Prey Retrieval by a Swarm of Robots

Thomas Halva LABELLA

Technical Report No.

TR/IRIDIA/2003-16
May 2003

DEA thesis

Prey Retrieval by a Swarm of Robots

by

Labella Thomas Halva

Université Libre de Bruxelles, IRIDIA
Avenue Franklin Roosevelt 50, CP 194/6, 1050 Brussels, Belgium
hlabella@ulb.ac.be

Supervised by

Marco Dorigo, Ph.D.

Maître de Recherches du FNRS
Université Libre de Bruxelles, IRIDIA
Avenue Franklin Roosevelt 50, CP 194/6, 1050 Brussels, Belgium
mdorigo@ulb.ac.be

August, 2003

A thesis submitted in partial fulfilment of the requirements of the *Université Libre de Bruxelles, Faculté de Sciences Appliquées* for the

DIPLOME D'ETUDES APPROFONDIES (DEA)

Abstract

Prey retrieval, also known as foraging, is a widely used test application for Multi Robot Systems (MRS). The task consists in searching for objects spread in the environment and in bringing them to a specific place called *nest* or *home*. Scientific issues usually concern efficient exploration, mapping, communication among agents, task coordination and allocation, and conflict resolution. In particular, conflicts prevent the performance of foraging (i.e. the number of items retrieved) from growing linearly with the number of robots. Collisions among robots or two robots that try to move an object but in different directions are examples of conflicts. From an energetic point of view, conflicts can be seen as energy that is lost during the retrieval. Several works in the literature investigate how the control system of each agent or some form of middle/long range communication can improve performance. In this work, we study a different approach to conflict resolution based only on information locally available to each robot. We show that it is possible to reduce conflicts by adapting for each robot the probability to leave the nest according to previous successes or failures. We derive an estimator of the efficiency of the system and show that this form of adaptation improves it. Efficiency increases through the allocation of tasks among members of the team and the exploitation of their individual abilities.

Acknowledgements

I would like first to thank all the people in IRIDIA: working with them makes hard times more enjoyable. I would like especially to thank the people that reviewed the manuscript and gave important suggestions (in completely random order): Vito, Elio, Josh, Ciro, Julia, Rodi. Thanks also to Mauro, for suggestions about the procedural aspects of the experiments, and to my supervisor, Marco Dorigo, for criticisms and suggestions about future directions. Last but not least, I would like to thank my family for their support and Daniela for standing a man that prefers to play with Lego instead of finding a real job. This work is dedicated to my father.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In the past few years, much research in robotics has focused on collective robotic systems, where several independent robots work together to achieve a given goal. Scientific issues can have both an engineering or a biological origin. From an engineering perspective, systems made of several agents are particularly appealing because they represent a way of improving the efficiency of the solution in tasks that are intrinsically parallel, such as the delivery of items in a factory or the exploration and mapping of unknown environments. Moreover, a collective robotic system is robust in case of failures of any of its members. From a biological perspective, many robots working together (or in competition) are interesting because they are a good test-bed for theories about self-organisation (Camazine et al., 2001).

Collective robotics belongs to the field that studies Multi Agent Systems (MASs). The distinction between the two is that while in the latter an *agent* can act also in a virtual environment, in the former a robot must deal with the real world. Much work in the literature uses simulations to help the designer to test solutions that would be too risky to be run on real robots (e.g. when the robot's motors could be damaged), or too slow to develop on them (e.g. in Evolutionary Robotics, see Nolfi and Floreano (2000)). In the real world, physical interactions among robots are more complex than in a simulated environment. This complexity makes robotic systems richer of features and more complex to implement.

An important issue in MAS is the design of control systems for each agents that can deal with complex and dynamic situations. Solutions usually make use of complex models of the environment. Recently, researchers involved in the design of robotic control systems have looked at biological systems to learn how very complex tasks are solved by swarms of animals through the exploitation of simple behaviours. They have noticed that it is not always necessary to use a high degree of complexity in a control system in order to cope with difficult environments. It has been shown that ant colonies can solve apparently complex tasks by exploiting the dynamics brought forth by the interactions among agents and between agents and the environment. For instance, ants often find the shortest path to a food source when foraging. This happens without the means of direct communication between them, but they exploit instead some features of the environment. The interplay between the pheromone laid by each ant and its evaporation makes the shortest path become the preferred one (Bonabeau et al., 1999, p. 26–31).

One advantage of using simple behaviours like in ants is that control programs usually do not rely explicitly on direct means of communication between agents, but exploit the environment. "In situations where many individuals contribute to a collective effort, such as a colony of termites building a nest, stimuli provided by the emerging structure itself can be a rich source of information for the individual" (Camazine et al., 2001, page 23). This form of communication is called *stigmergy* (Grassé, 1959). "In stigmergic labor, it is the product of work previously accomplished, rather than direct communication among nest-mates, that induces the insects to perform additional labor" (Wilson, 1971, p. 229). A controller for robots that relies on stigmergic communication is simpler because it does not have to care of the other robots and of the records and the analyses of the information they transmit. In order to emphasise the inspiration from biology, a system of such robots is sometimes called a *colony.* The approach paves the way toward the implementation of robotic systems that comprise a huge amount, a swarm, of individuals. The field of robotics that deals with swarms of robots with simple control based on stigmergic communication is called *Swarm Robotics*, which is part of the field of *Swarm Intelligence* (Bonabeau et al., 1999).

A swarm of robots programmed with swarm-intelligent techniques can theoretically perform as good as, or even better than, other systems. Its advantage is to be more robust in the case of failure of some elements. Simple behaviours also imply fewer assumptions about the environment and these systems result to be more robust to sudden changes. Anyway, *Swarm Robotics* has its drawbacks. The relationship between local and global behaviours is not easy to understand and, as it often occurs in complex systems, small changes at a local level might result in drastic and unpredictable changes at the global level. In this context, a good analysis and understanding of the dynamics of the system plays a crucial role.

Our work is a case study of a swarm-robotic system. We consider the costs and efficiency of foraging. Here, the word "cost" is used to describe anything that reduces the performance and the efficiency of the system. It can comprehend, for instance, conflicts between agents, energy consumption or the adversity of the environment. This study wants to show that a simple swarm-intelligent technique can improve the benefit/costs trade-off which governs the retrieval task of a colony of robots.

This work was carried out within the scope of the SWARM-BOTS project, a Future and Emerging Technologies project founded by the CEC, whose aim is to design new artifacts, called *s-bots*, able to self-assemble in a bigger structure called a *swarm-bot*. More details about this are given below.

Our experiments use real robots instead of software simulation. The latter is extensively used in several works in the literature and also inside the SWARM-BOTS project for the reasons explained above, but it has the disadvantage that it becomes unpractical when experiments need an accurate representation of physical constraints and interactions. Unfortunately, no *s-bot* was available to date. Therefore a simplified version made with Lego bricks was used to simulate the main *s-bot* features in a real environment. Instead of using a *digital* simulation, that is, a program running entirely on a computer, we used an *analogical* one, meaning that a simulated version of the *s-bots* was used in real world with real physics and real interactions.

This work is organised as follows: Section 1.1 describes the SWARM-BOTS project, its ground ideas and our contribution to it; Section 1.2 explains the aim of our study; Section 1.3 closes the chapter anticipating the results we achieved. Then, Chapter 2 reviews related work in the literature, Chapter 3 describes the hardware aspects of the work and Chapter 4 the implemented software. Chapter 5 details experiments and their results, and Chapter 6 ends with conclusions and future works.

1.1 The SWARM-BOTS Project

The aim of the SWARM-BOTS project is to develop a new robotic system, a *swarm-bot*, composed of several independent and small modules, called *s-bots*. Each module is self contained and capable of independent movements, but it can connect with other modules to form a *swarm-bot*. This process is intended to be self-organised and self-assembled in order to easily adapt to dynamic environments or difficult tasks. A task is considered difficult if it is not possible for a single *s-bot* to achieve it, thus it requires some form of collaboration. Examples of difficult tasks are pulling of heavy object or exploration on rough terrain, which could contain holes bigger than each *s-bot*. Collaboration is achieved by mean of indirect and non-symbolic communication, e.g. physical interactions through the environment and among *s-bots*.

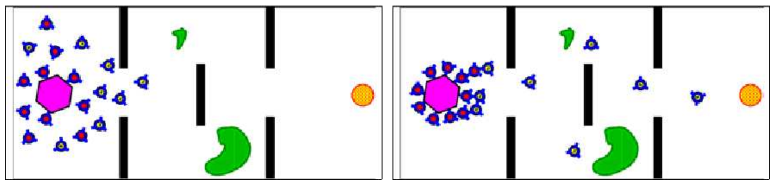
The SWARM-BOTS project aims to use techniques derived from *swarm intelligence*, the study of how collectively intelligent systems can be created by a number of simple autonomous agents. A *swarm-bot* is someway in between a *collective robotic system* and a *metamorphic robotic system*. The former is object of study in the field of *collective robotics*, the latter in *metamorphic robotics*. In collective robotics, the general issue is to understand the means by which groups of independent robots can interact to accomplish a given task. An overview of this field is given in Chapter 2. In metamorphic robotics, several modules must aggregate to be able to achieve a task. Usually, the modules are self-contained, but they need to be connected to other modules in order to move. The main research topic in this field is the achievement of a predefined shape starting from any configuration. Solutions to this problem use both centralised and distributed algorithms, but usually the target shape is determined *a priori* and is not task-related (Chirikjian, 1994; Yim, 1995; Pamecha et al., 1996; Kotay et al., 1998; Castano et al., 2000; Rus and Vona, 2001; Ünsal et al., 2001; Yim et al., 2001; Kamimura et al., 2001).

The main source of inspiration for the project is biology: several examples for self-assembling and self-organisation can be found in Nature (Camazine et al., 2001; Anderson et al., 2002). It is useful to present a scenario of a suitable task for a *swarm-bot* in order to better explain which are the main issues in the control design.

1.1.1 Imaginary scenario

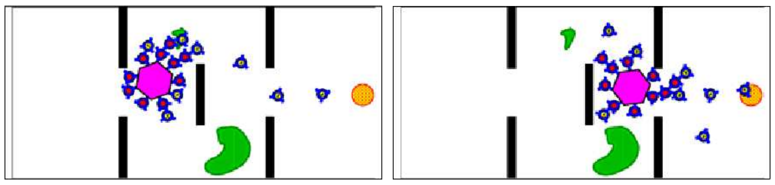
The following scenario is intended to describe the issues that the control system must deal with. It describes a hypothetical object-retrieving task, a task that requires co-operation and physical connections among *s-bots*. Two kinds of *s-bots* are in a rectangular arena: *explorers*, with fast motors and many sensors, and *carriers*, with powerful motors and fewer sensors. The object to be retrieved is the hexagon on the left side of Figure 1.1(a) and the circle on the right side indicates the location to which the object has to be moved (the *goal*). Walls are present too and hide the goal to the *s-bots*. The two other objects in the middle of each figure represent holes in the terrain.

When explorers find the object, they start signalling to nearby *s-bots*, which start clustering around the object. Carriers position themselves around the perimeter of the object, grasp it and try to push and pull till they can co-ordinate and move it in the right direction (Fig. 1.1(b)). In the meanwhile, explorers search for a path to the goal and signal it to the carriers (Fig. 1.1(c)). The last passage in front of the goal is too narrow for the present configuration of the *swarm-bot* and the *s-bots* have to rearrange their positions to pass through (Fig. 1.1(d)). Finally, *s-bots* can drop the target at the goal.



(a) *S-bots* looking for a target.

(b) The *s-bots* start to pull the object to the goal position.



(c) Selection of the best path.

(d) The *swarm-bot* reconfigures itself to pass through a narrow passage.

Figure 1.1: Imaginary scenario of a transportation task for a *swarm-bot*.

1.1.2 General issues

The general task depicted above includes the possible sub-tasks that the *s-bots* should be able to perform:

Dynamic pattern formation/change: *s-bots* must be able to self-assemble into a number of different configurations, choosing the most suitable one for the task and the environment in which they operate. They must also be able to change spatial arrangement in order to adapt to modifications in the environment.

Navigation on rough terrain: the *swarm-bot* must be able to move autonomously on rough terrain, using information gathered by single *s-bots*.

Pulling/pushing objects: the *swarm-bot* must be able to pull or push objects that are too heavy for a single *s-bot*. When collective pulling or pushing is required, *s-bots* must call for help and self-assemble taking care of the characteristics of the objects and the environment, e.g. the shape of the object or the presence of holes around it.

Task allocation: only some *s-bots* pull the prey while the others search for the way to the goal. The number of *s-bots* in each group should

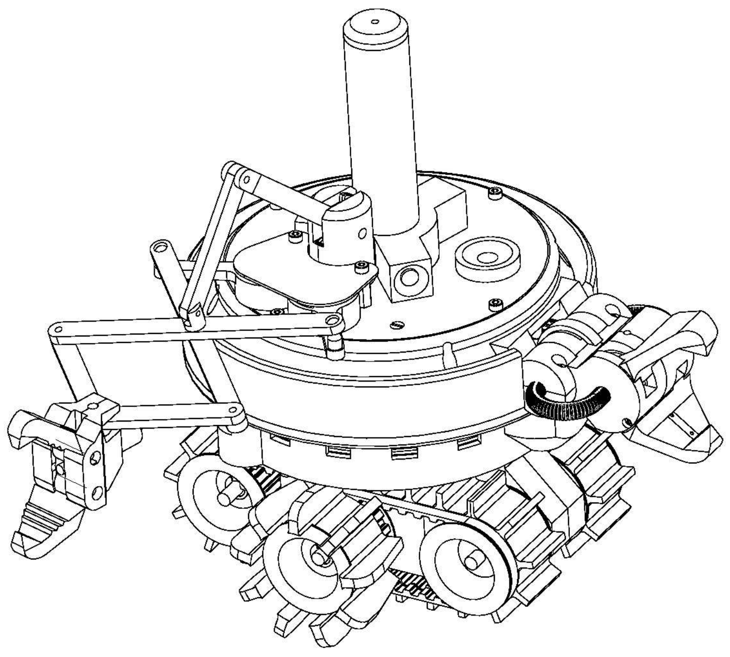


Figure 1.2: The design of an *s-bot*.

be adapted both to the weight of the object and to the adversity and complexity of the environment.

This work focuses particularly on the last issue. Considering that searching and retrieving is not costless, there is an optimal number of scout *s-bots* to be used that must be adapted according to the situation. Section 1.2 gives a more detailed explanation of this concept.

Only a sub-problem of retrieving is studied at the beginning, that is, when the object to collect is light enough to be retrieved by a single *s-bot*. This simplification facilitates the initial study of task allocation and resource optimisation. As a consequence, the following pages do not stress the self-assembly feature of a *swarm-bot*.

1.1.3 The *s-bot*

To date, only one prototype of the *s-bots* is available. This section briefly describes it in order to point out the features that the version with Lego bricks tries to simulate.

Figure 1.2 is the mechanical drawing of an *s-bot*, while Figure 1.3 depicts its real implementation. The *s-bot* uses *treels*©, a mix between tracks and wheels to address rough terrains. On the top there is an omni-directional

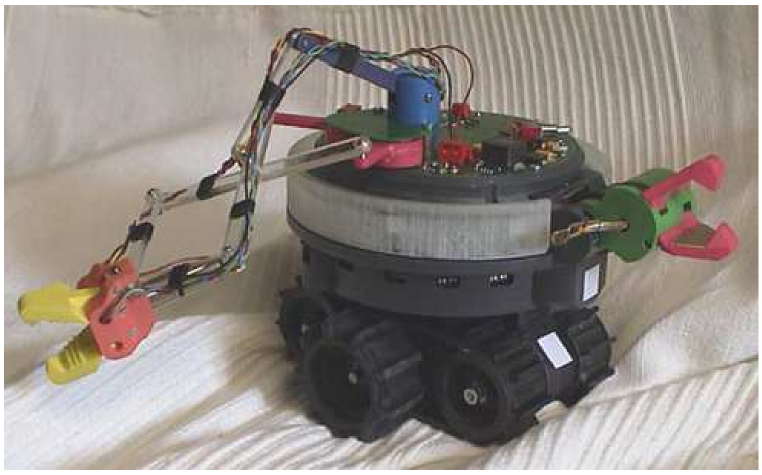


Figure 1.3: The hardware prototype of an *s-bot*.

camera that can be used to analyse the surroundings, looking for objects or obstacles. The *s-bot* has a fixed and a mobile grippers to provide both rigid and flexible connections. The body can rotate independently from the *treels*©. Infrared emitters and receivers, light intensity sensors and LEDs are all around the body to sense other *s-bots* or objects.

