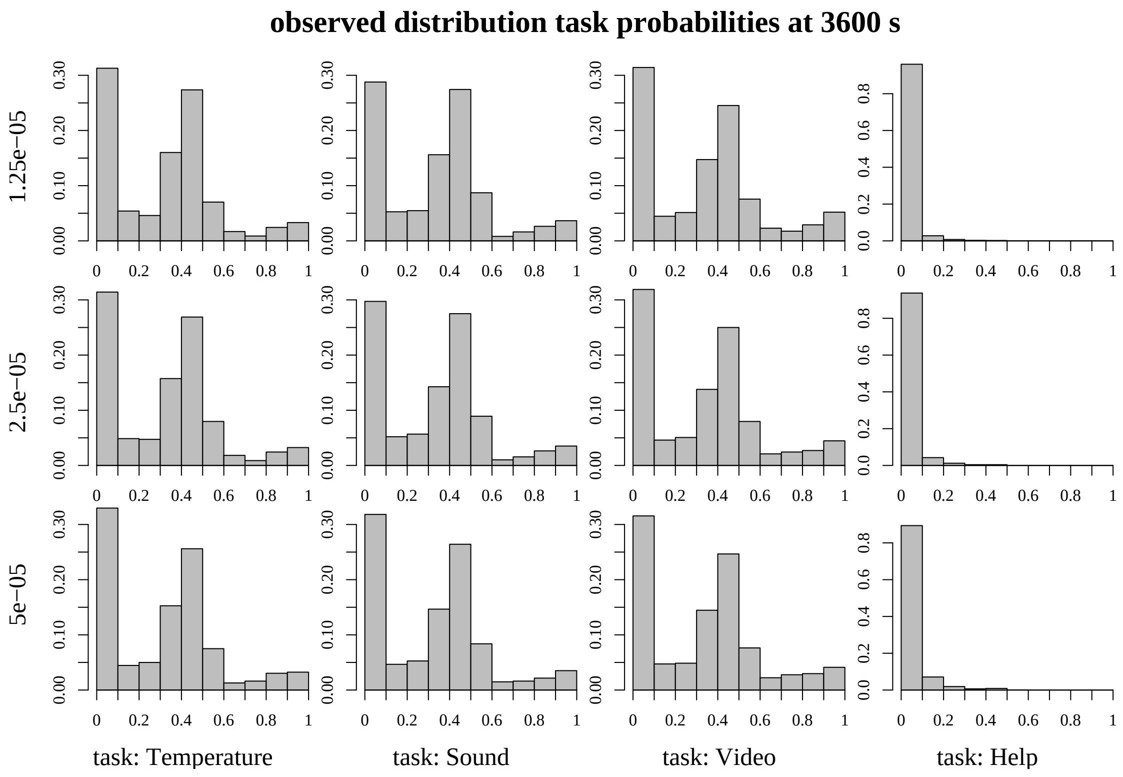


Figure 6.12: Final distribution of task probabilities (twelve robots). See Fig. 6.10 for the meaning of the plots.

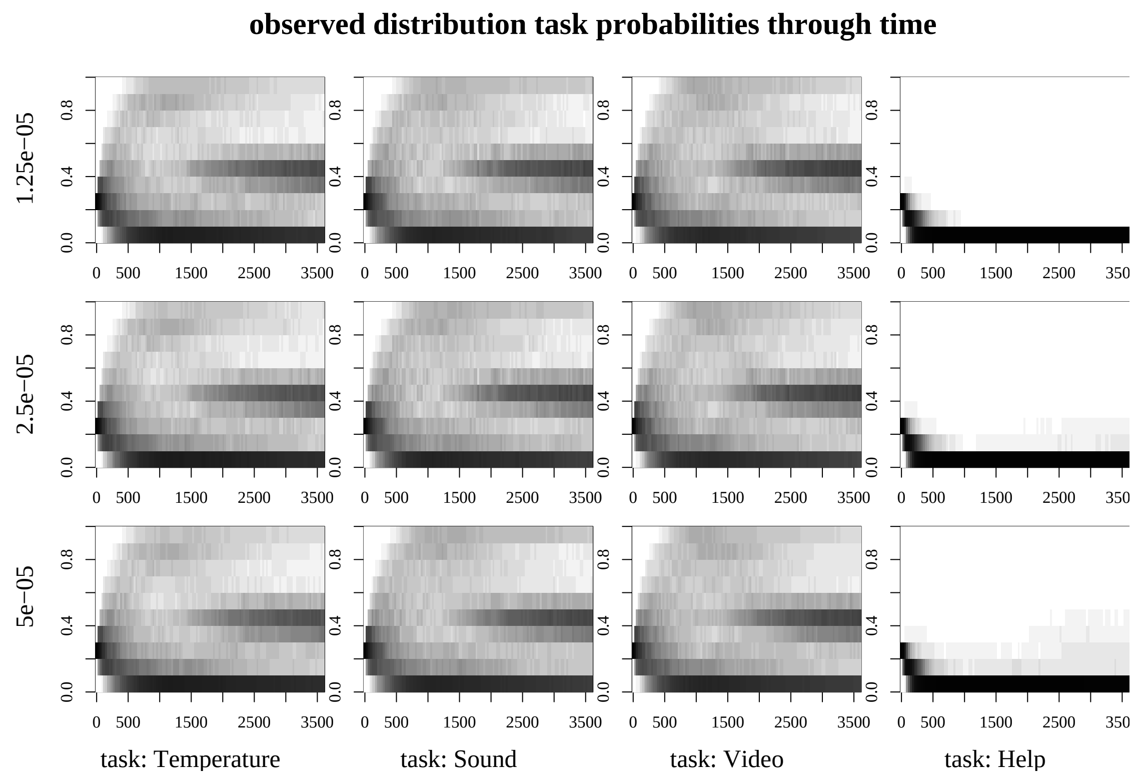


Figure 6.13: Dynamics of the distribution of task probabilities (four robots). Each plot in the array refers to a combination of tasks (one per column) and help densities (one per row). The task probabilities are on the y axes, the time from the beginning of the experiments on the x axes. The meaning of the plots is as in Fig. 4.13. The plots refer to experiments with four robots in the SANET.

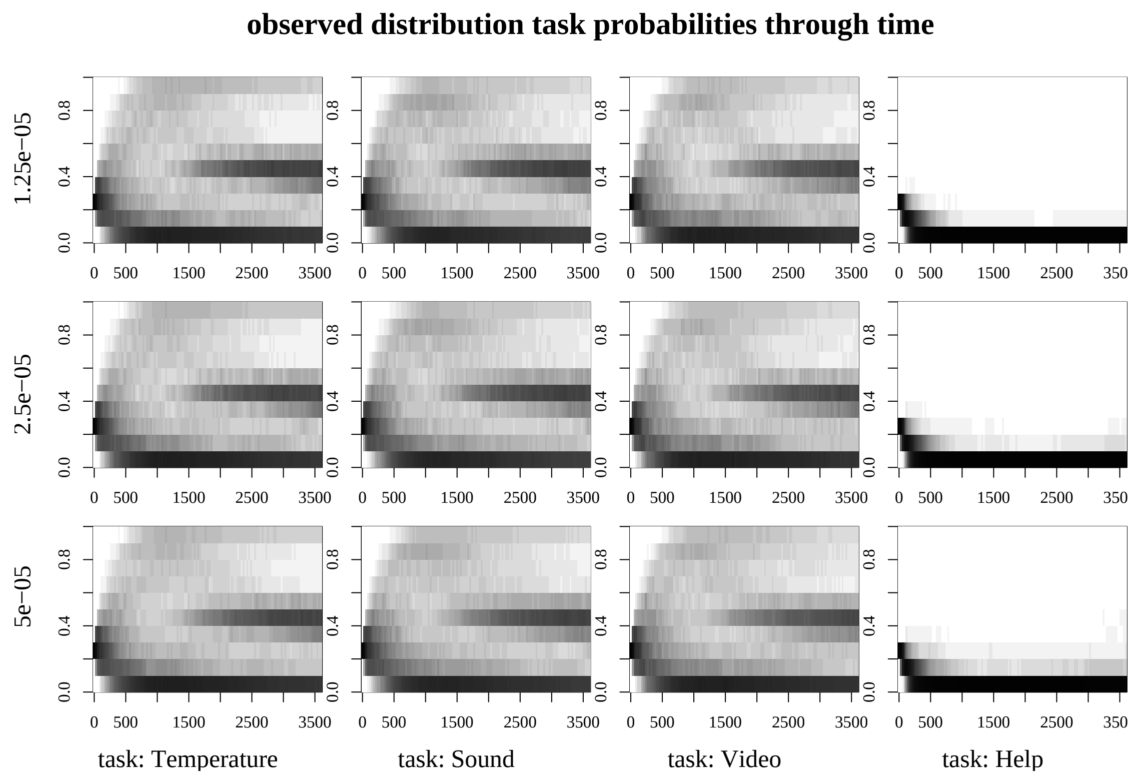


Figure 6.14: Dynamics of the distribution of task probabilities (eight robots)