For the task of prey retrieval, tracks and a strong gripper are the most important elements: the first provides a good traction while the second holds objects tightly. The large number of sensors, the camera, the additional gripper and the rotating base can be useful for this task, but are unfortunately hard to replicate with Lego bricks. Chapter 3 describes how the Lego model simulates an *s-bot*.

1.2 The Aim of this Work

This work takes inspiration from retrieval in biological systems as, for instance, in a colony of ants. Each ant needs food to obtain energy to survive, but it also needs a secure place, the nest, to rest. If all the ants go and search for prey, nobody can take care of the defence of the nest or of feeding 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. If we look at the energetic consumption of the whole colony, these effects represent costs that decrease the net income of energy.

We present a small mathematical model to illustrate the problem more formally (what follows has the only purpose of clarifying the explanation and it is not intended to model our results). If X is the number of foragers in a colony, we can assume that the rate of discovery and retrieval of prey is approximately αXP, where P is the total amount of prey in the environment and α is the probability that one forager finds one prey in one unit of time (i.e. the discovery rate). 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 another insect that is walking through the area), although this is not the case in the experiments of Chapter 5. Prey dynamics can be modelled according to the following equation:

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

where ϕ represent 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:

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

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 by the colony, 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, to consider the influence of negative interactions between foragers, which increases with the square of their number, the second derivative must also be greater than 0. As a first approximation, $c(X)$ can be given by:

$$c(X) = \delta N + \epsilon X + \gamma X^2 \ ,$$

with $\delta > 0$, $\epsilon > 0$ and $\gamma > 0$. Figure 1.4 shows two possible curves for the income and the costs. The colony survives when the incomes are bigger than the costs. Therefore the colony can allocate foragers between a minimum and a maximum value. For values outside this range, the income of energy is less than the consumption.

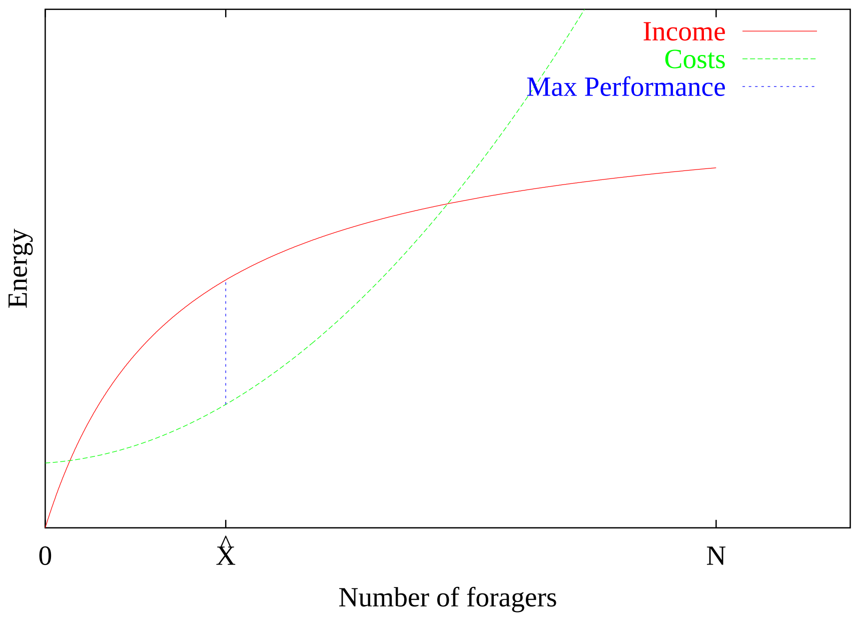


Figure 1.4: Relationship between the number of foragers and the energetic income/outcome in a colony.

There is an optimal number of foragers to be used that is given by

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

represented as a vertical line in Figure 1.4. $\hat{X}$ depends on conditions of the environment that can change during time, like the appearing and disappearing rates, or the presence of predators in the area.

Ants use a number of methods to reach and adapt this optimal value in case of dynamic environments. We can look at some of them to see if it is possible to implement them in a robotic system.

Evolution surely played an important role tuning parameters of ant behaviours. A solution based only on *evolutionary robotics*[1] is not robust with respect to changes in the environment faster than evolutionary dynamics. Moreover, experiments with this approach take a lot of time if they are performed on real robots, as we intend to do.

[1]Evolutionary robotics simulates biological evolution to find the best control program to achieve a given task. In a collection, called *population*, of different programs, each of them is tested to measure how good it achieves the task. The ones that can solve the task better in the current population are selected and simulated *recombination* and *mutation* are applied to them to create new programs and thus a new population. The process is then iterated on the new population. See Nolfi and Floreano (2000) for a more detailed explanation and examples.

Ants also exploit different forms of recruitment and stigmergic communication to adapt to a dynamic environment, for instance communicating the sudden appearance of a cluster of prey to nest mates. Unfortunately, these methods usually rely on chemical substances being released in the environment, and this makes it impossible to be implemented with robots as simple as the one we used (see Chapter 3).

In this work we investigate another way to reach the optimal value of foragers, or to approach it. Assuming that each robot can switch from a *resting* behaviour to a *foraging* behaviour in a probabilistic fashion, it is possible to modulate this probability in each individual according to its previous successes or failures. The foraging time cannot exceed a fixed timeout, after which robots come back to the nest. Each individual becomes a “tester” to estimate the quality of the environment. If, for instance, the environment is rich of prey easy to retrieve, each individual finds and retrieves prey whenever it is out of the nest and tends to rest little time by increasing its probability to leave. From the colony point of view, the mean number of individuals allocated to foraging increases. If the environment is poor, many individuals fail to find a prey and tend to increasingly rest in the nest. The mean number of foragers decreases. If there are too many foragers, some of them will not be able to find a prey and come back to the nest, decreasing the probability to leave. If they are too few, some nest-mate start foraging eventually with success, therefore tending to go out more and more often.

We argue that this simple individual learning algorithm can drift the colony toward the optimal point illustrated above. Moreover, the dynamics it creates can be used to allocate tasks to nest-mates thanks to the amplification of small random perturbations. Individuals that, because of luck or better ability, find more prey than others have a higher probability to leave the nest. Since they spend more time out of the nest, they find more prey. This positive feedback transforms them into foragers, while their nest-mates spend more and more time in idleness in the nest.

The next chapters describe the work done from the hardware and the software point of view to test these ideas. The focus is mainly on the dynamic allocation of an optimal number of foragers, and little on the retrieving behaviours themselves. Therefore, a simpler case is studied: solitary prey retrieval. In this framework, each forager is able to retrieve a single prey item, therefore no collaboration between individuals is needed. This situation is left as a future work in the SWARM-BOTS project.

1.3 Results

We anticipate here the preliminary results that Chapter 5 reports in more detail. We used a rule to modify the probability to leave the nest based on the number of successes or failures in a row. We tested it in a series of ten experiments[2] against a system where robots always exit from the nest with probability 1. The adaptation rule saves the colony's energy, resulting in a more efficient retrieval, without using communication among robots.

One would think of prey retrieval, as described in Section 1.2, as a multi-objective optimisation problem: the goals are, on the one hand, to increase the flux of incoming prey and, on the other hand to reduce energy consumption. The higher the number of foragers, the higher the flux of incoming prey, but the spent energy increases too. Our results suggest, on the contrary, that this is not always true. In some conditions, for instance when the negative interactions between robots are too strong, a decrease in the number of foragers improves the efficiency of retrieval without changing the flux of incoming prey.

The means by which the system performs efficiently is task allocation. Two classes of robots are present systematically in all experiments: *foragers*, which leave the nest with high probability, and *loafers*, which leave the nest with low probability. We also show that the task allocation process exploits differences among robots. Individuals that are better in retrieving are more likely to become foragers.

We do not claim that our method can find the optimal number of foragers, nor that the updating rule we use is the best one. We think that our approach is promising, but there is still much work to do. Possible future directions of research are listed in Chapter 6.

[2]The number of experiments was mainly decided by the reliability of the robots and by the time it took to complete them.

Chapter 2

Literature Overview

There has been an increasing interest in Multi Robot Systems (MRS) recently. Some of the reasons for their utility are:

> *... 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.)

Prey retrieval, also known as *foraging*, is among the different tasks that Cao et al. consider the canonical domains for MRS. It consists of searching for objects in the environment and bringing them to a region called *home* or *nest*. A retrieval task can be performed by each robot independently, and generally, the issue is whether or not multiple robots can achieve a performance gain.

In MRS, robots usually collaborate to achieve their task. There are also systems in which competition is important. For instance, in RoboCup,[1] two teams of robots (both simulated and real) must 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).

If we look at the field of *collaborative robotics*, i.e. where there is no competition among robots, a multi robot system can be classified according to different parameters as suggested by Dudek et al. (1996):

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

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 limited number n of robots. If n is big enough, then the group can be considered of infinite size.

Communication range. There can be no communication, communication limited to a certain distance or without limit. The latter means that each robot can communicate with every other robot independently from its distance. Limitless communication it is a property that depends both on the characteristic of the environment and on the means of communication.

Communication topology. Robots can use broadcasting techniques, address based messages or tree hierarchies to spread information. The communication topology can also have the form of a general graph. This is more robust to failures than a tree structure because of the possible presence of multiple paths between nodes of the graph.

Communication bandwidth. There can be costs related to communication which affect the available bandwidth. It is possible to have no costs and infinite bandwidth, high costs and low bandwidth or no communication at all because costs are prohibitive. The latter is considered to be an "impractical case if coordinated collective behaviour is desired" (Dudek et al., 1996, p. 381). Dudek et al. identify another category, which lies between between infinite and low bandwidth, that is more interesting for our work. The costs of communication for this category have "the same order of magnitude of the cost of moving the robots between locations" (Dudek et al., 1996, p. 381) and includes stigmergic communication.

Collective reconfigurability. It takes care of spatial arrangements of the groups. Robots can have fixed positions, change occasionally following communication or have completely dynamic locations.

Processing abilities. The control program of each robot can be implemented as a neural network, a finite state automaton, a push-down automaton or a Turing machine.

Collective composition. Units of the system can be identical both in form and function, homogeneous (same physical characteristic) or heterogeneous.

The system studied in this work has a limited size, although the aim is to eventually use a *swarm*. It uses no (direct) communication and therefore no particular communication topology. Costs are given by the movements of robots. The system is dynamically reconfigurable and composed of identical units. At the high level, the control system can be modelled as a finite state automaton (Section 4.2).

The following pages briefly review some of the issues addressed in MRSs (Section 2.1) and in swarm robotics (Section 2.2), focusing mostly on work related to foraging. The last section (2.3) describes what is known in biological systems about prey retrieval.

2.1 Multi Robot Systems

The task of the control system of a robot 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 state. The goal state can be to have an object carried from one place to another, to have the robot in a fixed position, to move it 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.

If the system is composed of 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 (*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 read the same data, 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 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* (Sutton and Barto, 1998).

Real time requirements. The product between state and action spaces 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 must operate in a critical environment where decisions must be fast or where fast learning is desired.

In the case of MRSs, these problems are amplified. Each robot must take care of the others. From point of view of the single 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.

In order to deal with changing and partially observable environments, Goldber and Matarić (2000) use several Augmented Markov Models (AMM), a kind of Markov Chain that is incrementally generated through node splitting in order to catch hidden states. Each AMM tracks a different timescale. The algorithm is tested on a foraging task, but only one robot is used to evaluate its performance.

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. After some time, the robot forgets what it has seen. Finally, a third reward is given by other agents in order to share reinforcement when involved in social interactions.

Hayes (2002) directly addresses the problem of increasing the efficiency providing an analytical model to find the optimal number of foragers to use when using both random and coordinated search. The author shows that coordinated search gives better results. The model is limited for our purposes because it uses the following hypotheses: there is only one item to search, the environment is not changing in time and the probabilities of each agent to succeed are fully independent. The last one is the most restrictive hypothesis because it implies that robots do not interact with each other. We will show in Section 5.4 that this is not true.

Balch and Arkin (1994) analyse the effects of communication in 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 low-level 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.

In most of the work done in the literature, the design of the control system is centred on the perspective of a single robot. The group and its dynamics are considered mainly to evaluate performance. Researchers acknowledge that there are negative interferences in a group of robots, but their solutions are mostly based on making the individual control system more complex. The following section discusses work in which the control system tries to exploit the interactions in the group to accomplish its task.

2.2 Swarm Robotics

Control algorithms in swarm robotics are very simple and do not use complex representations of the environment or of the other robots. 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.

In Jones and Matarić (2002), robots must 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.

Melhuish et al. (2001) use *patch sorting* as a test bed for their system. 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.

Holland and Melhuish (1999) 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. There is not enough space here to list all of them and they are less important in the context of our work. The interested reader is referred to the original paper.

Agassounon et al. (2001) and Agassounon and Martinoli (2002) address the problem of task allocation in a colony. The task is not the retrieval but 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 it 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 fixes 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}. Although their work might appear similar to our, we see some differences among them. We discuss this topic in Chapter 6, after having described our setup and methodology.

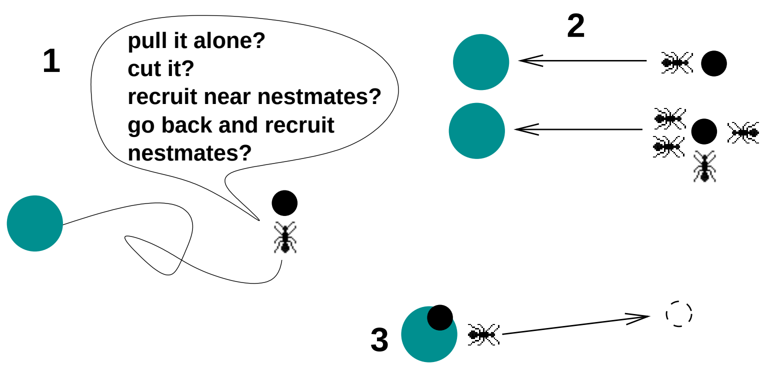


Figure 2.1: Foraging and retrieving behaviour of ants. An ant performs a random walk in the environment until it finds a prey. It takes some decisions about how to retrieve it. Transport can be done alone or in group. Once the prey is in the nest, the ant goes back directly to where the prey was found.

2.3 Prey Retrieval in Biology

A wide range of different foraging behaviours is observed during prey scavenging and prey retrieval (or building material), but the mechanisms governing the emergence of them remain unclear. Figure 2.1 sketches the general behaviour of ants. When they find a prey after having randomly explored the environment, they take decisions following this schema:

- 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 the place with their jaws and retrieve smaller pieces.

The retrieval can be done solitarily 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 any measure of 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.

Division of labour and adaptation to changes in the environment are often observed in ant colonies. Deneubourg et al. (1987) explain them with a simple probabilistic model. The environment is discretized into zones. Each ant has a given probability, for each simulated time step, to forage and, when it does, it chooses the zone where to go according to a given probability distribution. If an ant is successful in finding a prey, it increases by a fixed value the probability to leave the nest and to return to the same point where it was before. If it is not successful, then the probabilities are reduced by the same value. Monte Carlo simulations of this model gave results similar to the one observed on the field.

Chapter 3

Hardware

This chapter gives the details of the hardware implementation for the robots used to test the hypothesis illustrated in Section 1.2 and for the results described in Chapter 5.

Real robots have the advantage that it is not necessary to simulate their dynamics and their interactions in detail, which is the bottleneck of any realistic simulation. Dynamics is considered important because interactions among robots create those constraints that a control program must face to improve the performance of the system. Simulated robots also have the disadvantage of being identical, unless differences are intentionally integrated in the simulation, for instance in the form of noise. Two real robots are never the same. Sensors have different sensibilities and responses, the mechanics may be weaker in one of them or their speed can be different. Usually, this is something that makes the control system more complex, since it has to cope with different conditions. A swarm-intelligent solution does not tackle these problems directly, but lets the dynamics of the system solve them. Some of the robots could be mechanically better to retrieve objects, and therefore they will be more successful than others. The positive feedback explained in Section 1.2 allocates them to the retrieving task, while the unsuccessful ones stay more frequently in the nest. However, simulation has the big advantage of being faster and simulated robots being mechanically more reliable for experiments.

The actual *s-bot* is not used in this work because it was still in the prototyping phase at the time of writing. Therefore, we employed a simplified version of an *s-bot*, made with Lego bricks, which simulates some of its features. We call our artifacts made from Lego bricks *MindS-bot* (Figure 3.1). 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. From the mechanical point of view, the plastic used by Lego is not the best solution in order to obtain reliable artifacts because of the elasticity of some of its components.