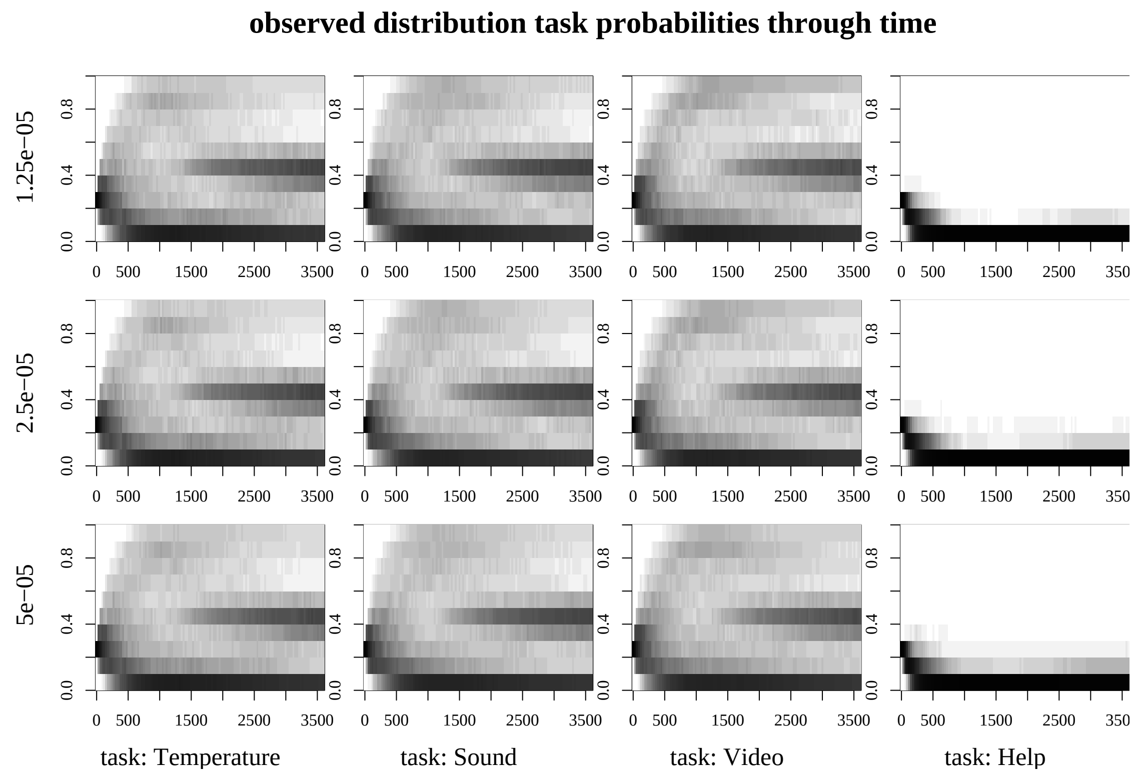


Figure 6.15: Dynamics of the distribution of task probabilities (twelve robots)

positions between the previous agents. They are therefore also physically nearer, and this is why the hierarchic social entropy decreases.

The distributions regarding task T_4 are qualitatively different from the others. There are many more inactive agents than in the other case. This can be easily explained by the low rates used to generate the requests. As a matter of fact, the higher the help density, the more agents get involved in the task T_4, as can be easily seen in the rightmost plots from Fig. 6.10 to Fig. 6.12

6.10 Conclusions

This chapter described a common architecture for division of labour in SANETs. We see our work as an important contribution to the field. Namely:

- We implemented a communication-less mechanism for division of labour. Although it might sound paradoxical, it could happen that agents in a SANET can not use the communication media (e.g., the wireless channel) to co-ordinate the division of labour. It is the case, for instance, when it is overloaded. To the best of our knowledge, ours is the first communication-less mechanism for division of labour in SANETs.
- We modified *AntHocNet* to explicitly take into consideration the heterogeneity of the agents.
- We proposed an architecture based on cross-layer interactions. Although this is an often-discussed topic in the community [e.g. Winter, 2006], we are aware of no work in which division of labour is achieved through cross-layer interactions.
- By means of the hierarchic social entropy, we showed how it is possible to analyse the SANET as a whole, and not by looking at its two components (robots and motes) independently.
- We implemented a new simulator that can handle both the simulation of the real world and of the networking. This is also something new in the community.

We could say, to summarise and abstract our contribution to the field, that we helped to look at SANETs in a holistic fashion, and not as made of two distinct parts. One of the purposes of our architecture is in fact the integration of two different platforms.

Future works will focus on energy-aware routing and reconfiguration. Motes are small devices and thus have limited energy capacity. In most operational conditions for SN, such as surveillance or monitoring, it is critical to reduce the energy consumption in order to increase the network life time. Most of the energy is used for wireless communication. Network life time can be increased by sending messages over routes that spare the overall energy consumption, or by selecting routes that do not involve a node with depleted capacity. Our modifications to *AntHocNet* allow to select routes based on several policies, included energy saving. They also allow to change the policy during the node lifetime, and to use different criteria for different packet types. This could be exploited to satisfy some Quality-of-Service requirements of the network.

Reconfiguration is essential for adaptive systems. If a SANET has to work in unknown or dynamic environments, its members should be able to change their behaviours to improve the group's performance. The architecture that we designed can be effectively used for autonomous reconfiguration and adaptation of the SANET to different conditions. The advantage of our approach is that the overhead is quite limited. The agents mostly adapt using the information that they can receive locally, that is, the packets that were routed through them. The only overhead is introduced by the ROUTEFAILURE packets.

Our architecture can still be improved. Figure 6.9 shows that the hierarchic social entropy is still far from the approximated upper bound. One could do better, for instance, by pre-programming the nodes to perform specific tasks, and then to deploy them uniformly in the environment. The hierarchic social entropy so achieved would probably be the maximum. The problem with this approach is that it is not an easy task to re-program the network after it has been deployed, especially if the network is overloaded.

Robots could be helpful for reprogramming the nodes. They could reach the nodes that have to be reprogrammed and instruct them about their new task. But this task of the robots' is nothing else than another formulation of our task T_4.

The work presented in the last pages is still preliminary, and thus it can be improved in many directions. Other researchers might also desire to compare their algorithms with ours. Another important feature discussed in the previous pages is that we propose a way of looking at the system which allows direct comparisons and evaluations. We hope that this could inspire other researchers to compare different solutions for SANETs, as we did for prey retrieval in Chap. 5.

Chapter 7

Conclusions

Division of labour is an effective way of improving the resource usage of Multi Robot Systems. We studied in the previous chapters how division of labour can be achieved through communication-less algorithms.[1] We focused on this class of algorithms because they can be helpful in situations likely to occur in real-world applications. It might happen that the robots can not access the communication channel, because it is not yet set up or it is overloaded. It might also be that robots have no communication channel at all.

Communication-less algorithms for co-ordination are also a stimulating intellectual challenge. Many, not only in robotics, still consider co-ordination without direct communication a sort of paradox. Works like ours and all the field of Swarm Robotics show that indeed direct communication is not always a necessity.

We proposed in Chap. 4 a division of labour algorithm inspired by ants' behaviour. The robots do not need to explicitly communicate with each other, but they exploit only the local information available in the environment for co-ordination. We showed that the algorithm could improve the efficiency of the group of robots. Moreover, robots using this algorithm could autonomously recruit the nest-mates that were best suited for retrieval. The algorithm used neither direct communication nor representations of the environment and of the robots. It worked by exploiting the complexity of the interactions among the robots and between the robots and the environment.