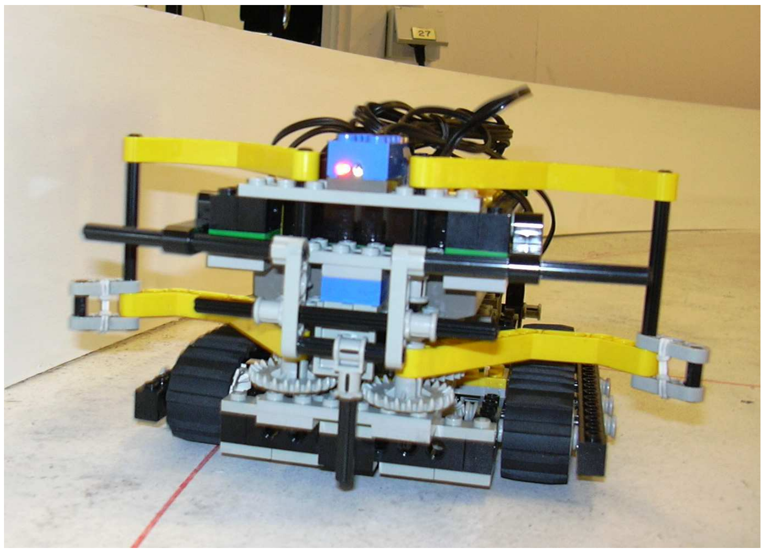


Figure 3.1: A picture of a *MindS-bot*. See Section 3.3 for details.

Section 3.1 describes the hardware characteristics of Lego Mindstorm^{TM}, the line of Lego products which was used to build the *MindS-bots*. Section 3.2 describes the main prototyping issues. Section 3.3 describes a *MindS-bot* in details.

3.1 Lego Mindstorm^{TM}

Lego Mindstorm^{TM} is the line of Lego products used to prototype and build the *MindS-bots*. A box usually contains pieces like the ones that can be found in a Lego Technic^{TM} box, that is, normal squared bricks, axles, gears and joints of different types and sizes, plus some electric and electronic components. A box of Lego Mindstorm^{TM} also includes a microprocessor embedded in a block, called *Brick* or *RCX* (Figure 3.2).

The Brick contains an Hitachi H8300-HMS 1 MHz microprocessor and slots for six AA batteries, 1.5V each. It has 32Kb RAM and a ROM onboard. On the front side, there is an infrared transmitter/receiver to communicate with a computer, with other Bricks or to receive commands from a remote controller. A program in the ROM takes care of downloading and executing the operating system from a base computer by means of the infrared port. The operating system runs user's programs and handles hardware interrupts. In this work, only communication with the computer is used both to download the program on the robot and to upload logs recorded by the *MindS-bots* during each experiment.

A 5-digits LCD display is on the top of the brick. Four keys and slots for three input and three output devices are situated around it. The out-

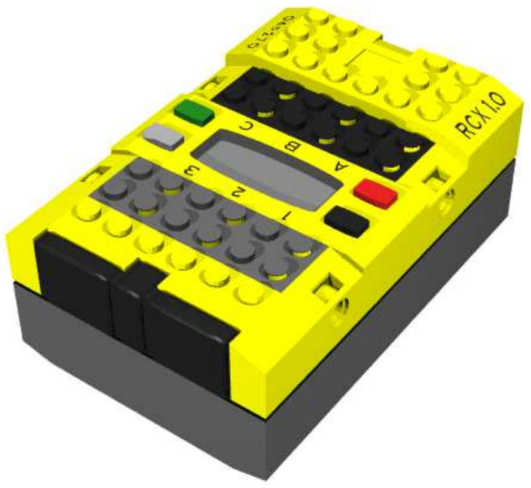


Figure 3.2: A picture of the Brick included in each Lego Mindstorm^{TM} box. The infrared transmitter is in the front. On the top there are an LCD display, four keys, three slots for sensors (the front grey stripe) and three slots for actuators (the back black stripe).

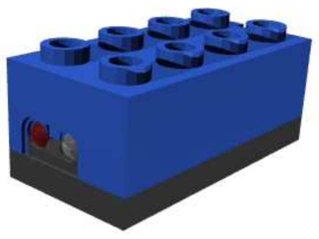
put connections provide the power for the actuators, like motors or lights. The input connections are used to sample and digitise, with a ten bit A/D converter, the signals coming from the sensors. They also provide power for those sensors that might need it, e.g. proximity sensors that emit infrared pulses to be reflected by an obstacle. If a sensor needs power in order to work, it is called an *active* sensor, otherwise it is called *passive*. More than one device can be connected to one slot, but the resulting signal is not the sum of all the individual ones. In fact, the devices are connected in parallel from an electronic point of view.

Lego provides two kind of actuators: 9V motors (Figure 3.3(a)) and white lamps. The variety of sensors is wider: there are rotation sensors (Figure 3.3(b)—can be used only in active mode), light intensity (Figure 3.3(c)—used both in passive and active mode) and touch sensors (Figure 3.3(d)—only passive mode). When a light intensity sensor is used in passive mode, the value it returns is simply the intensity of the light in the environment. Readings are stable and change only if a strong light source is in front of the sensor. When it is in active mode, a small red LED in the front is switched on and the sensor reads the reflected light of any object in the front. These readings are more noisy, but the sensor can reliably detect black objects if they are approximately 15-20 cm away, depending on their size.

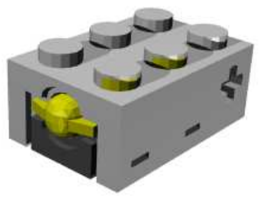
There is also a number of additional sensors and actuators to be added

(a) 9V motor.

(b) Rotation sensor. It measures the rotation of an axle that is inserted in the hole on the side.



(c) Light sensor. On the front there are a red LED and the actual sensor.



(d) Touch sensor. It is a simple switch that turns on when the front button is pressed.

Figure 3.3: Pictures of sensors and actuators available in the Lego Mindstorm^{TM} product line.

to the list, which are sold on the Internet but not provided by Lego.[1] Among these, there are temperature, PH, humidity, colour and proximity sensors, accelerometer and coloured lights. Most notably, there are also multiplexers both for sensor input and for motor output that allow to overcome the limitations imposed by the Brick.

3.2 Mechanical Issues

This section describes the mechanical issues that one must consider when designing a robot for prey retrieval. We illustrate in Section 3.3, when we

[1]For instance:
`http://www.mindsensors.com/`, `http://www.hitechnicstuff.com/products.htm` and `http://ex.stormyprods.com/lego/`

describe the *MindS-bot*, the solutions we found to these problems.

For the achievement of a prey retrieval task, a *MindS-bot* must be strong enough to pull a prey. Pulling is more often observed in nature than pushing. It simplifies the control system because a *MindS-bot* can grasp and hold one prey on one side, and look for the nest on the opposite side. If pushing occurred, the *MindS-bot* would not be able to see the nest because the prey would occlude most of the visual field. Therefore, the *MindS-bot* should somehow remember the position of the nest, for instance by means of a map that should be continuously updated.

The circular shape of the *s-bot* (Section 1.1.3) is hard to replicate with squared Lego bricks. We dropped the idea of simulating a circular body in order to have a compact volume and sizes comparable to a *s-bot*.

Recognition of the nest, prey, walls and other nest-mates is essential for the achievement of the task. We chose to use black prey in a white environment and a light to signal the position of the nest. A combination of short and long range sensors is needed on at least two opposite sides, if pulling is what we want. Therefore, light and touch sensors are essential for our task. This sums up to at least four sensors, but there are slots for only three of them.

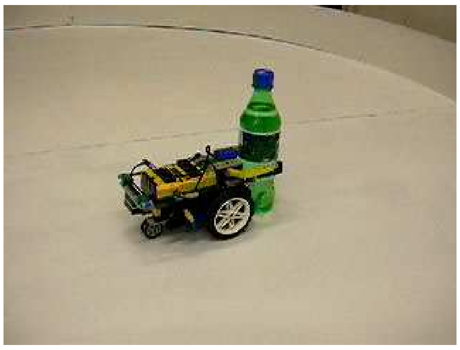
The gripper is the last issue we highlight. It must be able to hold a prey tightly. Lego motors can be short-circuited to brake their rotation, but this solution must be avoided because it is too energy consuming and batteries would not last long enough. A mechanical solution is better. Figure 3.4 gives an example of what could happen if the gripper was too weak.

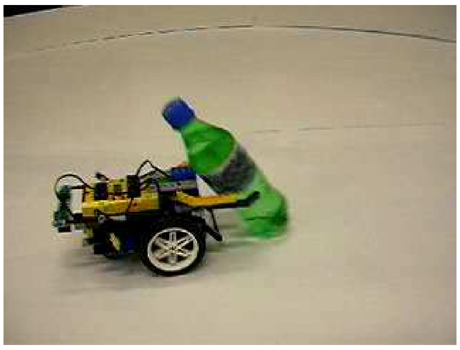
3.3 The *MindS-bot*

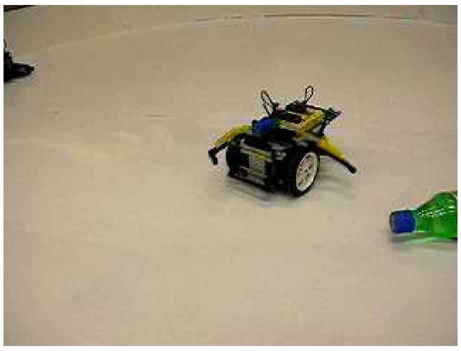
This final section is dedicated to the description of the *MindS-bot* used for the experiments. Figures 3.1 and 3.5 are pictures of a real *MindS-bot* and a computer generated draw. Figure 3.6 shows lateral views of a *MindS-bot* to highlight some details. In the following, there is a description of its main part and of the solutions to the problems discussed in Section 3.2. Appendix A reports the list of pieces to use and gives instructions on how to reproduce a *MindS-bot*.

3.3.1 Traction

A *MindS-bot* uses tracks to move on the ground (Figure 3.7). They offer a good compromise between traction power and space compactness. A first trial used big wheels to achieve a stronger traction to pull a prey, but the gripper was placed to high and the hold was not strong enough (the result is shown in Figure 3.4). 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 3.7. Low


(a) An early prototype of a *MindS-bot* has grasped a prey and is ready to pull it.


(b) The friction of the prey with the ground is too high for the gripper.


(c) The momentum that the prey applied to the gripper's arms was strong enough to break them.

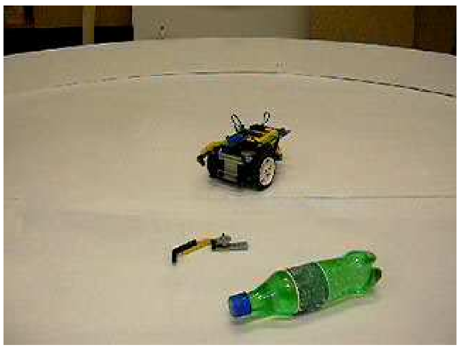

(d) The *MindS-bot* leaves with its remainders left on the battlefield.

Figure 3.4: Example of a weak gripper.

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.

A reduction between the motors' and the tracks' gears increases the traction power, although it slows down the *MindS-bot*. However, its speed is still about 12 cm/sec, which is considered enough for the experiments.

3.3.2 Gripper

Two arms, placed symmetrically with respect to the centre of the robot and actuated by the same motor, form the gripper (Figure 3.8). The movement is controlled by a motor, whose pulley is connected by an elastic ring to an axle

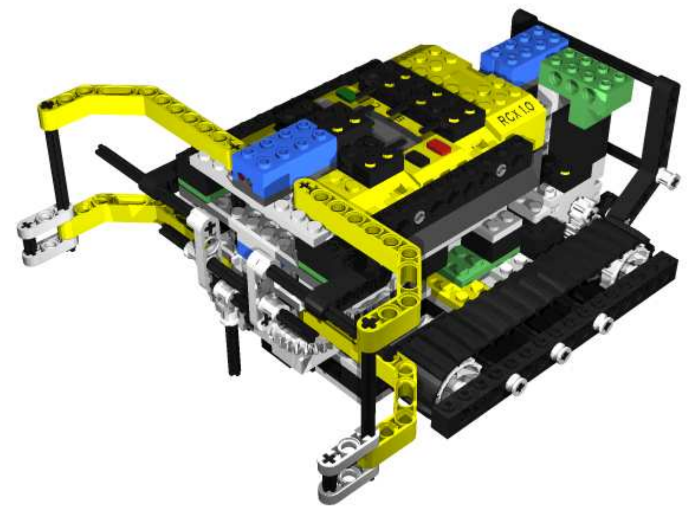


Figure 3.5: Computer generated view of a *MindS-bot*.

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 work 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. Figure 3.9 depicts the extension of the gripper.

The series of reductions, especially with the endless screw, allows the gripper to be strong and fix in a position without the need of short circuiting the motors, thus avoiding a waste of energy. However, the arms are not completely blocked because of the mobility between gears. Having more gears in a series amplifies this phenomenon, which has some effects when pulling a prey. Sometimes a *MindS-bot* can lose a prey while pulling it. This is not a problem in the framework of this work because it can be seen as an increase of the cost function described in Section 1.2. The colony compensates individual flaws by tuning the number of foragers.

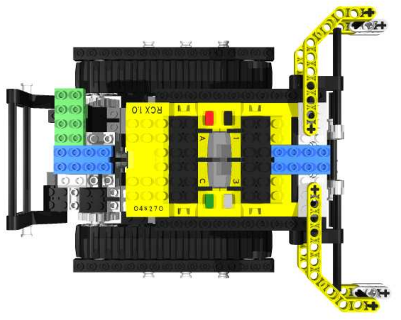
The rubber connecting the two pulleys is also used to prevent damages to the motor. If the gripper is stuck and cannot move, the rubber will slide over the pulley, allowing the motor to rotate freely.

3.3.3 Sensors

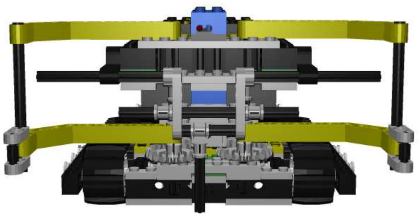
MindS-bots perceive the environment by means of two light sensors and two bumpers. The front light sensor (the blue brick on the top-right of Figure 3.10) is used in active mode to sense a prey. The back light sensor (top left of the same picture) is in passive mode and is used to search for the nest, identified by a light source. The front and back touch sensors are activated respectively by the movement of the back lever and by a pressure on the front whisker.

The back bumper and the front light sensor share the same input slot on the Brick. The control system can recognise if the value it reads comes from the touch or the light sensor. The former basically works like a switch, giving either no readings or the maximum value.[2] On the other hand, the latter's readings can reach neither the minimum nor the maximum value. Therefore, if the back bumper is not pressed, the value read by the control system corresponds to the value given by the light sensor. If a hit occurs, the read value is the maximum one. Finally, the back bumper is useful only when a *MindS-bot* is moving backward. The control system the *MindS-bot* moves backward only when to go back to the nest, with or without a prey. In these conditions, readings from the front light sensor are not needed.

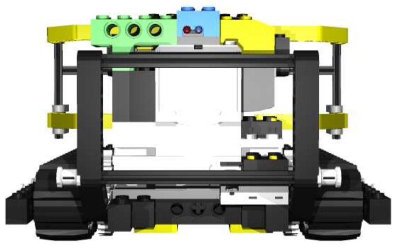
[2]The touch sensor is actually a variable resistor, controlled by the pressure on the yellow frontal button visible in Figure 3.3(d). It is very sensitive to small pressure, giving nearly immediately the maximum reading—corresponding to a short circuit in the sensor.



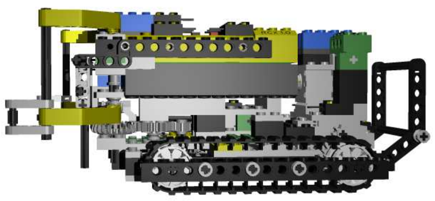
(a) Top view of a *MindS-bot*. The blue bricks are light sensors. The gripper arms are half opened to have an idea of the grasping area. The front bumper takes less space on the horizontal axis than the back one in order to permit an easier grasping of prey.



(b) Front view of a *MindS-bot*. The front light sensor is on the top. The whiskers of the bumper occupy nearly all the area to detect possible collisions more easily.



(c) Back view of a *MindS-bot*. This part is mainly covered by the back bumper for collision detection when moving backward. Behind it, there are the three motors for the tracks and the gripper. On the top there is a light sensor.



(d) Side view of a *MindS-bot*. Tracks do not occupy vertical space that can be used by the gripper's arms when it is open.

Figure 3.6: Details of a *MindS-bot*.

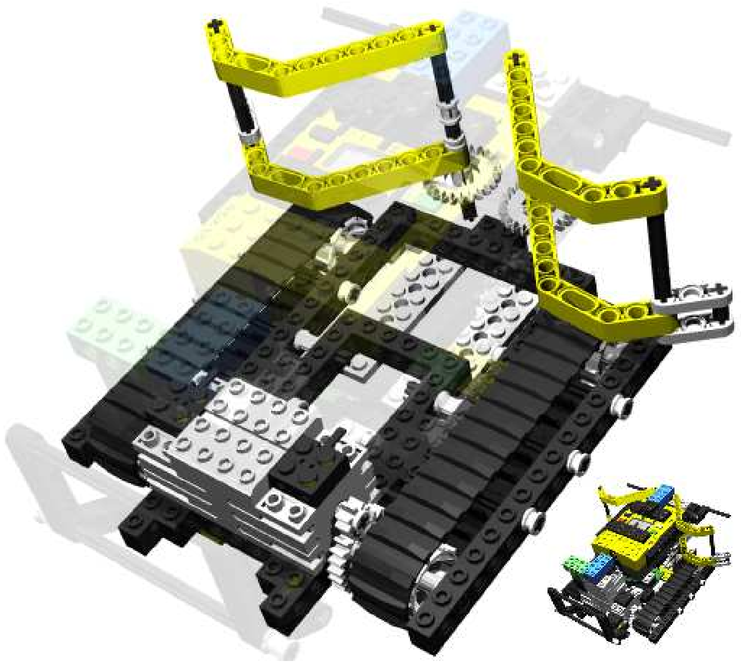


Figure 3.7: Details of the traction system. The gripper's arms can be placed over the tracks when the gripper is open. A small reduction in the transmission between the motors and the tracks allows for a stronger traction at the cost of a slower speed. The parts of the robots that are not of interest are transparent. On the bottom right corner, a small picture shows the perspective of the *MindS-bot*.

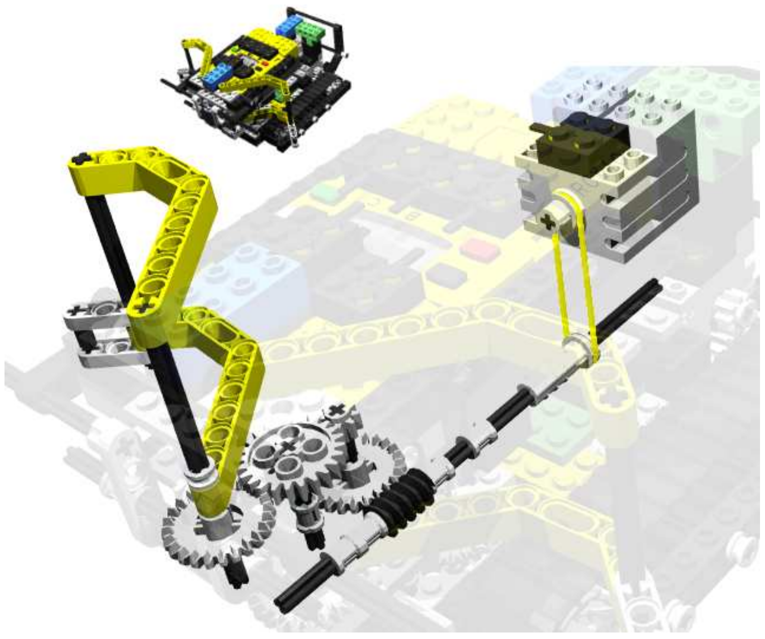
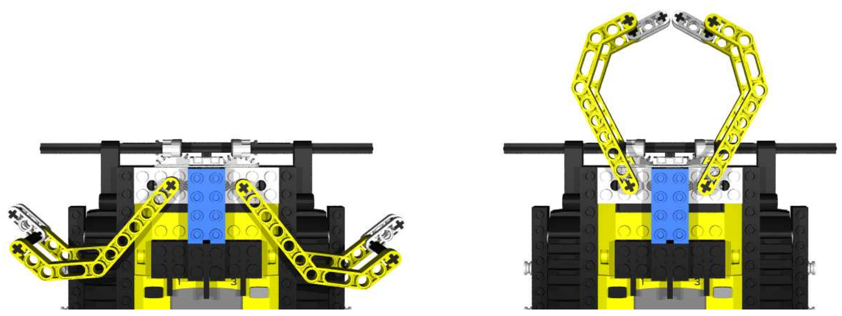


Figure 3.8: Details of the gripper. The series of gears create a reduction which is strong enough to block the gripper without short-circuiting the motor. The parts of the robots that are not of interest are transparent. On the top, a small picture shows the perspective of the *MindS-bot*.



(a) Open gripper.

(b) Closed gripper.

Figure 3.9: Spatial extension of the gripper.

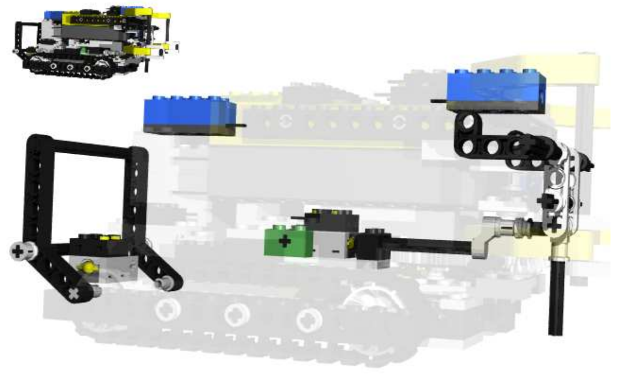


Figure 3.10: Details of the sensors. Light sensors are on the top to reduce problems with the shadows of other *MindS-bots*. The back touch sensor is pressed when the back lever is pushed against the *MindS-bot*. When the front whisker is pressed, an axle connected to it pushes the button of the touch sensor that is placed under the Brick. The parts of the robots that are not of interest are transparent. On the top left corner, a small picture shows the perspective of the *MindS-bot*.

Chapter 4

Software

Control systems used in swarm robotics are usually less complex because a number of programming techniques do not need to be implemented. There is no environment mapping, no inter-robot communication, no sensor analysis, no data fusion and so forth. The control system still has to face basic problems like searching for an object, avoiding obstacles, searching for the nest and retrieving objects. The following pages describe how the software that controls a *MindS-bot* deals with these subtasks.

Section 4.1 illustrates the features of the operating system that runs on the Brick. Section 4.2 illustrates the high level structure of the control and Section 4.3 ends describing the implementation of each behaviour.

4.1 BrickOS

A Brick (Section 3.1) runs the default program in ROM memory when it is switched on for the first time. The program waits on the infrared port for a firmware, also known as the *operating system*, to be downloaded. A Lego Mindstorm^{TM} box originally comes with a firmware provided by Lego, which is unfortunately designed for entertainment only. Its biggest disadvantage is that user programs can contain no more than thirty-two variables. Moreover, programs are compiled in a byte-code that is *interpreted* on the Brick by the firmware, which slows down execution time.[1] Lego's software does not provide other features that are typical of many operating systems, such as multitasking or resource locking.

Some people have overcome these limitations by writing their own operating system. It is possible to find free software for Lego Mindstorm^{TM} based, for instance, on Perl or Java.

Among different possibilities, we have chosen to use *BrickOS*.[2] A big

[1]Recall that the microprocessor, as explained in Section 3.1, works at a frequency of 1MHz. Interpretation time becomes crucial when facing dynamic environment.

[2]`http://brickos.sourceforge.net/`

advantage of this software is that it uses the original machine code of the Hitachi-H8300-HMS microprocessor also for the execution of programs. It is possible to write programs in any programming language and compile them with a compiler that supports the right machine code. GCC,[3] the most commonly used compiler within Linux and Unix distributions, is already configured for the H8300-HMS.

BrickOS is a POSIX-like operating system, with limitations mainly imposed by memory requirements. BrickOS uses between 14Kb and 18Kb of memory, depending on the options that the user wants to use. The rest (18–14Kb) is left to user programs. Moreover, BrickOS implements the following features:

- priority-based preemptive multitasking;
- process synchronisation with POSIX semaphores;
- CPU power saving;
- complete LCD control for debugging;
- key input handling;
- a random number generator inspired by Press et al. (1992, pages 279–281);
- infrared communication protocols that implement:
  - undirected broadcasting of messages
  - UDP-like directed communication based on address/port.

The communication protocols implemented in BrickOS do not guarantee that recipients receive the messages. Two robots (or one robot and one computer) must face each other with their infrared ports in order to communicate and the control system has to make sure that this situation takes place. This condition cannot be guaranteed while robots are moving. BrickOS' job is just to send packets and to guarantee that, when a packet is received, it is not corrupted. The following section describes a method to overcome this problem in a simple situation.

4.1.1 Communication with the Base

BrickOS functions for communication with other robots or with a computer take care only of the correctness of messages. If a message is received corrupted, the operating system drops it. BrickOS does not ensure that a message is received. As the *MindS-bots* themselves log data about each experiment, it is indispensable that all the logs are uploaded on a computer at

[3] `http://gcc.gnu.org/`

the end of each run. Moreover, it was noticed that only packets up to eight bytes can be successfully sent without corruption. Therefore, the control program needed a small wrapper around BrickOS functions to ensure the delivery of a message.

The solution is simple if it is considered that *MindS-bots* must communicate with the computer only at the end of each experiment and that communication is limited to one direction: only one *MindS-bot* transmits data and, for each packet sent by a *MindS-bot*, the computer sends back an acknowledgement. After sending, the *MindS-bot* waits 1 second for the acknowledgement that the computer sends after the packet was processed. If there is no answer, e.g. the packet is lost, the *MindS-bot* repeats the transmission up to 10 times.

The first transmitted packet is always 1-byte long. It contains a number that identifies the format and the order of the following data. The first packet allows the receiving program to select the right procedure to correctly parse the incoming data. Usually a number of constants used during each experiment are sent first, then the total number of log records and finally the records themselves. These include an integer to identify an event (exit from the nest, retrieval of a prey, or timeout reached) and the elapsed time from the beginning of the experiment when it occurred.

4.2 Control Architecture

Figure 4.1 gives a finite state machine abstract representation of the control program of the *MindS-bots*. 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 *MindS-bot* looks for a prey. It has to avoid collision with other *MindS-bots*. If a prey is found, the *MindS-bot* grasps it. If it has spent too much time searching a prey without finding any, it gives up.

Retrieve The *MindS-bot* looks for the nest and pulls a prey into it.

Deposit The *MindS-bot* 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 *MindS-bot* looks for the nest and returns to it.

Rest The *MindS-bot* 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 Figure 4.1 show the conditions (called also *predicates*) that must be true for the transition to occur. The complete list of the predicates and their meaning is given in

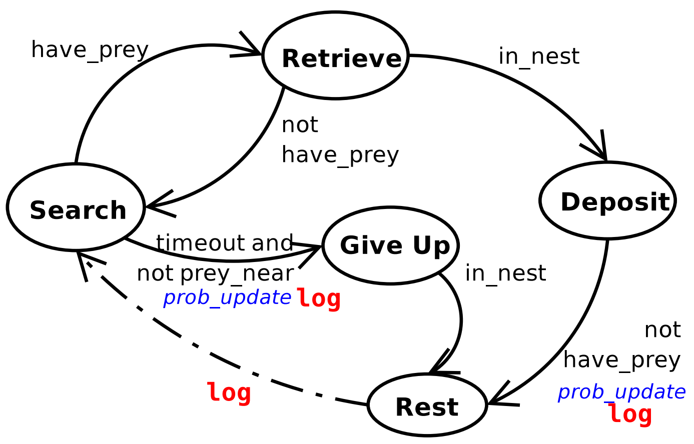


Figure 4.1: Finite state machine that describes the transitions between the states in a *MindS-bot*. The labels on each edge represent the predicates that let the transition occur whenever they are true. Their meaning and definition are given in Table 4.1. The label "*prob_update*" shows when the probability to leave the nest is updated. The label "**log**" shows which transitions are logged for the analyses of the experiments. The edge between **Rest** and **Search** is dash-dotted to indicate that the transition occurs probabilistically.

Table 4.1. Their truth values are evaluated from raw sensor readings every 100 ms, i.e. in each control cycle (see below).

The transition between **Rest** and **Search** is probabilistic. It is the purpose of our work to show that the adaptation of the probability of this event can improve the efficiency of prey retrieval. Probability updates occur during the transitions from **Search** to **Give Up** (in case of failure) and from **Deposit** to **Rest** (in case of success). Section 5.1 explains how the probability changes during these transitions.

The *MindS-bot* uses a number of behaviours, i.e. sub-procedures that directly react to sensor input, 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 one whose activation condition is true is executed. This architecture is known in the literature as *subsumption architecture* (Brooks, 1991).

Each state uses a restricted set of behaviours from the pool that we developed. They are listed in Table 4.2 together with the list of their activation conditions. The order in which the behaviours appear also reflect

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 half table) or of conditions (bottom half). 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
P_T	front light sensor reading when a black object is 20 cm away (*P_T*>*B_T*)
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*
`front_bumper`	the front bumper is pressed
`back_bumper`	the back bumper is pressed
`gripper_close`	the last command issued to the gripper motor was to close it
`gripper_open`	the last command issued to the gripper motor was to open it
`prey_near`	front light reading $\leq$ *P_T*
`prey_in_gripper`	front light reading $\leq$ *B_T*
`have_prey`	`gripper_close` $\wedge$ `prey_in_gripper`
`hit_prey`	`front_bumper` $\wedge$ `prey_in_gripper`
`timeout`	time elapsed from the beginning of the search phase > *T_L*

the hierarchy among them. The description of each behaviour is given in Section 4.3.

The control program exploits the multitasking feature of BrickOS. Each behaviour is implemented as a standalone process that takes care of reading the input and controlling the motors. An arbitrator process controls both state switching and behaviour activation. If the arbitrator detects that the activation condition of a higher-level behaviour is true, the currently running process is killed and the new one is started. Both arbitrator and behaviour processes repeat their execution every 100 ms. BrickOS does not provide real-time features, therefore, the period is not always precise and depends on the scheduler of the operating system. However, we consider the precision sufficient for our experiments.

Finally, there is a main process that controls the arbitrator. Its task is just to wait for the user's start and stop command and to upload logged data onto a computer.

Table 4.2: List of behaviours used by each state. 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. 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	`front_bumper` $\wedge \neg$`have_prey` $\vee$ `back_bumper`
	OpenGripper	`gripper_closed` $\wedge \neg$`prey_in_gripper`
	Explore	
Give Up	*AvoidObstacle*	`back_bumper`
	SearchNest	
Retrieve	*AvoidObstacle*	`back_bumper`
	SearchNest	
Deposit	*LeavePrey*	`have_prey`
	SearchNest	
Rest	*ExitFromNest*	

4.3 Behaviours

The previous section introduced the behaviours implemented in each *MindS-bot*. Their names are enough to understand *what* they do. In this section it is explained in more detail *how* they work. After this section, the reader should have enough information to re-implement the control algorithm in order to replicate our experiments.

4.3.1 AvoidObstacle

AvoidObstacle tries to avoid an obstacle after the *MindS-bot* hit it. It first checks the state of the front and the back bumper. If only the back bumper is pressed, the *MindS-bot* moves forward at maximum speed for 500 ms.[4] If only the front bumper is hit, the *MindS-bot* moves backward at maximum speed for 500 ms, and then rotates with equal probability either to the right or to the left for 500 ms. Rotation was introduced to avoid dead locks. In fact, other behaviours start by moving the *MindS-bot* forward. If they were executed after *AvoidObstacle*, the *MindS-bot* would hit the obstacle again and again.

[4]This value, as all the other given in this section, was hand tuned. In 500 ms, the *MindS-bot* can cover approximately one third of its body length, which is enough to freely turn.

Finally, if both bumpers are pressed, the motors are switched off and the *MindS-bot* stays still.

4.3.2 CloseGripper

This behaviour first switches off the track motors, then activates the gripper motor for 9 s. This time is enough for the arms to completely close and to tighten the hold. At the end, the new status of the gripper is registered into memory.

4.3.3 OpenGripper

The *OpenGripper* behaviour works exactly like *CloseGripper* except that the gripper motor turns in the opposite direction, thus opening the arms. At the end the new status of the gripper is registered in the memory.

4.3.4 Explore

The *Explore* behaviour is somewhat more complex than the others because it deals with all the aspects of searching and finding a prey. It can be split in three sub-behaviours: *randomWalk*, *searchNear* and *gotoPrey*. Its execution always starts with *randomWalk*: a random direction is chosen by turning left or right with equal probability for a time randomly distributed between 1 and 3 seconds. Then the *MindS-bot* moves straight on. The upper limit for the rotation time allows the *MindS-bot* to turn nearly 180 degree.