We took then some of the algorithms known in the literature and we compared them with ours in Chap. 5. The comparison was done using concepts of experiment design that that helped us to spare experimental time. We were able to find a difference between the algorithms. Thanks to the results, we were able to say which algorithms performed better. We were also able to trace the global behaviour of the group to the particular implementations of the algorithms, that is, to the learning rule used by the robots. The work in this chapter is still not sufficient to name one of the algorithms as the "state of the art".

[1]We recall that by "communication-less" algorithms we refer to algorithms that do not use *direct* communication (see Chap. 1).

Nevertheless, this is the first step in this direction and paves the way to future comparisons.

We finally exploited the knowledge of the mechanisms at work to move from one task to more tasks in Chap. 6. The control architecture for robots and motes that we described can be used to address at the same time the problems of division of labour, multi criterion routing and reconfiguration, although we analysed only the first one. Even though the robots and the motes can communicate with each other, the division of labour occurs by means of indirect communication. We mean that the agents never exchange messages which explicitly assign one agent to one task. Nor were the packets explicitly used for co-ordination. The agents took their decisions only by looking at the packets which travel through them. Agents never asked for information to far away nodes (except for when robots were answering to motes' help requests). They use instead the information that is already included in the packets they receive for routing.

The main advantage of using communication-less systems, like ours and those of SR, is that the system might perform more robustly than using a traditional approach. The redundancy and the simplicity of single individuals can also have important economic aspects. Each single unit might be produced with cheap methods and nevertheless the system might reach better results than using a few expensive robots.

If robotic systems in general are not yet ready for real-world outdoor applications, this is even truer for SR. In fact, the advantage of SR is counterbalanced by a non-trivial problem. The experiments described in the previous pages showed that the global behaviour of the robots is mostly due to the interactions among robots and between robots and environment. Such interactions are usually complex and non-linear. It is hard for a human designer to foresee which global behaviour might emerge from the simple behaviours of the robots. This implies also that it is difficult to find the robots' behaviours that bring to a desired global behaviour.

7.1 Summary of Contributions

This thesis showed how division of labour can be achieved without direct communication among the robots. The main contributions of our work are as follows:

- We introduced a new, bio-inspired, algorithm for division of labour. We proved that this algorithm is a valid mean to improve the efficiency of the group. It is adaptive and can also automatically select the individuals best suited for the task. The analysis of its characteristics allowed us to understand how the algorithm works and gave us also some means of measuring its qualities.

- Through our analysis of the algorithm, we validated also the original model proposed in biology, which was tested only in numerical simulations.

- We introduced elements of experiment design for the comparison of different algorithms. This allowed us to reduce the number of experiments required to assess significant differences, and thus to spare resources.

- We assessed which algorithm is more efficient and which performs better. We were also able to trace the global behaviour of the group to the particular implementations of the algorithms, that is, to the learning rule used by the robots. Continuing in this direction, we will be eventually able to create a common knowledge base about the algorithms for robot co-ordination.

- We developed a communication-less algorithm for SANETs. To the best of our knowledge, this is the first algorithm of this kind. We analysed it and proposed also some way of measuring the degree of division of labour in SANETs. This method will be used for future comparisons with other solutions.

- We developed a new simulator, BARAKA, for experimentation with SANETs. It was intentionally developed for our work and represents a novelty in the SANET research panorama.

Looking back at our work, we can summarise it by saying that we showed how we can synthesise robots' behaviour for SR. The problem of synthesising robots' behaviour given a desired global behaviour is in fact a hard one, as we explained in the previous section. Some recent works have followed a top-down approach. For instance, in Evolutionary Robotics (ER) [Nolfi and Floreano, 2000, Trianni, 2006] the designer of the system fixes an evaluation function of the group (called *fitness* function) and then uses evolutionary algorithms to set up the parameters of the neural networks that control the robots.

This thesis showed how we can proceed in the opposite direction, that is, bottom-up. We started from a given algorithm, we tried to understand its characteristics and we compared it with others. Finally we tried to apply our understandings to other domains.

We think that it is really important to test different solutions in this process, as we did in Chap. 5. Comparisons are necessary for the development of robotics. We could see the scientific evolution of a field, and also of robotics, as an evolutionary process. Evolution is roughly based on two processes: generation of new individuals and selection. Mutations and crossover (for species that use sexual reproduction) continuously generate new individuals with new characteristics. Selection lets only the fittest mate and reproduce, increasing thus the average fitness of the population. In the same fashion, new scientific ideas are continuously brought forth to explain some facts or to propose new solutions to a problem. Only the best ideas, that is, the ones that show best results, are the ones that remain "alive" in the community. We do think that robotics has showed a lot of new ideas, but very little selection.

7.2 Future Work

Our work can be expanded in several directions. We can expand the work presented in each chapter individually, but we can also think of new research directions.

The first way of expanding our work is to find other algorithms for comparisons, and then use also other test applications. Pursuing this direction, we will be able to create a common knowledge in the robotic community about the characteristic of different algorithms for different applications. We will eventually know which is the best algorithm for a given application.

Our work on SANETs still requires validation with real hardware. This is the most-likely next step of our work. There is however a practical problem, because this validation requires a lot of resources. Working with fewer resources, we will set up only some proof-of-concept experiments with fewer motes and in a smaller area. We are however confident that the validation will be successful. We took particular attention in the development of our simulator, using the knowledge acquired by the previous experience. Since the previous simulations were successfully validated, we do not expect BARAKA to fail.

A further extension of our work in SANETs consists in making the task more complex. We will put more obstacles in the environment. The obstacles could eventually move, simulating for instance a collapse of some structure. This will be useful to study the ability of our SANET to adapt.

The whole thesis focused only on division of labour based on individual learning. An interesting extension would be to study the effects of other techniques combined with ours. For instance, we will study the effects of division of labour which uses both communication and individual learning. We recall that we forbade our robots to use communication not because we are against it, but because it is a situation still poorly analysed in the literature. Yet, it is likely to occur. We are sure that communication combined with learning can improve the efficiency of the group of robots. What we do not know yet is whether the two elements simply sum up their effects or whether they work synergically. It could also occur that the result is less than the sum of both effects. In the last two cases, then it would be important to know why.

In every case, it is still a long way before we will be able to have the decent coffee of which we were dreaming at the beginning of this thesis.

Appendix A

Reinforcement Learning and MRSs

Reinforcement Learning (RL) is a computational approach to learning from interaction to achieve a goal. RL means

> ...learning what to do—how to map situations to actions—so as to maximize a numerical reward signal [Sutton and Barto, 1998].

The learner and decision-maker is called an *agent*. The things it interacts with, comprising everything outside the agent, are called the *environment*. We first introduce the theory of RL for a single agent and then expand it to multiple agents, MRS included.

The agent-environment interaction is shown in Fig. A.1. The agent interacts with the environment at a sequence of discrete time steps $t = 0, 1, 2, 3 \ldots$. At each time step t, the agent can perceive the environment through its sensors. The environment can be in any state $s_t \in S$, where S is the set of all possible states. The agent can perform an action $a_t \in A(s_t)$, where $A(s_t)$ is the set of actions available in state s_t. As a consequence of this action, the agent receives a reward signal $r_{t+1} \in \mathbb{R}$ and the environment is in state s_{t+1}. This reward signal is the most important aspect of RL.

The agent keeps a mapping $\pi : S \times A \to [0, 1]$, where $A = \bigcup_{s \in S} A(s)$. $\pi(s, a)$ is called a *policy*, and returns the probability that the agent performs action a if the state is s. RL methods modify the agent's policy as a result of its experience. Because it is time dependent, the policy is often written as $\pi_t(s, a)$. The agent's goal, roughly speaking, is to find the policy $\pi^*(s, a)$ that maximises the total amount of reward it receives over the long run.