The behaviour randomly changes from *randomWalk* to *searchNear*, in which the *MindS-bot* turns on the spot checking with the front light sensor if there is a prey, that is, if `prey_near` is true. The transition has a probability such that the expected covered distance before *searchNear* starts is 3/4 of the arena diameter. If more than 10 seconds are spent without finding anything, *randomWalk* restarts.

Finally, *gotoPrey* is executed to move toward a prey whenever there is one in front of the *MindS-bot*, that is when `prey_near` is true. If the prey is lost, *searchNear* is again executed to explore the neighbourhood.

4.3.5 SearchNest

SearchNest follows the gradient of light intensity with the back sensor. At the beginning of its activation, *SearchNest* turns the *MindS-bot* on the spot and searches the maximum light intensity with the back light sensor, used in passive mode (Section 3.1). Then, it moves backward and checks that the readings of the back light sensor are not decreasing. This would mean that the *MindS-bot* is not heading toward the nest and that *SearchNest* must start from scratch.

4.3.6 LeavePrey

LeavePrey turns the *MindS-bot* clockwise for 8,75 s, so that it is oriented w.r.t. the centre of the nest with an angle of 90°, and then opens the gripper, thus releasing the prey. It must be noted that this behaviour is executed only if the *MindS-bot* has gripped a prey, so the gripper is always closed when *LeavePrey* starts (see Table 4.2).

4.3.7 ExitFromNest

The *ExitFromNest* behaviour does not move the *MindS-bot*, but selects a random number between 0 and 1 each second. If it is below the current probability to leave the nest, it signals to the arbitrator module that the transition to the **Search** state can occur.

Chapter 5

Results

This chapter describes the experiments and the results we obtained to validate our hypotheses, which are here summarised:

Efficiency: foraging efficiency improves in a colony where the probability to leave the nest, P_l, is adapted on-line.

Task allocation: the dynamics of the adaptation leads to task allocation among robots and to the creation of two robots classes: *foragers* and *loafers*.

Individual capabilities: the task allocation mechanism exploits natural differences among robots so that those that are mechanically better for retrieval are more likely to become foragers.

This chapter is organised as follows: Section 5.1 describes the adaptation mechanism. Section 5.2 describes the experimental setup and the values of several control parameters. Section 5.3 tests and analyses the efficiency of the system. Section 5.4 deals with the task allocation and Section 5.5 with the individual capabilities.

5.1 *Variable Delta Rule* to Adapt the Probability to Leave the Nest

P_l, the probability to leave the nest, is modified according to Algorithm 1. It keeps trace of the consecutive number of successes and failures of the *MindS-bot*. 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 is, the larger is the reward given by this algorithm, which thereby introduces a form of non-linearity. We call this method "*Variable Delta Rule*" (VDR).

Algorithm 1 Variable Delta Rule. P_l is the probability of leaving the nest.

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

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

The range of P_l is limited to $[P_{min}, P_{max}]$ in order to avoid that a value of 0 or too high values are reached. In fact, if all *MindS-bots* had 0 probability, there would be no foragers any more. If P_l was too high, adding or subtracting Δ to it would have no sensible effects on the mean time spent in the nest, as Figure 5.1 shows.

We decided for this rule out of several alternatives. We first intended to update P_l according to the total number of successes and failures or to the mean time to find a prey. We discarded these alternatives because the information they used keeps track of the history of each *MindS-bot.* We expect that, if the environment changed when the *MindS-bots* had worked for a long time, the adaptation would be very slow since the past would still have a lot of impact. This is only a hypothesis and will be tested in future work.

We investigated an algorithm, which we call "*Fixed Delta Rule*" (FDR) that does not take care of the past for the adaptation of P_l. It adds Δ each time a prey is retrieved and subtract Δ at each failure. Its drawback is related to its speed. The plots in Figure 5.2 show what happens in two example cases when eight prey are placed in the environment and a colony of four robots must retrieve them. When FDR is used (Figure 5.2(a)), some of the *MindS-bots* reach the minimum P_l only after 2500 s. When VDR is used (Figure 5.2(b)), all of them have reached the lower bound after 1700 s.

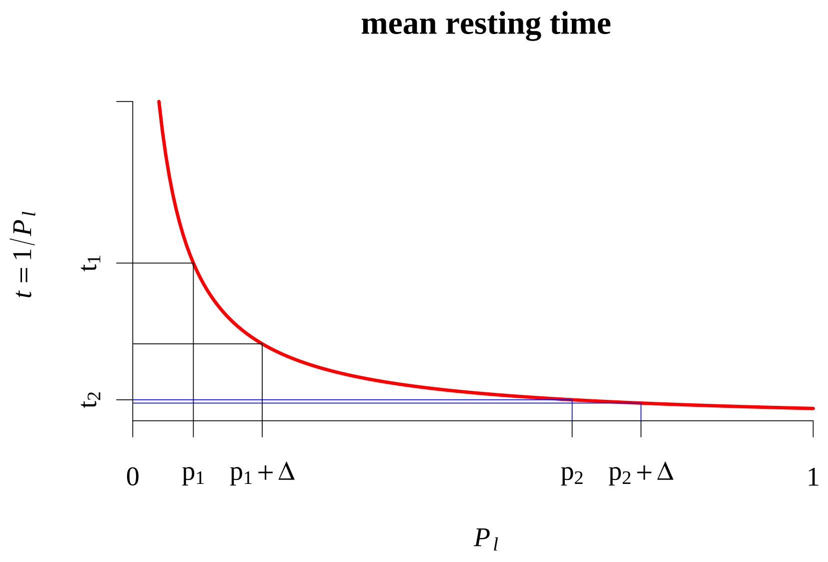


Figure 5.1: Relationship between the probability to leave the nest, P_l, and the mean time t spent in the nest (equal to $1/P_l$). Adding a fixed value Δ to p_2 has no noticeable effect on the mean time spent in the nest t_2. The effect is more sensible with low probabilities, for instance p_1. High probabilities are therefore of no use in our framework and this is why the control algorithm of the *MindS-bots* sets an upper bound to P_l.

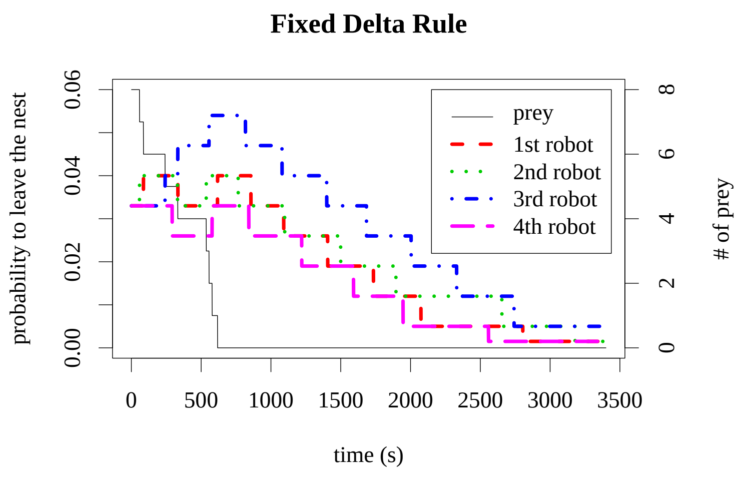
This suggest that VDR allows faster convergence to the equilibrium point of the system. As a consequence, it allows us to carry out our experiments in less time.

5.2 Experimental Setup

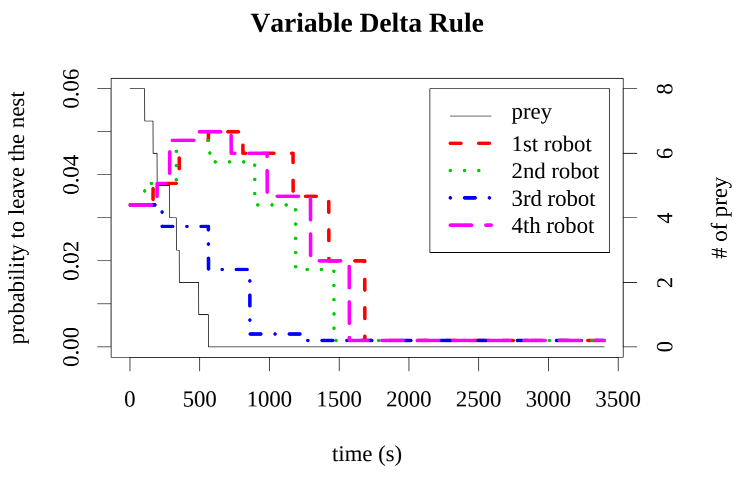
In our experiments, we used a circular arena (Figure 5.3) with a diameter of 2,40 m. A light bulb is placed over the nest area in the centre. Walls and floor are white painted to be more reflective. Prey are made of small bottles covered with black curly cotton. Curls and black decrease the reflectivity of the tissue and improve prey recognition capabilities.

The three thresholds used by the control algorithm (*B_T*, *P_T* and *N_T*, see Table 4.1) are calibrated at the beginning of each experiment on each robot to adapt to the luminosity of the experiment room. The calibration is done placing the *MindS-bots* and a prey in positions that are fixed for all *MindS-bots* and for all the experiments. Each *MindS-bot* takes some samples of sensor readings, whose average is used to set the thresholds.

The search time limit is set to 228 s. This value was decided after having



(a) Fixed Delta Rule to update the probability to leave the nest. *MindS-bots* reach the minimum value nearly two thousand seconds after the last prey is retrieved.



(b) Variable Delta Rule to update the probability to leave the nest. *MindS-bots* reach the minimum value nearly one thousand seconds after the last prey is retrieved.

Figure 5.2: Comparison between the dynamics of the probabilities of leaving the nest in two example cases when a system of four *MindS-bot* 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 Fixed Delta Rule, while the bottom one was obtained with *MindS-bots* that used the Variable Delta Rule (see text for details). The thin black line is the number of prey in the environment and is scaled to fit the plot. The thick dotted lines are the probabilities of leaving the nest for each robot used in the experiment.

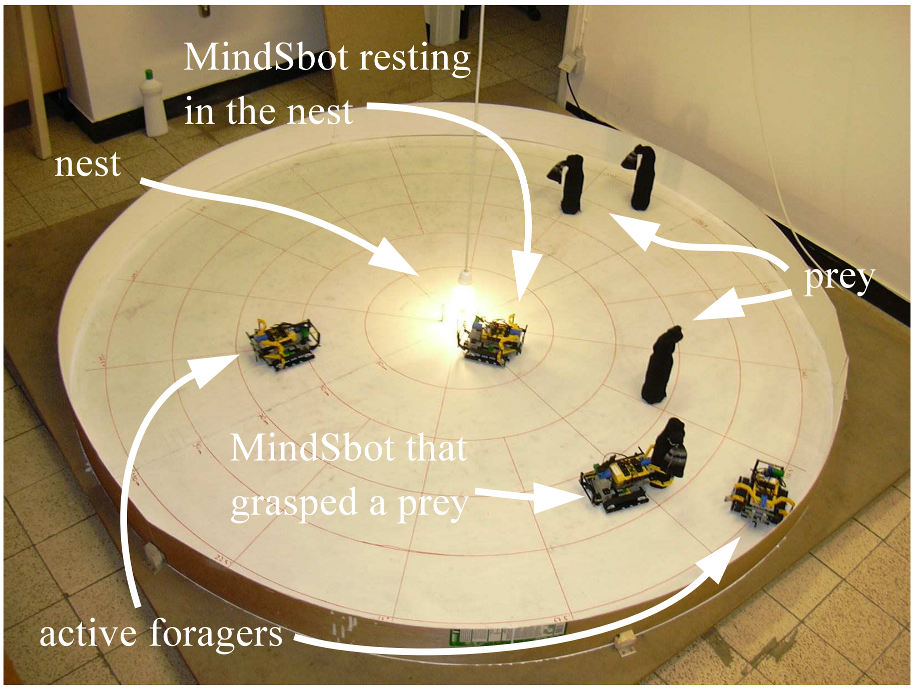


Figure 5.3: Snapshot of an experiment. Four *MindS-bots* are looking for three prey. The nest is indicated by a light in the centre. One *MindS-bot* is resting in the nest and is therefore inactive. Two others are exploring the environment. One found a prey. It grasped it and is searching for the nest to retrieve it.

analysed the mean time a *MindS-bot* took to find a prey. The times to find a prey were recorded in a series of 15 experiments. In each experiment, one prey was placed in a random position uniformly distributed in space, with the only constraint that the distance from the nest must be in the range $[0.5\,m,\ 1.1\,m]$. Figure 5.4 shows the empirical cumulative density function of the observed times. Given a time t on the abscissa, the corresponding value on the ordinate is the relative number of observations that were less than or equal to t. The probability function of a geometrical distribution with expected value equal to the observed average time is superposed upon the empirical data. A Kolmogorov-Smirnoff test (see Siegel and Castellan Jr., 1988, p. 51–55) for the goodness of fit of the two distribution gives a p-value of 0.74. It is therefore reasonable to use the geometrical distribution as a rough approximation of the experimental data. 228 s is its 0.5–quantile, i.e. a single *MindS-bot* has probability 0.5 to find a prey before the timeout is reached.

The upper bound for P_l (see Section 5.1) is set to 0.05, which corresponds to a mean idleness time of 20 s. The lower bound is 0.0015 (mean idleness

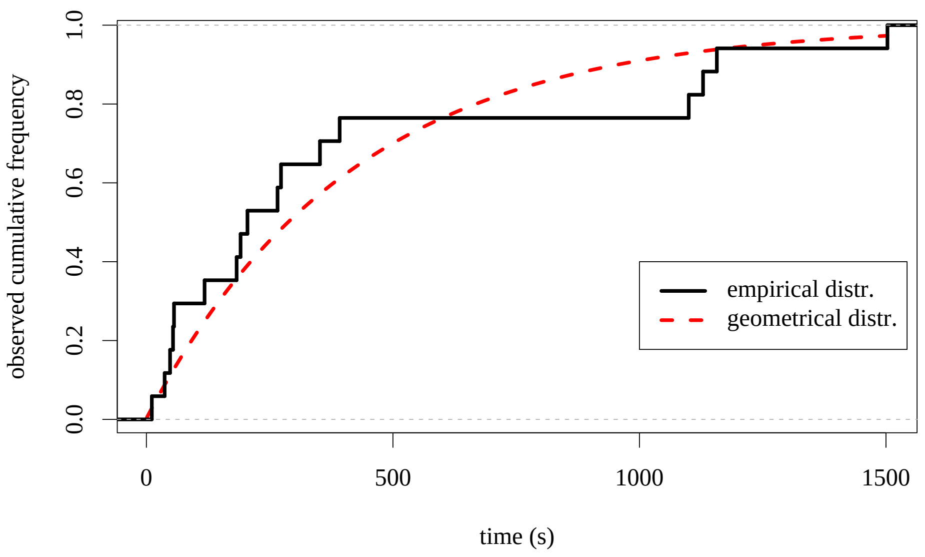


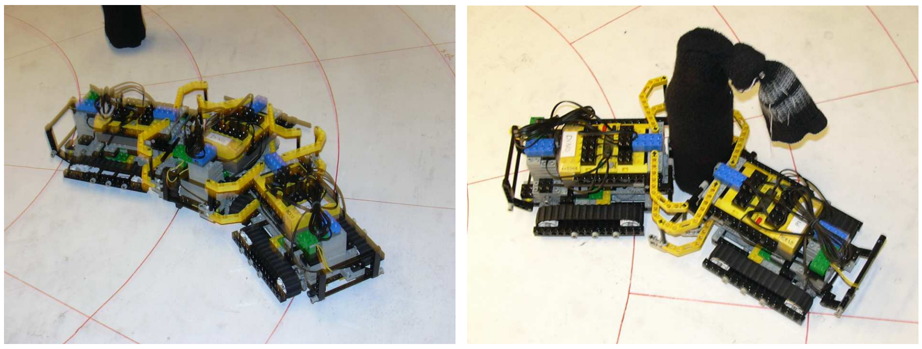
Figure 5.4: Comparison between the empirical cumulative frequency of the time to find one prey and a geometrical distribution. The black continuous line gives the relative number of observations that were less than or equal to the corresponding value on the abscissa. The average time is 416.18 s. The dotted line is a geometrical distribution with the same mean time. A Kolmogorov-Smirnoff test with the null hypothesis that the two curves are equal, returned a p-value of 0.74. Therefore, although the two lines do not fit perfectly, the geometrical distribution can be considered a good approximation of the experimental data.

time: 666.6 s). At the beginning of each experiment P_l is set to 0.033 (mean idleness time: 30 s). The values were tuned with a trial-and-error methodology until satisfactory values were reached. The base increment and decrement of probability, Δ, is set to 0.005.