The total amount of reward to maximise can be specified in different ways. If, for instance, the agent should work for only T time steps, at time t it chooses the action that maximises

$$R_t = \sum_{k=1}^{T-t} r_{t+k} \quad , \tag{A.1}$$

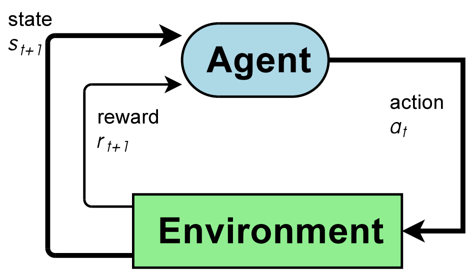


Figure A.1: Interactions between one agent and the environment in the RL framework.

called *return*. If $T \to \infty$, that is if the agent's task goes on without limit, (A.1) can easily diverge. In such cases, it is preferable to maximise the *discounted return:*

$$R_t = \sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k} \quad , \tag{A.2}$$

where γ is a parameter, $0 \leq \gamma \leq 1$, called *discount factor*.

A Markov Decision Process (MDP) is used as the theoretical background of RL. A MDP is defined by the set S of states and A of actions. In a MDP, the following property is true:

$$P\left(s_{t+1} = s, r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0, r_0\right) = \\ = P\left(s_{t+1} = s, r_{t+1} = r | s_t, a_t\right) \ ,$$

for every time step t, for every state s and s_t, for every reward r and r_t and for every action a_t. This means that the probability to go into a new state and obtain a certain reward does not depend on the history of the system, but only on the current state and action. Given this property, a MDP is completely described by the following distribution:

$$\mathcal{P}^a_{ss'} = P\left(s_{t+1} = s' | s_t = s, a_t = a\right) \quad ,$$

which gives the *transition probability* from one state to the other, and by

$$\mathcal{R}^a_{ss'} = E\left[r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\right] \quad ,$$

that is, the expected reward of the next action [Sutton and Barto, 1998].

The extension of a MDP to more agents is quite straightforward, from the formal point of view. We assume that there are k agents working together. The set of states is still indicated by S. Each agent i has its own set of available actions $A_i = \bigcup_{s \in S} A_i(s)$. The set of combined actions $\mathbf{A}$ is defined as the Cartesian product of all sets A_i, that is, $\mathbf{A} = \times_{i=1}^{k} A_i$. Therefore, an element $\mathbf{a} \in \mathbf{A}$ is in fact the vector $[a_1, a_2, \ldots, a_k]$, which represent the actions of all the agents. The transition probabilities are expressed as above, except that a is substituted by $\mathbf{a}$, a vector of actions. Each agent might have different goals, which are expressed by different rewards. Therefore, the expected reward becomes a vector to account for different agents:

$\mathbf{R}^a_{ss'} = [\mathcal{R}^a_{ss',1}, \mathcal{R}^a_{ss',2}, \ldots, \mathcal{R}^a_{ss',k}]$. Such framework is often referred to as a *Markov Game* [Littman, 1994]. From the practical point of view, the extension to MRS increases combinatorially the search space, therefore it is more difficult to find optimal policies for all the agents.

Things can go even worse. In fact, an important assumption so far was that each agent can perfectly perceive the state of the environment and knows the actions of the other agents. Such assumption might not hold true in case of unreliable sensors or communication media among the agents. In this case, the MDP model must be extended into a Partially Observable Markov Decision Process. The sets S, $\mathbf{A}$, the transition probabilities and the rewards are defined as in a MDP. The sensors of agent i can produce an observation $\omega \in \Omega_i$. The set $\mathbf{\Omega}$ of combined observations of the group is defined as $\mathbf{\Omega} = \times_{i=1}^{k} \Omega_i$. The relationship between the observations and the states in the environment is given by

$$\mathcal{O}^{\mathbf{a}}_{\omega s} = P(\omega_t = \omega | s_t = s, \mathbf{a}_{t-1} = \mathbf{a}) \quad ,$$

that is, the probability that the combined observation at time t of the actual state s_t given the previous combined action $\mathbf{a}_{t-1}$ is ω. Note that according to this formulation, it could happen that the agents would perceive the same observation ω in two different states, hence the partial observability of the environment.

Pynadath and Tambe [2002] analysed the time complexity of finding the optimal combined policy in a POMDP using different communication strategies among the agents. Their finding is that communication does not decrease the complexity when either the observations are not related to the states or there is only one possible observation for each state. Otherwise, communication can decrease the complexity only if it is cost-less, that is, if using it does not effect the reward of the agents. For the other cases, the more general ones, the complexity of finding optimal policies is NEXP.[1] The reader can refer to the original paper for further details.

A.1 POMDP for Prey Retrieval

The state of the environment is described by the number p of prey, while $s_i \in S_i = \{searching, resting\}$ describes the state of each robot. The environment is therefore fully described by the tuple $\langle p, \mathbf{S} = \times_i^N S_i \rangle$, where N is the number of robots. We can reduce the action set $\mathbf{A}$ by considering only the most important action for each robot i, that is, whether to exit the nest ($a_i = 1$) or not ($a_i = 0$). Therefore, $\mathbf{A} = \times_{i=1}^{N} \{0, 1\}$. If a robot k is in the *searching* state, then $a_k = 0$. The result can be seen in Fig. A.2. The state transition probabilities could be calculated knowing the probability that a new prey appears and the probability that one robot successfully retrieves a prey. Likewise for the expected reward vector $\mathbf{R}$. We do not report them here because they are not important for this discussion.

[1]NEXP is the class of problems which can be solved in exponential time by a non-deterministic Turing Machine [Garey and Johnson, 1979].

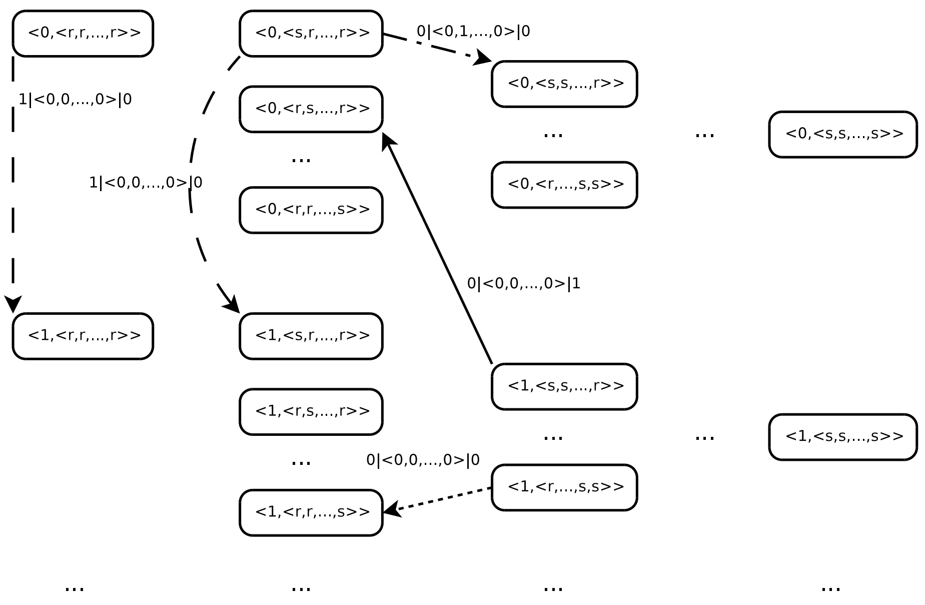
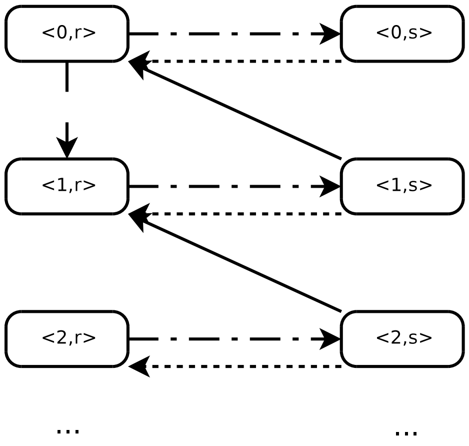
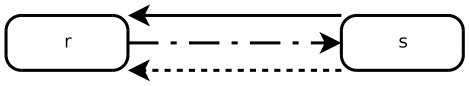


Figure A.2: A MDP for prey retrieval. Each state of the environment is identified by the number of the prey in it and by the state of all the robots ('s' stands for *searching* and 'r' for *resting*). We did not report all the possible transition between states, but just some to give an idea. Dashed arrows are transition due to new prey appearing. Dash-dotted arrows represent a robot that exits the nest. Continued arrows are for successful retrievals, while dotted arrows are for failures. Labels next to the arrow have the form $p|\mathbf{a}|r$, where p indicates the number of appeared prey (0 or 1), $\mathbf{a} \in \mathbf{A}$ is an action vector (see text), and r is the reward.

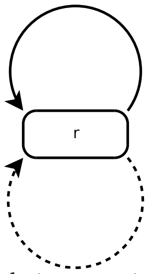
The important observation is that, given the sensor capabilities of our robots, the environment is in fact partially observable. The robots do not communicate and do not try to recognise the nest-mates, thus they can only observe their own state s_i. The MDP reduces to the POMDP shown in Fig. A.3(a). Given that our robots can not know the number of prey in the environment either, the POMDP becomes as in Fig. A.3(b). The only possible transitions are: *resting* $\rightarrow$ *searching* and *searching* $\rightarrow$ *resting*. Given that the decision point of the robot is in the *resting* state, the other state can in fact be ignored. The MDP is therefore a degenerated one: only one state (*resting*), two actions (exit the nest or not), and the expected reward which is the average of the rewards of all the real environment states unknown to the robots (Fig. A.3(c)). The methods already developed for RL are best suited for more complex MDP.



(a) First simplification: the robots can know only their own state.



(b) Second simplification: robots do not know how many prey are in the environment.



(c) Final simplification: a robots can not perform any action in state 's' (a timeout is modelled as a automatic transition to 'r' with null reward associated).

Figure A.3: POMDPs derived from the MDP. The limitations of our robots are introduced stepwise to show the degradation from a fully specified MDP for prey retrieval to a degenerated case. States and transitions are as in Fig. A.2. For simplicity, we do not report the labels associated with each transition.

Appendix B

On the "Efficiency" and the "Performance" of a Group of Robots

To compare the results of different algorithms, we use the concept of efficiency and performance that we describe in Sec. 4.1. However, the robotics literature offers several definitions for them, especially for "performance". For instance: Hayes [2002] defines "performance" as the time to find one object; Krieger and Billeter [2000] consider the inverse of the the total energy used by the robots normalised by their total number; Balch and Arkin [1998] use the time until completion of the task; Schneider-Fontán and Matarić [1996] the number of the retrieved prey and the time required to retrieve 80% of the prey; Balch and Arkin [1994] use "a combination of the time to complete the task and the cost of the system" (in their case, the cost of the system is estimated using the number of robots, multiplied by a constant). This appendix justifies our choice on the base of the definition, the etymology and the common use of the two terms.

Both "efficiency" and "performance" are used when someone or something is doing an action or an operation. The word that gives less problems is "efficiency", defined in the Merriam-Webster dictionary,[1] as:

> **1** : effective operation as measured by a comparison of production with cost (as in energy, time, and money) **2** : the ratio of the useful energy delivered by a dynamic system to the energy supplied to it.

"Efficiency" comes from the adjective "efficient", that derives from the participle "efficiens" of the Latin verb "efficiere", which means "to work out, to accomplish". It acquired the meaning "productive, skilled" only starting from 1787.[2]

[1] http://www.m-w.com
[2] http://www.etymonline.com

A fast search on the Internet, returns several different definitions given to "efficiency" according to the domain in which it is used. Here are some examples:[3]

- skillfulness in avoiding wasted time and effort;[4]
- the ratio of the energy output to the energy input;[5]
- an ability to perform well or achieve a result without wasted energy, resources, effort, time or money; [. . .] greater efficiency is achieved where the same amount and standard of services are produced for a lower cost, if a more useful activity is substituted for a less useful one at the same cost or if needless activities are eliminated;[6]
- a measure of the amount a parallel program spends doing computation as opposed to communication;[7]
- in mechanics, it is the ratio of the actual mechanical advantage (AMA) to the ideal mechanical advantage (IMA).[8]

From its definitions and etymology, we understand that "efficiency" is a concept related on *how well* a job is accomplished, regardless of the final result. The ability in concluding the operation is measured looking at the ratio between two quantities, ideally "what has been obtained" and "how much has been spent". For instance, if we are talking about a thermodynamical machines, efficiency is the ratio between the work delivered and the heat supplied. It we are talking about a financial operation, it is the ratio between income and costs. The definition given by (4.3) falls perfectly into line with these interpretations.

Merriam-Webster defines "performance" as:

> **1 a** : the execution of an action, **b** : something accomplished

but also as:

> **4 a** : the ability to perform (see EFFICIENCY) **b** : the manner in which a mechanism performs.

"Performance" derives from the verb "to perform", which comes from the Anglo-French "performir", altered from the old French "parfornir" which meant "to do, to carry out, to finish, to accomplish". It derives from "par" ("completely") and "fornir" ("to provide").

"Performance" is used with different meanings on the Internet:[9]

- the act of performing, of doing something successfully;[10]

[3] `http://www.google.com/search?q=define:efficiency`
[4] `http://www.cogsci.princeton.edu/cgi-bin/webwn`
[5] `http://sol.crest.org/renewables/SJ/glossary/E.html`
[6] `http://www.iime.org/glossary`
[7] `http://www.tc.cornell.edu/Services/Edu/Topics/Glossary/index.asp`
[8] `http://www.free-definition.com/Mechanical-efficiency.html`
[9] `http://www.google.com/search?q=define:performance`
[10] `http://www.cogsci.princeton.edu/cgi-bin/webwn`

- any recognised accomplishment: "they admired his performance under stress";[11]
- a measure of how well a fund is doing: two commonly used mutual fund performance measures are yield (which measures dividends) and total return (which measures dividends plus changes in net asset value);[12]
- refers to the metrics related to how a particular request is handled: for example, if a particular query takes 5 seconds to run, and after performance tuning, it now takes 3 seconds to run, we have boosted the performance of this query;[13]
- in engineering, performance relates to measuring some output or behaviour.[14]

Alongside with these, we find also some other definitions that tends to use "efficiency" and "performance" as synonyms, such as the following definition of "efficiency":

- A measure of the body's performance – the ratio of output over input.[15]

The impression is that when we talk about the "performance" of an operation, we usually focus on its final result. The "performance" is measured on the base of a metric that we define *a priori* and that is used to compare the results of the systems under observation. We are not, generally speaking, interested in how the final result has been obtained, i.e. in an efficient or inefficient way. As an example, let us consider a race car. It is more important the maximum speed than how much it consumes. Thus, the performance metric in this case is km/h and not l/km (litre per kilometre).

Confusion arises because sometimes the efficiency is used as a metric for the performance, as when an engineer has to reduce the consumptions of a car engine. This case explains why the two words may be used sometimes as synonym and why one can be used in the definition of the other. However, we would like to stress the point that "performance" is related to one metric, whereas "efficiency" involves always a comparison of two quantities. The following definition of "performance" states it quite clearly:

- "performance" is the degree to which a project or institution operates or operated according to various criteria or quality standards, such as Efficiency, Effectiveness, and Relevance.[16]

Given that "performance" focuses on the final result of the system under observation, each researcher defines the performance metric according to what his final purpose is. If we stick to prey retrieval in

[11] `http://www.cogsci.princeton.edu/cgi-bin/webwn`
[12] `http://www.oneinvest.com/bancone/mfgNOP.htm`
[13] `http://www.sql-server-performance.com/glossary.asp`
[14] `http://www.free-definition.com/Performance.html`
[15] `www.soton.ac.uk/~engenvir/glossary.html`
[16] `http://www.dfid.gov.uk/aboutdfid/files/glossary_p.htm`

robotics, one researcher might be interested in the retrieval of toxic items in the environment, which must be removed as soon as possible. The performance metric he would likely choose is the time to complete the mission. If he envisions a demining application, then the metric will be the number of retrieved item. If he just want to improve other solutions, he will choose to use the efficiency as performance metric.

Bibliography

M. Abolhasan, T. Wysocki, and E. Dutkiewicz. A review of routing protocols for mobile ad hoc networks. *Ad Hoc Networks*, 2:1–22, 2004.

W. Agassounon and A. Martinoli. Efficiency and robustness of threshold-based distributed allocation algorithms in multi-agent systems. In C. Castelfranchi and W.L. Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, pages 1090–1097. ACM Press, New York, NY, 2002.

W. Agassounon, A. Martinoli, and R.M. Goodman. A scalable, distributed algorithm for allocating workers in embedded systems. In *Proceedings of IEEE System, Man, and Cybernetics Conference SMC-01*, pages 3367–3373. IEEE Press, New York, NY, 2001.