5.3 Efficiency of the System

The first of our hypotheses is that the adaptation of P_l increases the efficiency of the foraging task. We first need to define a metric for the efficiency in order to evaluate the results we get from experiments. The most straightforward definition of efficiency is:

$$\eta = \frac{income}{costs} \ . \tag{5.1}$$



(a) Two *MindS-bots* mistake a nestmate for a prey and try to grasp it. This occurs because the shadow that the *MindS-bot* casts can be sometimes enough to be misinterpreted by other *MindS-bots*.

(b) Two *MindS-bots* want to grasp the same prey. Both will fail and retry again. Eventually, either one of them goes somewhere else or its search time limit is exceeded, living the prey to the other. It could also happen that both give up.

Figure 5.5: Examples of some conflicts that decrease the efficiency of the system and that are part of the cost function described in Section 1.2. Some of them could be solved making the control system more sophisticated and thus more complex or, as in our case, decreasing the mean number of foragers.

"Income", as defined in Section 1.2, refers to the benefit that the colony has from retrieval. "Costs" include those phenomena that reduce the income of energy. The former can be easily computed because it is proportional to the number of retrieved prey, but the latter is more complex because it depends on unknown factors of the environment. For instance, a *MindS-bot* could get lost or, if it is working in a very hard environment, it could be destroyed. These events can be considered costs because they reduce the resources of the colony, although they are hard to quantify. Other examples of costs are the phenomena depicted in Figure 5.5.

We decided to use a value that is directly related to the efficiency of the system:

$$\nu = \frac{number\ of\ retrieved\ prey}{colony\ duty\ time}\ , \tag{5.2}$$

where "*colony duty time*" is the sum of the time each *MindS-bot* spent to search for a prey. As written above, the number of prey is proportional to the income of the colony. The colony duty time is an indirect measure of the costs. The more time spent by each agent to search, the more energy is consumed, the more the chances to get lost or to break. If the colony duty time increases, also the costs are higher. We can use ν as an estimator for η because colony duty time and costs are dependent, although their relationship could be non linear.

To test our hypotheses, we measured ν as a function of time in a colony that used the Variable Delta Rule and compared it to a colony where adaptation did not occur. We used the method of paired experiments: each experiment done with a colony that used the Variable Delta Rule was replicated with a colony where P_l was constant and equal to 1. In the rest of this section, the experiments where the Variable Delta Rule was used are called "*base experiments*" and the corresponding ones, where P_l was constant, "*control experiments*".

Data was collected from a set of ten experiments. The number of experiments was constrained mainly by the time it took to finish them. The same group of four *MindS-bots*, out of a pool of six, was used both in base and control experiments, and changed among experiments. The decision on which *MindS-bots* to use for each couple of experiments depended purely on the charge level of their batteries and their mechanical and sensor reliability at the moment the decision had to be taken. State transitions were recorded during the experiments according to what is shown in Figure 4.1.

Prey appeared randomly in the arena. The probability of prey appearance each second was 0.006, that is a new prey appeared in mean every 167 s. Its position was selected randomly in the zone between 0.5 m and 1.10 m from the centre of the arena. The reasons for the bounds are twofold: if a prey had been placed further than 1.10 m, it would have been against the wall and a *MindS-bot* would not have been able to grip it and, if it had been placed nearer than 0.5 m, the nest light intensity would have been so strong that a *MindS-bot* would not have been able to recognise the prey. A random number generator, running on a computer next to the arena, decided times and positions of prey appearance for the base experiments. Times and positions were recorded and reused also for the control experiments.

The idea behind a paired test is to see how two different systems behave in the same conditions. *MindS-bots* and prey appearance schemata were kept constant in each base-control experiment couple. Also the control program did not change, except that P_l was fixed to 1 in the control experiments. More specifically, the search timeout mechanism was kept in both cases. In control experiments, when the time limit was reached, *MindS-bots* returned in the nest to exit in the subsequent control cycle. We think that this mechanism can avoid deadlocks as those depicted in Figure 5.5, and therefore it can have a sensible effect on the efficiency of a colony. If we had eliminated it in the control experiments, we would have introduce a bias between the two setups and we would not have been able to assess the role of the adaptation only.

Figure 5.6 plots the mean value of ν in time for both the base and the control experiments. It can be seen that the mean value for the colony that uses the Variable Delta Rule is higher than in the control experiments. The

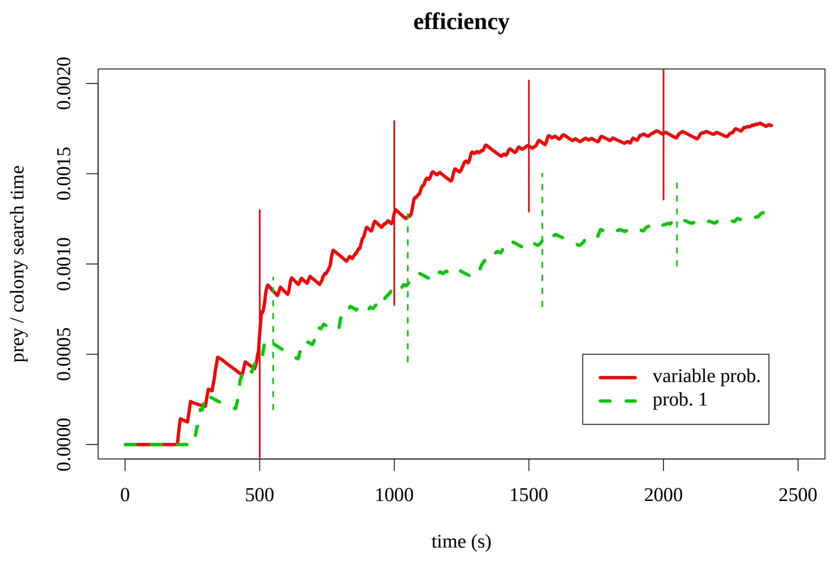


Figure 5.6: Comparison of system efficiency (number of retrieved prey divided by colony duty time) when probability is adapted (continuous line) and when it is not (dotted line). The average over 10 experiments is plotted. Vertical lines represent the standard deviation. The two curves are statistically different with confidence 95% after 1400 s.

difference between the curves is statistically significant after 1400 s.[1] If ν increases, then the efficiency increases. Since the adaptation of P_l is the only thing that changes in the two setups, we can conclude that Variable Delta Rule does improve the efficiency of foraging and retrieval.

We are interested to understand why ν is higher in the base experiments. ν can improve in two ways: an increase in the number of retrieved prey or a decrease of the mean time to search and retrieve a prey. Obviously, a combination of the two phenomena can also occur. The question we want to answer is therefore: is the increase of ν due to more retrieved prey or to shorter retrieving time (or both)?

We can reject the first hypothesis. In the case of the base experiments, the final ν is nearly 1.41 times bigger than in the control experiments ($1.71\ 10^{-3}$ vs. $1.22\ 10^{-3}$). If we look at the number of retrieved prey (Table 5.1), we see that *MindS-bots* retrieved more prey in the base experiments than in the control ones. However, it is only 1.04 times more and

[1]We used the sign test (see Siegel and Castellan Jr., 1988, p. 80–87) with the null hypothesis that the two efficiencies are equal. The alternative is that the efficiency of the system with Variable Delta Rule is better. The p-value is 0.01074 after 1400 s and 0.00098 after 1500 s.

Table 5.1: Number of prey retrieved for each experiment when the probability to leave the nest is either adapted or when it is set to the constant value of 1. The first column contains the experiment number and the second the total amount of prey that appeared during the experiment. The third one is the number of prey retrieved when the probability is modulated with the Variable Delta Rule, the forth one is the same but for the control experiments. The last row sums the results. Bold numbers are used to indicate which setup retrieved more prey.

Exp.	*Tot. prey*	*prey retrieved*	
		with adaptation	without adaptation
1	15	**14**	13
2	17	14	14
3	12	**8**	7
4	18	**12**	11
5	16	11	**12**
6	21	**18**	15
7	14	10	**12**
8	16	12	**15**
9	16	**16**	14
10	24	**19**	15
Total:	169	134	128

there is no statistical difference between the two setups.[2] Therefore, we can conclude that the difference of retrieved prey is not enough to justify the increase of ν, which is due therefore to a decrease of the search and retrieve time.

The results about the number of retrieved prey, which can be considered as an index of the *performance*[3] of the retrieval, were at the beginning a bit surprising. Foraging could be thought as an example of multi objective optimisation. On the one hand, we would like to retrieve as many prey as possible, i.e. to maximise the performance. On the other hand, we would like to maximise the efficiency. We expected that reducing the number of foragers would have improved the efficiency but would have decreased the performance. Our experiments showed that this was not true. This can be

[2]We performed a permutation test on the data in Table 5.1. This kind of test is among the most powerful because it uses all the information available in a data set (Siegel and Castellan Jr., 1988, p. 95). The null hypothesis was that the number of retrieved prey is equal in the two setups and it was tested against the hypothesis that the base experiments performed better. The p-value we obtained was 0.2637, which is the double in the case the alternative hypothesis is that the number of retrieved prey is different in the two setups.

[3]The meaning we give to the word "*performance*" follows the one used, among the others, by Balch (1999) and Schneider-Fontán and Mataric (1996).

explained by the fact that the environment was "tough" for the *MindS-bots*, probably because of the density of robots was too high or the density of prey was too low and three or four *MindS-bots* could have overcrowded our arena.

We think of three possible scenarios in a foraging task when changing the robot density:

- Low robot density reduces the chances of negative interactions among *MindS-bots*. Therefore, both the efficiency and the performance increase with the number of foragers, since each robot successfully retrieves prey without conflicts with the others.
- Higher robot density increases the effect of negative interactions. The efficiency increases by reducing the number of foragers, but the performance decreases.
- In an overcrowded environment, as it was presumably our case, the conflicts among robots are overwhelming. Reducing the number of foragers increases both the efficiency and the performance.

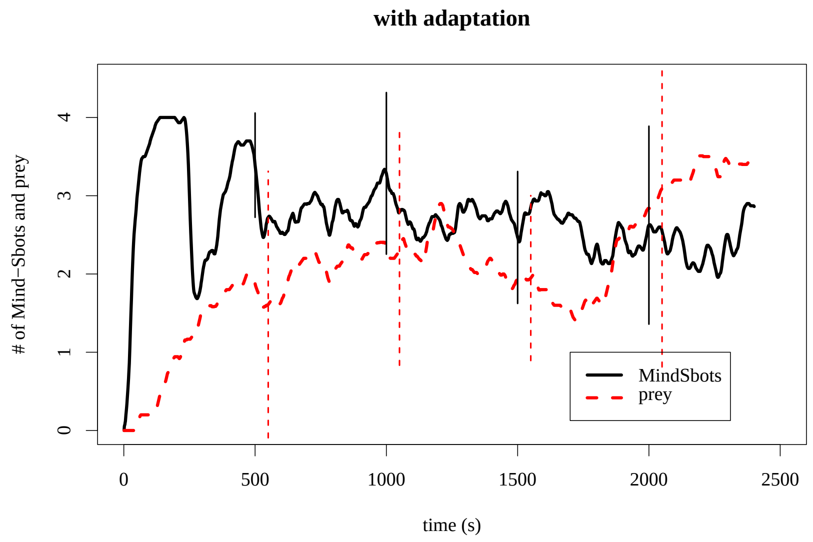
Further analyses and tests on this topic are left as future work.

5.4 Task Allocation

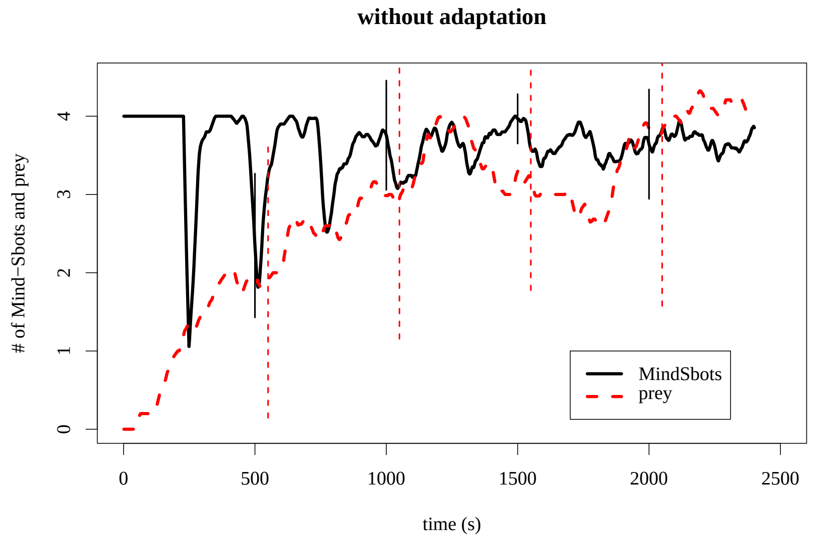
The second of our hypotheses regards task allocation. We expect that the adaptation mechanism leads to the creation of two classes of *MindS-bots* in the colony: *foragers* and *loafers*. The task of the first group is to search and retrieve prey while the second group stays in the nest and avoids to interfere with the activity of the others.

We need a measurable characteristic that differentiates the two groups in order to test this hypothesis. The only means by which the Variable Delta Rule can create task allocation is the modification of P_l. Therefore, P_l is the observable at which we can look for our analysis. If we consider a colony of *MindS-bots* during a foraging experiment, we can follow their P_l during time. We can then consider an instant t when the colony has reached its working regime. The value of P_l of each *MindS-bot* at time t is generally a random variable that assumes different values for different experiments according to an unknown distribution. The distribution of P_l can give us enough information to test our hypothesis. Moreover, it can give us deeper understanding about the mechanism that improves the efficiency of foraging.

Figure 5.7 plots the mean number of prey and searching *MindS-bots* observed both in the base and control experiments of Section 5.3. In the control experiments, *MindS-bots* that were not searching, were going to the nest. When the Variable Delta Rule is used, there were 2.57 foragers and 2.44 prey on average in the arena in the period between 1000 s and 2400 s.



(a) Mean number of *MindS-bots* and prey when the Variable Delta Rule (Section 5.1) is used. After 1000 s, there are on average 2.57 foragers and 2.44 prey.



(b) Mean number of *MindS-bots* and prey in control experiments with probability to leave the nest equal to 1. After 1000 s, there are on average 3.63 foragers and 3.49 prey.

Figure 5.7: Mean number of searching *MindS-bots* and prey observed during the first 2400 seconds in experiments with probability adaptation and in control experiments with P_l equal to 1. The continuous line represents *MindS-bots*, whilst the dotted one represents prey. Vertical lines show the standard deviation. Data is collected over 10 experiments.

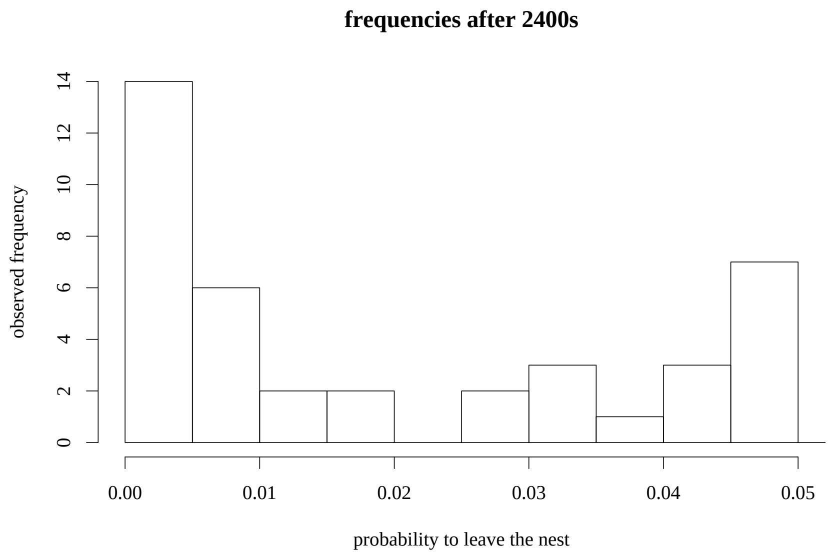


Figure 5.8: Histogram of the observed probability to leave the nest 2400 s after the beginning of experiments. On the abscissa there are intervals of probability to leave the nest, P_l. The height of boxes indicates how many times the value of P_l of each *MindS-bot* is observed to be in the corresponding range. The distribution has two peaks, suggesting that only some of the *MindS-bots* become foragers (high probability) and the others tend to rest in the nest (low probability). The boundary between the two groups is around 0.025. 60% of the observations are below this threshold. Data refers to ten experiments, four *MindS-bots* per experiment.

In the control experiments, there were 3.63 foragers and 3.49 prey. In both cases the ratio is nearly one robot per prey but there are less robot searching prey when adaptation is used. It could be that the adaptation tuned the mean number of foragers by changing the P_l equally in each *MindS-bot* till the optimal mean number of foragers is reached or, as we hypothesise, by decreasing P_l in some robots and increasing it in the others. If the first phenomenon happened, then the distribution of P_l would spread around a single peak. If the second happened, then the distribution would have at least two peaks, one that indicates the nominal P_l of *foragers* and the other of *loafers*. The further the peaks and the steeper the valley between them are, the stronger is the distinction between the classes.

The data of the experiments of Section 5.3 can be used to estimate the distribution of P_l and thus to asses the occurrence of task allocation. We collected the value of P_l for each *MindS-bot* (four *MindS-bot* for ten experiment) after 2400 s. Each *MindS-bot* in each experiment represent an observation from which we can build the empirical distribution that is

Table 5.2: Number of *MindS-bots* per experiment with probability to leave the nest, P_l, less and greater than 0.025. Only in one case all *MindS-bots* have P_l lower than 0.025 (in bold). If combined with the histogram in Figure 5.8, this table suggests that the allocation of tasks among *MindS-bots* is a systematic phenomenon. Data refers to ten experiment, four *MindS-bots* per experiment.

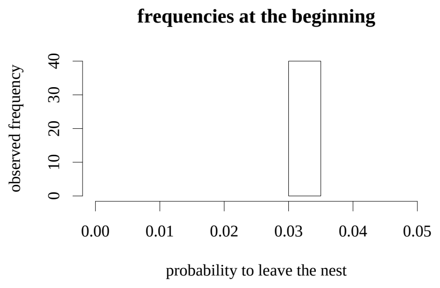
Exp.	$\#_{P<0.025}$	$\#_{P>0.025}$
1	3	1
2	3	1
3	**4**	**0**
4	1	3
5	3	1
6	2	2
7	3	1
8	2	2
9	1	3
10	2	2

shown in Figure 5.8. Its U-shape suggests, as we explained above, that task allocation occurs. There are two groups of *MindS-bot*: those with P_l higher than 0.025, i.e. the foragers, and those with P_l lower than 0.025, i.e. the loafers. There was no observation with P_l near 0.0025.

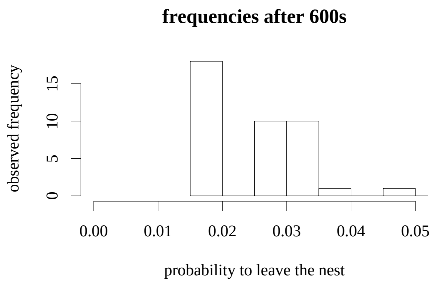
A possible criticism could come from the fact that there are few experiments done and therefore it could be that the right peak of Figure 5.8 is given by a few lucky cases. We can reject this argument by looking at the number of foragers and loafers in each experiment. The numbers are summarised in Table 5.2 and they show foragers were present in nearly all experiment[4] and thus their presence is a systematic phenomenon.

Finally, Figure 5.9 shows the dynamics of the distribution of P_l. At the beginning, all *MindS-bots* starts with the same P_l, and therefore the distribution is a central peak. Slowly, some *MindS-bots* begin to decrease or increase their P_l. Fewer and fewer *MindS-bots* can keep their P_l in the centre of the distribution. At the end, all of them have either a low or high P_l. The two extremes of the distribution (0.0015 and 0.05) seems to be the attractors for P_l.

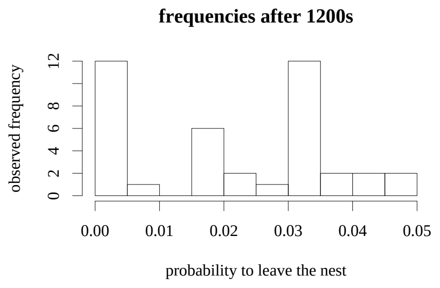
[4]It is worth observing that no forager was present in experiment 3, which is also the one where less prey appeared, as shown in Table 5.1



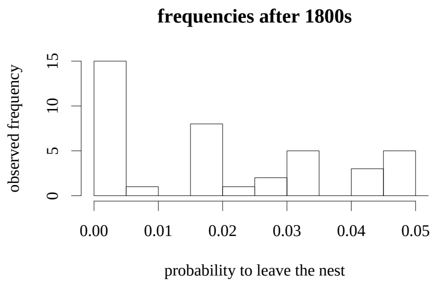
(a) Frequency distribution at the beginning. All *MindS-bots* have the same probability.



(b) Frequency distribution after 600 s. A tail of the distribution drifts toward lower values.



(c) Frequency distribution after 1200 s. Nearly one fourth of the *MindS-bots* have the minimum probability. One other fourth has still high values in the centre.



(d) Frequency distribution after 1800 s. The distribution has no more peaks in the middle. The *MindS-bots* with probability in the centre of the distribution will move right or left to reach the distribution of Figure 5.8

Figure 5.9: Time evolution of observed frequency of the probability to leave the nest, starting from the beginning of experiments (where all *MindS-bots* have the same initial probability) till 1800 s. Data refers to ten experiments, four *MindS-bots* per experiment.

Table 5.3: Number of experiments for each *MindS-bot* in which it had P_l less or greater than 0.025 after 2400 s. The first column reports the identification name of the *MindS-bot*. In the second, there are the total number of experiments each *MindS-bot* attended. The third reports the number of experiments in which its P_l was less than 0.025 and the fourth when P_l was greater than 0.025. A statistical test (see text) shows that the probability to become foragers is not the same for each *MindS-bot*. Therefore, task allocation exploits individual capabilities. Data refers to ten experiment, four *MindS-bots* per experiment.

ID	Tot. Exp.	$\#_{P<0.025}$	$\#_{P>0.025}$
Mind1	6	1	5
Mind2	3	1	2
Mind3	9	8	1
Mind4	9	5	4
Mind5	3	3	0
Mind6	10	6	4

5.5 Role of Individual Capabilities

The third and last hypothesis concerns the role of individual capabilities in the allocation of tasks. We expect that individuals that are better retriever than the others can benefit from the Variable Delta Rule and become foragers more likely.

If our hypothesis is true, then the probability that each *MindS-bot* has to become forager in an experiment, indicated as P^f, is not constant for all the *MindS-bot* or depends on the nest-mates. It could happen that one *MindS-bot* is the best one and therefore it is always going to be forager, that is, his P^f is the highest among the *MindS-bots*. Or, it could be that it becomes a forager only if specific nest-mates are present, that is, its P^f depends on the others. Therefore, the value of P^f can be used to test our hypothesis.

We looked at the number of times in which each *MindS-bot* was a forager after 2400 s in the ten experiments of Section 5.3. We recall that four robots out of six were used in each experiments and that they changed across experiments. Table 5.3 summarises the data: for each robot, identified by an identification name, the amount of experiments in which it was a forager or a loafer is reported. The total number of experiment is not equal for all the *MindS-bot*. This is due to the fact that the main criteria used to select which *MindS-bot* had to participate in an experiment were the mechanical reliability and the battery charge. Some *MindS-bot* resulted to be less reliable than the others and, therefore, they were used less.

It is possible to use a χ^2 test to determine if P^f is not constant for all the *MindS-bots*. An important hypothesis that has to be satisfied in order to apply this test is that the data in each row must be independent (Siegel and Castellan Jr., 1988, p. 191–200). In our case, this means that the individual P^f shall not depend on the other *MindS-bots*, but we do not know if this holds true. Therefore, there are two possibilities:

- The values of P^f are independent: the χ^2 test reject the hypothesis that P^f is the same for all the *MindS-bot* with a significance level of 95,4%. Therefore, individual differences play a role in the task allocation process.

- The values of P^f are not independent: it is not possible to use the χ^2 test. However, as mentioned above, dependent P^f show that differences among *MindS-bots* are relevant and therefore we can still conclude that exploitation of individual differences takes place.

Our hypothesis is proved in both cases.

We do not have enough data to describe the process in more detail. We cannot say, for instance, whether the individual P^f is constant in each experiment or whether its value depends on the nest-mates. But we can conclude that the difference among *MindS-bots* play a crucial role in the task allocation mechanism. This is considered an interesting research path, especially in the context of the SWARM-BOTS project (see Section 1.1.2).

Chapter 6

Conclusions and Future Work

This work is based on a rigorous methodological approach: we defined some hypotheses that we wanted to prove (Section 1.2), we created our research tools, the *MindS-bots* (Chapter 3 and 4), and finally we designed a set of experiments for validation (Chapter 5). The results show that the starting hypotheses are true: the efficiency of the system improves using a swarm-intelligent technique, that is, the adaptation of the probability to leave the nest, and by means of task allocation, which exploits physical differences among robots.

Adaptation works independently on each robot and it is based only on locally available information. No form of direct or symbolic communication is used, and therefore the control system of each robot is simple. We do not claim that the adaptation rule we used (Section 5.1) achieves optimal results, but we show that it is theoretically feasible to reach global efficiency optimisation by exploitation of local behaviours and without direct communication. Since no direct communication or global information is used, this process can be considered *self-organised*.

The preliminary results shown in Chapter 5 raise other issues that are worth of analysis. However, the main problem we have to face before continuing our research regards the time it takes to run experiments. Each run lasted approximately forty minutes, and it took a lot of time to reach any statistical significance, slowing down our research. We plan to move to simulation using the criteria discussed in Jakobi et al. (1995) to fill the gap between reality and simulation. To achieve this, we need to identify a number of features essential to describe a real situation, like the probability to lose a prey or the probability to get stuck in a nest-mate, and implement them in the simulator. Further experiments will be run in simulation, but cross validation will occasionally occur using real robots.

So far, experiments use a distribution of prey that is constant in time and space. We think that the Variable Delta Rule (Section 5.1) can be an efficient way for the colony to adapt to sudden changes. Therefore, adaptability of the system is another direction of work that will be addressed in future work. In this context, it is interesting to compare the performance of our system, based on probability, with the one by Agassounon and Martinoli (2002) discussed in Section 2.2, based on time thresholds. We find several differences between our work and theirs:

- In their setup the density of items in the environment is fixed. In our work prey appear probabilistically.
- We use probabilities instead of time thresholds. This could be more advantageous if the system had to adapt to a dynamic environment. If behaviours were probabilistic, some robots would take the right action in a changed environment before the others. The frequency of right actions would be amplified over time by means of learning. After a while, some members of the colony would be adapted to the new conditions. If behaviours were based on time thresholds, the dynamics of adaptation would be fixed and might be too slow if the thresholds were not well tuned.
- We use different information for our updating rule. The mean time used by Agassounon and Martinoli keeps track of the whole history of the system, while we are more interested in remembering only the recent past and to forget the rest in order to have a faster adaptation.

We plan to run a series of experiments to study the differences in adaptation speed between the two setups.

Another topic that is worth of analysis is how performance, efficiency and task allocation depend on the initial density of robots. Concerning task allocation, we expect that there is a critical value for the density of robots after which task allocation does not occur anymore. In fact, in an overcrowded environment, each robot has a small probability of finding a prey, and therefore it tends to become a loafer. This is different from other biological phenomena in which the higher the density, the more specialised the individuals are, e.g. human cells that become organised in tissues with different tasks.

We see also some limitations in our system. Some of the control parameters (timeout, minimum and maximum probabilities) are tuned to work well in our arena. We want to study their effects on efficiency and task allocation and, eventually, to relax these constraints in order to have more flexibility.

Finally, collective retrieval is a fundamental task to analyse in the scope of the SWARM-BOTS project and raise several interesting issues. We are interested in two aspects: the self-assembling process needed to solve the task and how this form of cooperation can arise without direct communication.

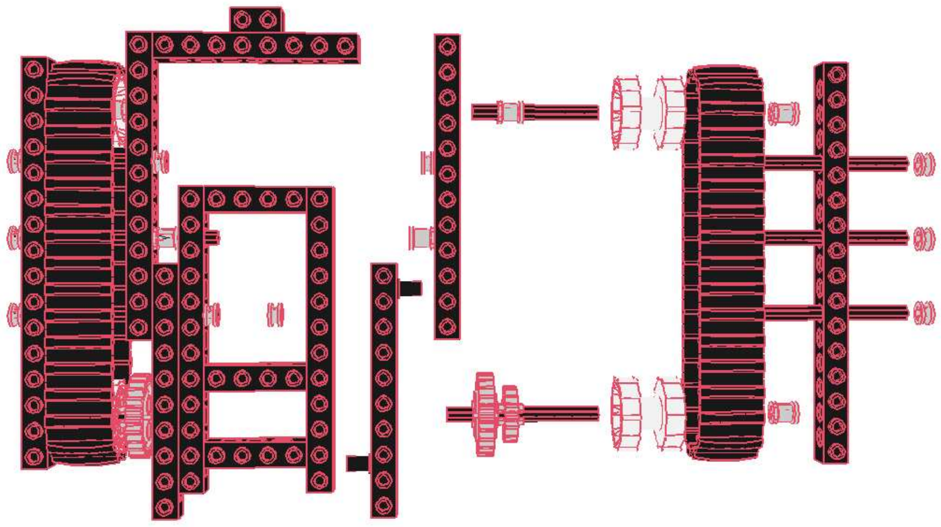
Appendix A

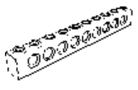
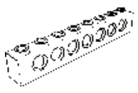
List of Pieces and Assemblage Instructions

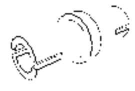
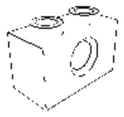
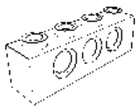
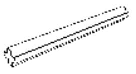
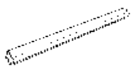
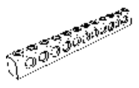
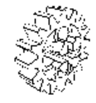
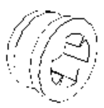
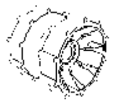
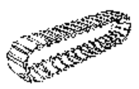
The following pages show how to build the *MindS-bots* used in our work. The instructions are divided in sixteen steps. Each step shows a picture of the partially built *MindS-bot* and a table with the list of the needed pieces for the step. Red edges highlight the new pieces in each picture. For each new piece, its picture, its description, the quantity and the colours are reported in the table. The identification number of each piece is between brackets in the description. Nearly all the pieces come from a single Lego Mindstorm^{TM} box. Only one additional motor (piece number 71727C01) and one extra light sensor (number 2982C01) have to be taken from another source, e.g. from another box or a Lego retailer.[1]

[1]The pictures and the instructions in this appendix were generated with the help of *LeoCAD* (`http://www.leocad.org/`).