W. Agassounon, A. Martinoli, and K. Easton. Macroscopic modeling of aggregation experiments using agents in teams of constant and time-varying sizes. *Autonomous Robots*, 17(2–3):163–192, 2004.

I.F. Akyildiz and I.H. Kasimoglu. Wireless sensor and actor networks: research challenges. *Ad Hoc Networks*, 2(4):351–367, 2004.

I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4), 2002.

J. Alcock. *Animal Behavior.* Sinauer, Sunderland, MA, 5th edition, 1995.

M.F. Ali, E.D. Morgan, C. Detrain, and A.B. Attygale. Identification of a component of the trail pheromone of the ant *Pheidole pallidula* (Hymenoptera: Formicidae). *Physiological Entomology*, 13:257–265, 1988.

R.C. Arkin. *Behavior Based Robotics*. MIT Press/Bradford Books, Cambridge, MA, 1998.

M. Asada and H. Kitano, editors. *RoboCup-98: Robot Soccer World Cup II*, volume 1604 of *Lecture Notes in Computer Science*, 1999. Springer Verlag, Heidelberg, Germany.

M. Asada, H. Kitano, I. Noda, and M. Veloso. RoboCup: Today and tomorrow – What we have learned. *Artificial Intelligence Journal*, 110: 193–214, 1999.

I. Asimov. *I, Robot.* Bantam Books, NY, 1991.

T. Balch. Hierarchic social entropy: An information theoretic measure of robot group diversity. *Autonomous Robots*, 8(3):209–238, 2000.

T. Balch. The impact of diversity on performance in multi-robot foraging. In O. Etzioni, J.P. Müller, and J.M. Bradshaw, editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 92–99. ACM Press, New York, NY, 1999.

T. Balch and R.C. Arkin. Communication in reactive multiagent robotic systems. *Autonomous Robots*, 1(1):27–52, 1994.

T. Balch and R.C. Arkin. Behavior-based formation control for multirobot teams. *IEEE Robotics and Automation*, 14(6):926–939, 1998.

M.A. Batalin and G.S. Sukhatme. Sensor network-based multi-robot task allocation. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS2003)*, volume 2, pages 1939–1944. IEEE Press, New York, NY, 2003.

M.A. Batalin and G.S. Sukhatme. Using a sensor network for distributed multi-robot task allocation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA2004)*, volume 1, pages 158–164. IEEE Press, New York, NY, 2004.

M.A. Batalin, M. Rahimi, Y. Yu, D. Liu, A. Kansal, G.S. Sukhatme, W.J. Kaiser, M. Hansen, G.J. Pottie, M. Srivastava, and D. Estrin. Call and response: Experiments in sampling the environment. In A. Arora and R. Govindan, editors, *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys'04)*, pages 25–38. ACM Press, New York, NY, 2004a.

M.A. Batalin, G.S. Sukhatme, and M. Hattig. Mobile robot navigation using a sensor network. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA2004)*, pages 158–164. IEEE Press, New York, NY, 2004b.

G. Beni and J. Wang. Swarm intelligence. In *Proceedings of the 7th Annual Meeting of the Robotics Society of Japan*, pages 425–428, Tokyo, Japan, 1989.

E. Bonabeau, G. Theraulaz, and J.-L. Deneubourg. Quantitative study of the fixed threshold model for the regulation of division of labor in insect societies. *Proceedings of the Royal Society of London, Series B-Biological Sciences*, 263:1565–1569, 1996.

E. Bonabeau, A. Sobkowski, G. Theraulaz, and J.-L. Deneubourg. Adaptive task allocation inspired by a model of division of labor in social insects. In D. Lundh, B. Olsson, and A. Narayanan, editors, *Biocomputing and Emergent Computation*, pages 36–45. World Scientific, UK, 1997.

E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York, 1999.

A. Bonarini, M. Matteucci, G. Invernizzi, and T. H. Labella. Context and motivation in coordinating fuzzy behaviors. In M. Colombetti, A. Bonarini, and P.L. Lanzi, editors, *Proceedings of the Seventh Meeting of the Italian Association for Artificial Intelligence (AI*IA 2000)*. AI*IA, Milano, Italy, 2000.

A. Bonarini, M. Matteucci, G. Invernizzi, and T.H. Labella. An architecture to coordinate fuzzy behaviors to control an autonomous robot. *Fuzzy Sets and Systems*, 134(1):101–115, 2001.

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

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, 2002.

N. Bulusu, J. Heidemann, and D. Estrin. Adaptive beacon placement. In P. Dasgupta and W. Zhao, editors, *Proceedings of the 21st Conference on Distributed Computing Systems (ICDCS-21)*, pages 489–498. IEEE Press, New York, NY, 2001.

J. Butler. Robotics and microelectronics: Mobile robots as gateways into wireless sensor networks. *Technology@Intel Magazine*, May 2003. Available on-line at `http://www.intel.com/technology/magazine/research/it05031.pdf`.

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, 2001.

M.C. Cammaerts. Systémes d'approvisionnement chez *Myrmica scabrinodis*. *Insectes Sociaux*, 27(4):328–242, 1980.

M.C. Cammaerts and R. Cammaerts. Food recruitment strategies of the ants *Myrmica sabuleti* and *Myrmica ruginodis*. *Behavioral Processes*, 5:251–270, 1980.

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

C.-Y. Chong and S. Kumar. Sensor networks: Evolution, opportunities, and challenges. *Proceedings of the IEEE*, 91(8), 2003.

M. Cirillo. A formal methodological approach for comparison of control algorithms for *Swarm Robotics*. Master's thesis, Politecnico di Milano, Milan, Italy, 2005.

Collodi. Pinocchio, 1883.

P. Corke, R. Peterson, and D. Rus. Networked robots: Flying robot navigation using a sensor net. In P. Dario and R Chatila, editors, *Proceedings of the Eleventh International Symposium of Robotics Research (ISRR2003)*, volume 15 of *Springer Tracts on Advanced Robotics (STAR)*. Springer Verlag, Heidelberg, Germany, 2003.

P. Corke, S. Hrabar, R. Peterson, D. Rus, S. Saripalli, and G. Sukhatme. Autonomous deployment and repair of a sensor network using an unmanned aerial vehicle. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA2004)*, volume 4, pages 3602–3608. IEEE Press, New York, NY, 2004.

E. Şahin, T.H. Labella, V. Trianni, J.-L. Deneubourg, P. Rasse, D. Floreano, L.M. Gambardella, F. Mondada, S. Nolfi, and M. Dorigo. SWARM–BOT: Pattern formation in a swarm of self–assembling mobile robots. In A. El Kamel, K. Mellouli, and P. Borne, editors, *Proceedings of IEEE International Conference on System, Man and Cybernetics (SMC2002)*. IEEE Press, New York, NY, 2002.

D.E. Culler and H. Mulder. Smart sensors to network the world. *Scientific American*, June 2004.

K. Dantu, M.H. Rahimi, H. Shah, S. Babel, A. Dhariwal, and G.S. Sukhatme. Robomote: Enabling mobility in sensor networks. In *IEEE/ACM Fourth International Conference on Information Processing in Sensor Networks (IPSN-SPOTS)*, pages 404–409. IEEE Press, New York, NY, April 2005.

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. Detrain. Field study on foraging by the polymorphic ant species *Pheidole pallidula*. *Insectes Sociaux*, 37(4):315–332, 1990.

C. Detrain and J.-L. Deneubourg. Scavenging by *Pheidole pallidula*: a key for understanding decision-making systems in ants. *Animal Behaviour*, 53:537–547, 1997.

C. Detrain and J.M Pasteels. Caste differences in behavioral thresholds as a basis for polyethism during food recruitment in the ant *Pheidole pallidula* (Nyl.) (Hymenoptera: Myrmicinae). *Journal of Insect Behavior*, 4(2):157–176, 1991.

C. Detrain and J.M. Pasteels. Regulated food recruitment through individual behavior of scouts in the ant *Myrmica sabuleti* (Hymenoptera: Formicidae). *Journal of Insect Behavior*, 7(6):767–777, 1994.

G. Di Caro, F. Ducatelle, and L.M. Gambardella. AntHocNet: An adaptive nature-inspired algorithm for routing in mobile ad hoc networks.

European Transactions on Telecommunications, Special Issue on Self-organization in Mobile Networking, 16(5):443–455, 2005.

E. A. Di Paolo. An investigation into the evolution of communication. *Adaptive Behavior*, 6(2):285–324, 1998.

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

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

M. Dorigo, E. Tuci, R. Groß, V. Trianni, T.H. Labella, S. Nouyan, C. Ampatzis, J.-L. Deneubourg, G. Baldassarre, S. Nolfi, F. Mondada, D. Floreano, and L.M. Gambardella. The SWARM-BOTS project. In E. Şahin and W. Spears, editors, *Proceedings of the First International Workshop on Swarm Robotics at SAB 2004*, volume 3342 of *Lecture Notes in Computer Science*, pages 31–44. Springer Verlag, Berlin, Germany, 2004b.

M. Dorigo, E. Tuci, V. Trianni, R. Groß, S. Nouyan, C. Ampatzis, T.H. Labella, R. O'Grady, M. Bonani, and F. Mondada. *Computational Intelligence: Principles and Practice*, chapter SWARM-BOT: Design and Implementation of Colonies of Self-assembling Robots. IEEE Computational Intelligence Society, New York, NY, 2006.

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

D. Estrin, D. Culler, K. Pister, and G.S. Sukhatme. Connecting the physical world with pervasive networks. *PERVASIVE Computing*, pages 59–69, January 2002.

A. Farinelli, L. Iocchi, and D. Nardi. Multirobot systems: a classification focused on coordination. *IEEE Transactions on System Man and Cybernetics, part B*, 34(5):2015–2028, 2004.

M Flint, E. E. Fernández-Gaucherand, and M.M. Polycarpou. A probabilistic framework for passive cooperation among UAV's performing a search. In B. De Moor, P. Van Dooren, V. Blondel, and J. Willems, editors, *Proceedings of the Sixteenth International Symposium on Mathematical Theory of Networks and Systems (MTNS2004)*, Leuven, Belgium, July 5–9 2004.

D. Fresneau. Individual foraging and path fidelity in a Ponerine ant. *Insectes Sociaux*, 32:109–116, 1985.

A. Galstyan, B. Krishnamachari, K. Lerman, and S. Pattem. Distributed online localization in sensor networks using a moving target. In K. Ramchandran and J. Sztipanovits, editors, *IPSN'04: Proceedings of the third international symposium on Information processing in sensor networks*, pages 61–70. ACM Press, New York, NY, 2004.

Michael R. Garey and David S. Johnson. *Computers and Intractability / A Guide to the Theory of $\mathcal{NP}$-Completeness.* W.H. Freeman & Company, San Francisco, CA, 1979.

B.P. Gerkey and M.J. Matarić. A market-based formulation of sensor-actuator network coordination. In G.S. Sukhatme and T. Balch, editors, *Proceedings fo the AAAI Sping Symposium on Intelligent Embedded and Distributed Systems*, pages 21–26. AAAI Press, San Jose, CA, 2002.

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.

D. Goldberg and M.J. Matarić. Reward maximization in a non-stationary mobile robot environment. In C. Sierra, M. Gini, and J.S. Rosenschein, editors, *Proceeding of The Fourth International Conference on Autonomous Agents (Agents 2000)*, pages 92–99. ACM Press, New York, NY, 2000.

D. Goldberg and M.J. Matarić. Interference as a tool for designing and evaluating multi-robot controllers. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97)*, pages 637–642. MIT Press, Cambridge, MA, 1997.

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.

I. Harvey. Untimed and misrepresented: connectionism and the computer metaphor. *AISB Quarterly*, (96):20–27, 1996.

A.T. Hayes. How many robots? Group size and efficiency in collective search tasks. In H. Asama, T. Arai, T. Fukuda, and T. Hasegawa, editors, *Proceedings of the 6th International Symposium on Distributed Autonomous Robotic Systems (DARS-02)*, pages 289–298. Springer Verlag, Heidelberg, Germany, 2002.

G. Hinton and T.J. Sejnowski, editors. *Unsupervised Learning and Map Formation: Foundations of Neural Computation.* MIT Press, Cambridge, MA, 1999.

O. Holland and C. Melhuish. Stigmergy, self-organization and sorting in collective robotics. *Artificial Life*, 5(2):173–202, 1999.

B. Hölldobler. Recruitment behavior, home range orientation and territoriality in harvester ants *Pogonomyrmex*. *Behavioral Ecology and Sociobiology*, 1:3–44, 1976.

B. Hölldobler. Canopy orientation: a new kind of orientation in ants. *Science*, 210:86–88, 1980.

B. Hölldobler and E.O. Wilson. *The Ants*. Springer Verlag, Heidelberg, Germany, 1990.

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.

Yan Jin, A.A. Minai, and M.M. Polycarpou. Cooperative real-time search and task allocation in UAV teams. In *Proceedings of the 42nd IEEE Conference on Decision and Control*, volume 1, pages 7–12. IEEE Press, New York, NY, 2003.

C. Jones and M.J. Matarić. Sequential task execution in a minimalist distributed robotic system. Technical Report IRIS-02-414, Institute for Robotics and Intelligent Sytems, University of Southern California, Los Angeles, CA, March 2002.

C.V. Jones and M.J. Matarić. Adaptive division of labor in large-scale minimalist multi-robot systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 1969–1974. IEEE Press, New York, NY, 2003.

H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, and H. Matsubara. Robocup: a challenge AI problem. *AI Magazine*, 18(1), 1997.

M.J.B. Krieger and J.-B. Billeter. The call of duty: Self-organised task allocation in a population of up to twelve mobile robots. *Robotics and Autonomous Systems*, 30(1-2):65–84, 2000.

C. Kube and E. Bonabeau. Cooperative transport by ants and robots. *Robotics and Autonomous Systems*, 30(1-2):85–101, 2000.

C. Kube and H. Zhang. The use of perceptual cues in multi-robot boxpushing. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 3, pages 2085–2090. IEEE Press, New York, NY, 1996.

V. Kumar, D. Rus, and S. Singh. Robot and sensor network for first responders. *PERVASIVE Computing*, pages 24–33, October 2004.

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 and M. Birattari. Polyphemus: De alieni generorum abacorum racemo. Technical Report IRIDIA-TR-2004-15, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2004.

T.H. Labella and F. Dressler. A bio-inspired architecture for division of labour in SANETs. In *Proceedings of the First IEEE/ACM International Conference on Bio Inspired Models of Network, Information and Computing Systems (BIONETICS 2006)*, Cavalese, Italy, December 11–13, 2006. In press.

T.H. Labella, M. Dorigo, and J.-L. Deneubourg. Efficiency and task allocation in prey retrieval. In A.J. Ijspeert, M. Murata, and N. Wakamiya, editors, *Biologically Inspired Approaches to Advanced Information Technology: First International Workshop, BioADIT 2004*, volume 3141 of *Lecture Notes in Computer Science*. Springer Verlag, Heidelberg, Germany, 2004a.

T.H. Labella, M. Dorigo, and J.-L. Deneubourg. Self-organised task allocation in a group of robots. In R. Alami, editor, *7th International Symposium on Distributed Autonomous Robotic Systems (DARS04)*, pages 371–380, Toulouse, France, June 23–25, 2004b.

T.H. Labella, M. Dorigo, and J.-L. Deneubourg. Division of labour in a group of robots inspired by ants' foraging behavior. *ACM Transactions on Autonomous and Adaptive Systems*, 1(1):4–25, 2006a.

T.H. Labella, G. Fuchs, and F. Dressler. A simulation model for self-organised management of sensor/actuator networks. In *Fachgespräch über Selbstorganisierende, Adaptive, Kontextsensitive verteilte Systeme (SAKS)*, University of Kassel, Germany, March 23–24 2006b.

T.H. Labella, I. Dietrich, and F. Dressler. BARAKA: A hybrid simulator of sensor/actuator networks. In *Proceedings of the Second IEEE/Create-Net/ICST International Conference on COMmunication System softWAre and MiddlewaRE (COMSWARE 2007)*, Bangalore, India, January 7–12, 2007. In press.

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, A. Martinoli, and Y.S. Abu-Mostafa. Emergent specialization in swarm systems. In H. Yin, N. Allinson, R. Freeman, J. Keane, and S. Hubbard, editors, *Intelligent Data Engineering and Automated Learning (IDEAL 2002)*, volume 2412 of *Lecture Notes in Computer Science*, pages 261–266. Springer Verlag, Heidelberg, Germany, 2002.

L. Li, A. Martinoli, and Y.S. Abu-Mostafa. Diversity and specialization in collaborative swarm systems. In C. Anderson and T. Balch, editors, *Proceedings of the 2nd International Workshop on the Mathematics and Algorithms of Social Insects*, pages 91–98, Atlanta, Georgia, December 15–17 2003.

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.

M.L. Littman. Markov games as a framework for multi-agent reinforcement learning. In W.W. Cohen and H. Hirsh, editors, *Proceedings of the 11th International Conference on Machine Learning*, pages 157–163. Morgan Kaufmann, San Fransisco, CA, 1994. ISBN 1-55860-335-2.

K.H. Low, W.K. Leow, and M.H.Jr. Ang. Autonomic mobile sensor network with self-coordinated task allocation and execution. *IEEE Transactions on Systems, Man and Cybernetics, Part C*, 36(3):315–327, 2006.

M. Marina and D. Das. On-demand multipath routing for mobile ad hoc networks. In *Proceedings of IEEE International Conference on Network Protocols (ICNP)*, pages 14–23. IEEE Press, New York, NY, 2001.

A. Martinoli. Collective complexity out of individual simplicity. invited book teview on "swarm intelligence: From natural to artificial systems" by e.bonabeau, m. dorigo, and g. theraulazg. *Artificial Life*, 7 (3):315–319, 2001.

A. Martinoli and F. Mondada. Collective and cooperative group behaviours: Biologically inspired experiments in robotics. In *Proceedings of the Fourth International Symposium on Experimental Robotics*, pages 3–10, Stanford, CA, 1995. Springer Verlag.

M.J. Matarić. Learning social behaviors. *Robotics and Autonomous Systems*, 20:191–204, 1997a.

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

M.J. Mataric, M. Nilsson, and K.T. Simsarin. Cooperative multi-robot box-pushing. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 556–561. IEEE, 1995.

C. Melhuish, M. Wilson, and A. Sendova-Franks. Patch sorting: Multi-object clustering using minimalist robots. In J. Kelemen and P. Sosík, editors, *Proceedings of the Sixth European Conference on Artificial Life*, volume 2159 of *Lecture Notes in Computer Science*, pages 543–552. Springer Verlag, Heidelberg, Germany, 2001.

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

S. Nolfi and D. Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press/Bradford Books, Cambridge, MA, 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.

C.E. Perkins and E.M. Royer. Ad hoc On-Demand Distance Vector Routing. In *2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100. IEEE Computer Society, Los Alamitos, CA, 1999.

C.E. Perkins, E.M. Belding-Royer, and S. Das. Ad hoc on demand distance vector (AODV) routing. IETF RFC 3561, 2003. URL `http://www.ietf.org/rfc/rfc3561.txt`.

S. Poduri and G.S. Sukhatme. Constrained coverage for mobile sensor networks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA2004)*, pages 165–172. IEEE Press, New York, NY, 2004.

D.V. Pynadath and M. Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, 16:389–423, 2002.

P. Riley and M Veloso. Planning for distributed execution through use of probabilistic opponent model. In M. Ghallab, J. Hertzberg, and P. Traverso, editors, *Proceedings of the Sixth International Conference on AI Planning and Scheduling (AIPS-2002)*, pages 72–81. AAAI, Menlo Park, CA, 2002.

R. Rosengren and W. Fortelius. Orstreue in foraging ants of the *Formica rufa* group. *Insectes Sociaux*, 33:306–337, 1986.

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, February 2003.

F. Schätzing. *Nachrichten aus einem unbekannten Universum*. Kiepenheuer & Witsch, Köln, Germany, 2006.

M. Schneider-Fontán and M.J. Matarić. A study of territoriality: The role of critical mass in adaptive task division. In P. Maes, M.J. Matarić, J.-A. Meyer, J. Pollack, and S.W. Wilson, editors, *From Animals to Animats 4, Fourth International Conference on Simulation of Adaptive Behavior (SAB-96)*, pages 553–561. MIT Press/Bradford Books, Cambridge, MA, 1996.

C.E. Shannon. *The Mathematical Theory of Communication*. Univeristy of Illinois Press, 1949.

M.W. Shelley. Frankestein, or the modern prometheus, 1818.

G.T. Sibley, M.H. Rahimi, and G.S Sukhatme. Robomote: a tiny mobile robot platform for large-scale ad-hoc sensor networks. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA'02)*, volume 2, pages 1143–1148. IEEE Press, New York, NY, 2002.

B. Sinopoli, C. Sharp, L. Schenato, S. Schaffert, and S. Sastry. Distributed control applications within sensor networks. *Proceedings of the IEEE*, 91(9):1235–1246, 2003.

J.H. Sudd. How insects work in group. *Discovery*, 24(6):15–19, 1963.

J.H. Sudd and M.E. Sudd. Seasonal changes in the response of woodants to sugar baits. *Ecological Entomology*, 10:89–97, 1985.

G.S. Sukhatme and M.J. Matarić. Embedding robots into the internet. *Communications of the ACM*, pages 67–73, May 2000.

R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, Cambridge, MA, 1998.

P. Trakadas, T.B. Zahariadis, S. Voliotis, and C. Manasis. Efficient routing in pan and sensor networks. *Mobile Computing and Communications Review*, 8(1):10–17, 2004.

V. Trianni. *On the Evolution of Self-Organising Behaviours in a Swarm of Autonomous Robots*. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium, 2006.

V. Trianni, T.H. Labella, R. Groß, E. Şahin, M. Dorigo, and J.-L. Deneubourg. Modeling pattern formation in a swarm of self-assembling robots. Technical Report IRIDIA-TR-2002-12, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2002.

V. Trianni, R. Groß, T.H. Labella, E. Şahin, P. Rasse, J.-L. Deneubourg, and M. Dorigo. Evolving aggregation behaviors in a swarm of robots. In W. Banzhaf, T. Christaller, P. Dittrich, J.T. Kim, and J. Ziegler, editors, *Advances in Artificial Life - Proceedings of the 7th European Conference on Artificial Life (ECAL)*, volume 2801 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, Berlin, Germany, 2003.

V. Trianni, T.H. Labella, and M. Dorigo. Evolution of direct communication for a swarm-bot performing hole avoidance. In M. Dorigo, M. Birattari, C. Blum, L.M. Gambardella, F. Mondada, and T. Stützle, editors, *Ant Colony Optimization and Swarm Intelligence – Proceedings of ANTS 2004 – Fourth International Workshop*, volume 3172 of *Lecture Notes in Computer Science*, pages 131–142. Springer Verlag, Berlin, Germany, 2004.

A. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936.

A. Turing. The chemical basis for morphogenesis. *Philosophical Transactions of the Royal Society of London, Series B*, 237(641):37–72, 1952.

V.N. Vapnik. *Statistical learning theory*. Wiley, New York, NY, 1998.

R. Wehner, R.D. Harkness, and P. Schmid-Hempel. *Foraging Strategies in Individually Searching Ants* Cataglyphis bicolor. G. Fisher Verlag, Stuttgart, Germany, 1983.

L.L. Whitcomb. Underwater robotics: Out of the research laboratory and into the field. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation (ICRA 2000)*, volume 1, pages 709–716, New York, NY, 2000. IEEE Press.

M. Wilson, C. Melhuish, A.B. Sendova-Franks, and S. Scholes. Algorithms for building annular structures with minimalist robots inspired by brood sorting in ant colonies. *Autonomous Robots*, 17:115–136, 2004.

R. Winter. *A Cross-layer Framework for Network-wide Adaptations and Optimizations in Mobile Ad Hoc Networks*. PhD thesis, Freie Universität Berlin, Germany, 2006.

M. Younis, K. Akkaya, and A. Kunjithapatham. Optimization of Task Allocation in a Cluster-Based Sensor Network. In *Proceedings of the Eighth IEEE Symposium on Computers and Communications (ISCC 2003)*, pages 329–334. IEEE Computer Society, Los Alamitos, CA, 2003.