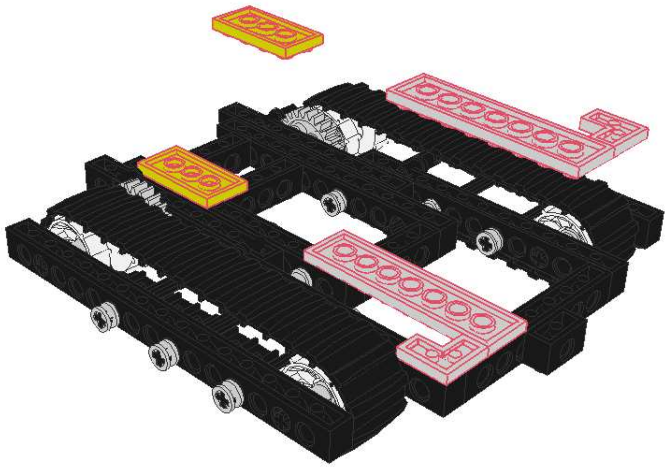
Step 1

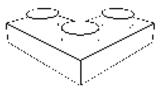
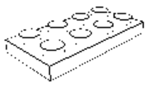
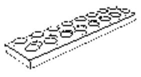


	Descr	White	Black	Silver
	Technic Brick 1 x 10 with Holes (2730)	-	2	-
	Technic Axle 5 (32073)	-	2	-
	Technic Gear 24 Tooth (3648)	-	-	2
	Technic Brick 1 x 8 with Holes (3702)	-	1	-
	Technic Brick 1 x 16 with Holes (3703)	-	2	-
	Technic Bush (3713)	-	-	8

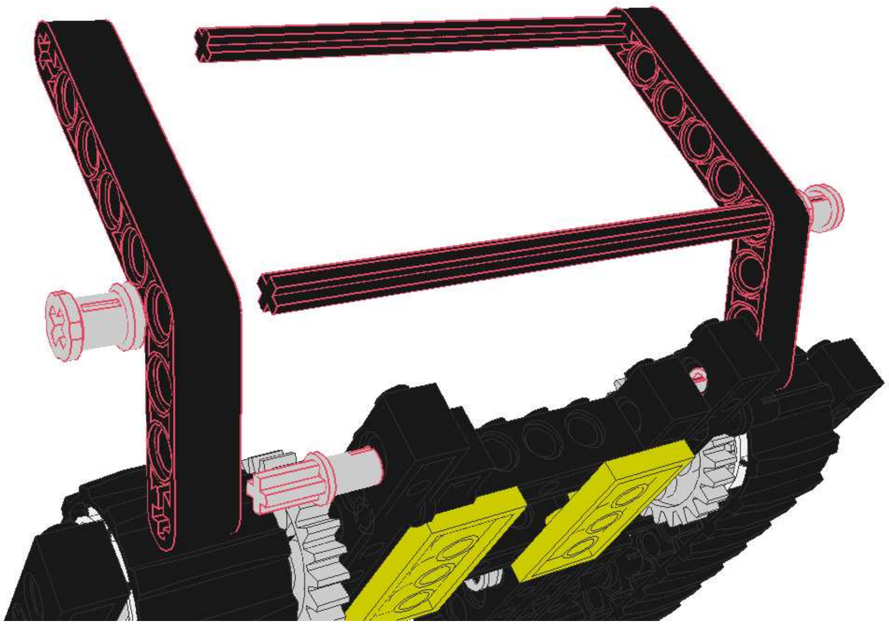
	Descr	**White**	**Black**	**Silver**
	Technic Pin (3673)	-	4	-
	Technic Brick 1 x 2 with Hole (3700)	-	1	-
	Technic Brick 1 x 4 with Holes (3701)	-	3	-
	Technic Axle 6 (3706)	-	4	-
	Technic Axle 8 (3707)	-	4	-
	Technic Brick 1 x 12 with Holes (3895)	-	4	-
	Technic Gear 16 Tooth (4019)	-	-	2
	Technic Bush 1/2 Smooth (4265C)	-	-	10
	Technic Tread Sprocket Wheel (32007)	4	-	-
	Technic Tread (680C01)	-	2	-

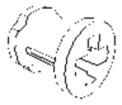
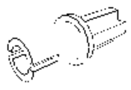
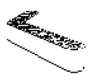
Step 2



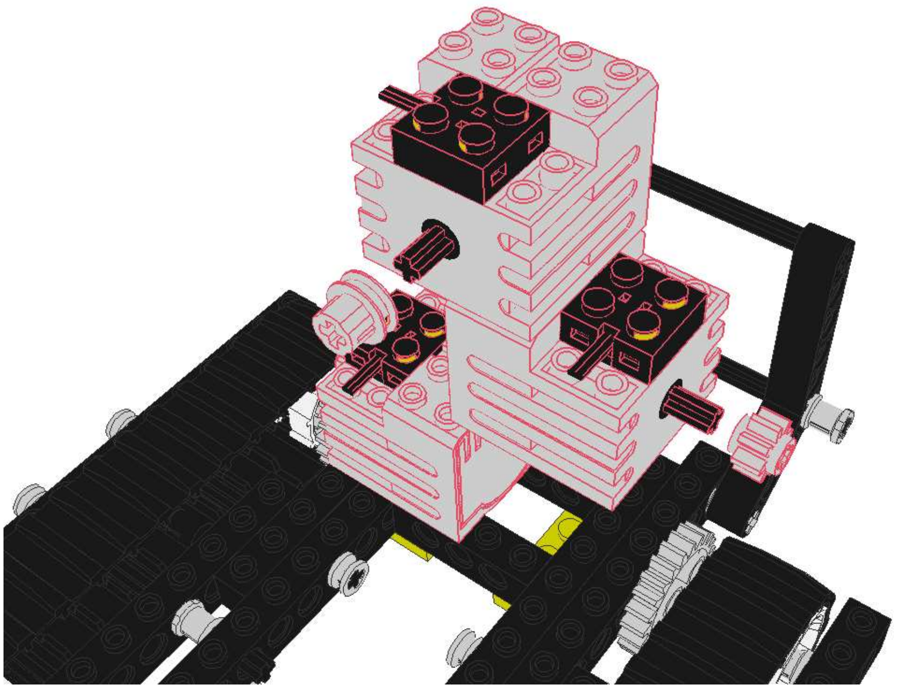
	Descr	**Yellow**	**Silver**
	Plate 2 x 2 Corner (2420)	-	2
	Plate 2 x 4 (3020)	2	-
	Technic Plate 2 x 8 with Holes (3738)	-	2

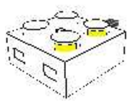
Step 3



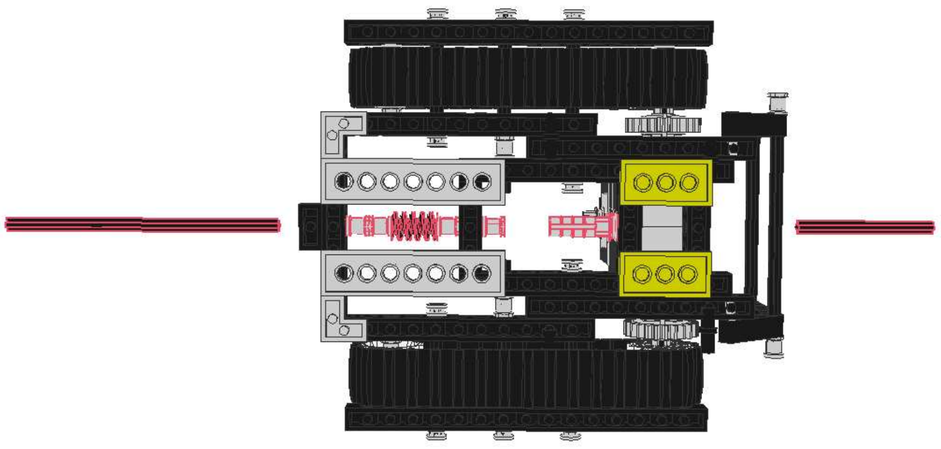
	Descr	Black	Silver
	Technic Bush (3713)	-	2
	Technic Axle 12 (3708)	1	-
	Technic Axle 10 (3737)	1	-
	Technic Axle Pin (3749)	-	2
	Technic Liftarm 1 x 9 Bent (6629)	2	-

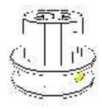
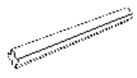
Step 4

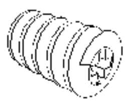
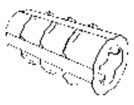


	Descr	Black	Silver
	Technic Micromotor Pulley (2983)	-	1
	Technic Gear 8 Tooth (3647)	-	2
	Brick 2 x 2 x 2/3 Electric Wire End (5306)	3	-
	Technic Mini-Motor 9v (71427C01)	-	3

Step 5

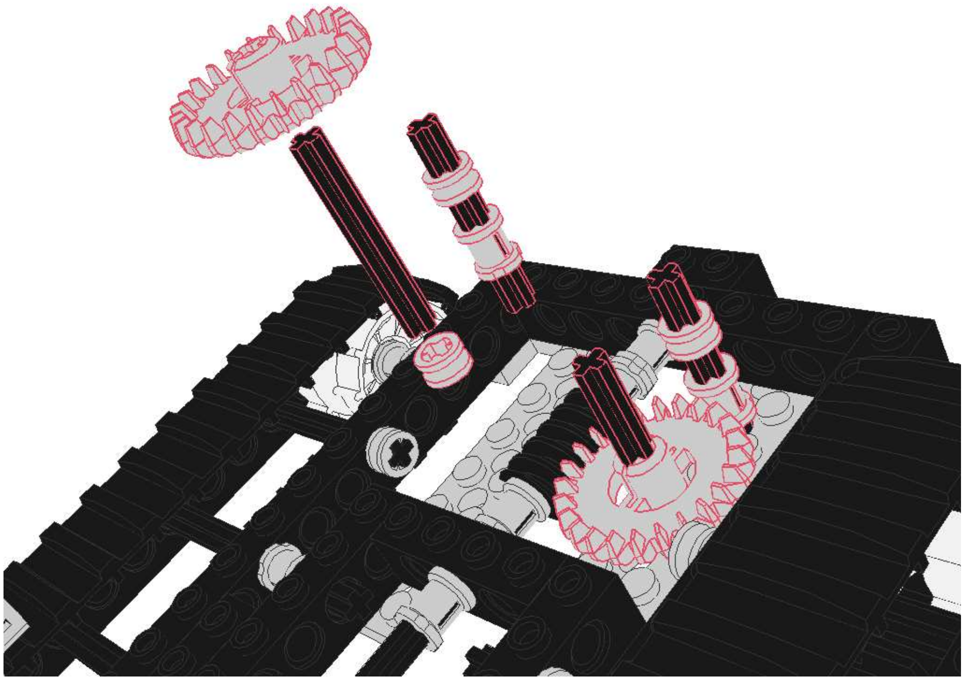


	Descr	**Black**	**Silver**
	Technic Micromotor Pulley (2983)	-	1
	Technic Bush (3713)	-	4
	Technic Axle 6 (3706)	1	-
	Technic Axle 12 (3708)	1	-

	Descr	Black	Silver
	Technic Worm Screw (4716)	1	-
	Technic Axle Joiner Offset (6538B)	-	1

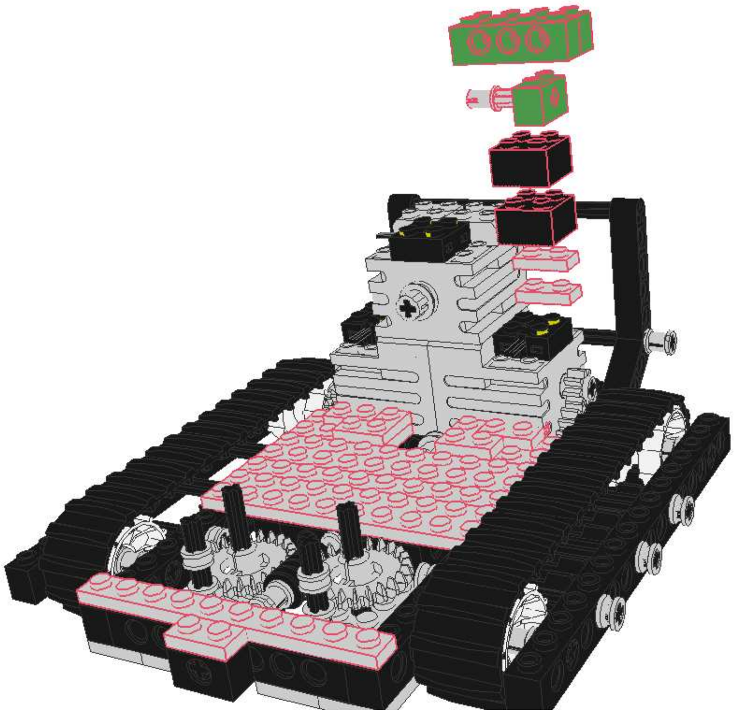
A yellow elastic, as those that can be found in a normal Lego Mindstorm^{TM} box (not shown in this and the following pictures), connects the pulley used here and the one of Step 4.

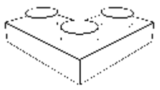
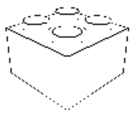
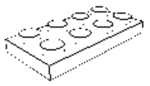
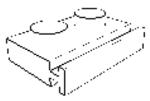
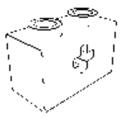
Step 6

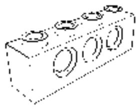
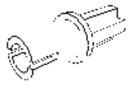
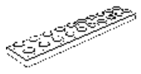
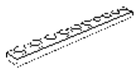


	Descr	Black	Silver
	Technic Gear 24 Tooth Crown (3650A)	-	2
	Technic Bush (3713)	-	2
	Technic Axle 4 (3705)	4	-
	Technic Bush 1/2 Smooth (4265C)	-	4

Step 7

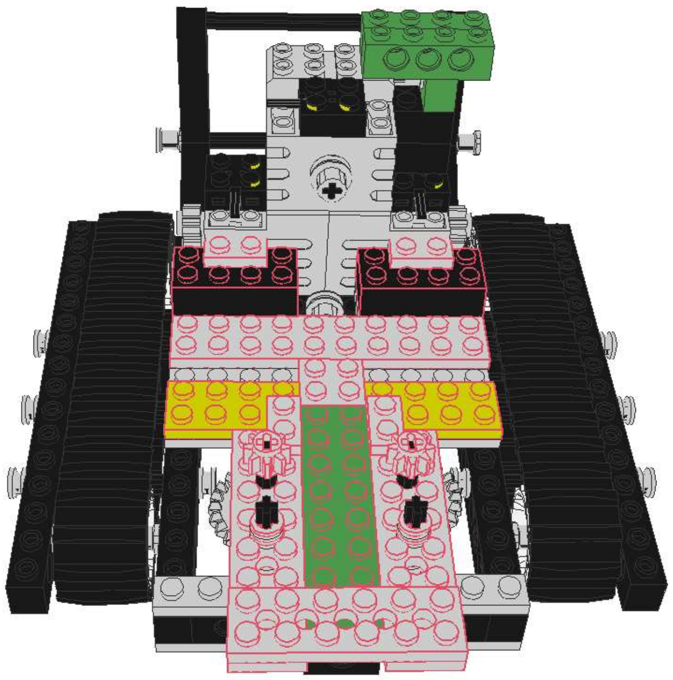


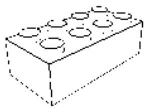
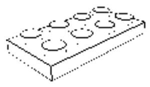
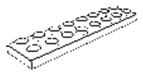
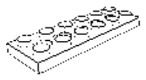
	Descr	**Light Green**	**Black**	**Silver**	0
	Plate 2 x 2 Corner (2420)	-	-	2	
	Brick 2 x 2 (3003)	-	2	-	
	Plate 2 x 4 (3020)	-	-	2	
	Plate 1 x 2 (3023)	-	-	3	
	Plate 1 x 2 with Door Rail (32028)	-	-	2	
	Technic Brick 1 x 2 with Axle-hole (32064)	1	-	-	

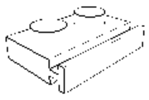
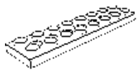
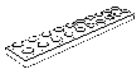
	Descr	Light Green	Black	Silver
	Technic Brick 1 x 4 with Holes (3701)	2	-	-
	Technic Axle Pin (3749)	-	-	1
	Plate 2 x 10 (3832)	-	-	2
	Plate 1 x 10 (4477)	-	-	2

A yellow elastic, as those that can be found in a normal Lego Mindstorm^{TM} box (not shown in this and the following pictures), connects the pin and the axle on the top of the back bumper that was built in Step 3

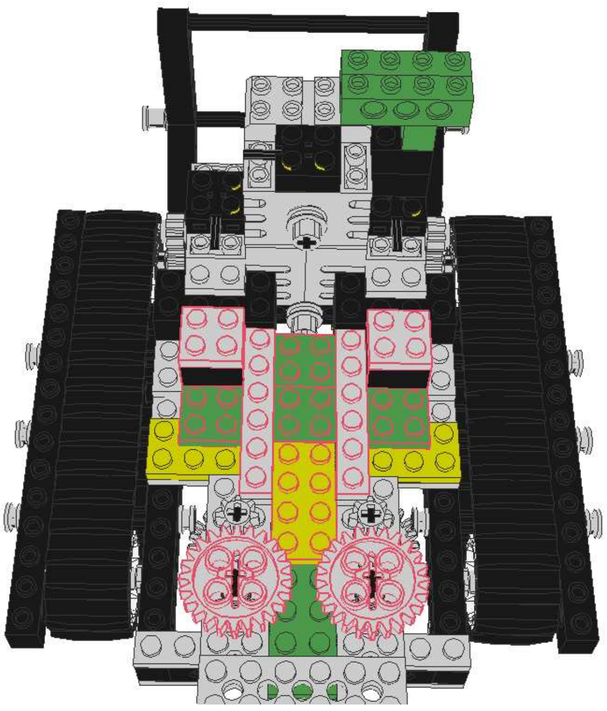
Step 8

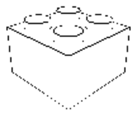
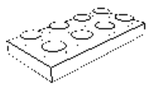
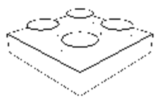
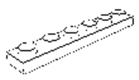


	Descr	Light Green	Yellow	Black	Silver
	Brick 2 x 4 (3001)	-	-	2	-
	Plate 2 x 4 (3020)	-	2	-	-
	Plate 2 x 2 (3022)	-	-	-	1
	Plate 1 x 2 (3023)	-	-	-	2
	Plate 2 x 8 (3034)	1	-	-	-
	Plate 2 x 6 with Holes (32001)	-	-	-	1

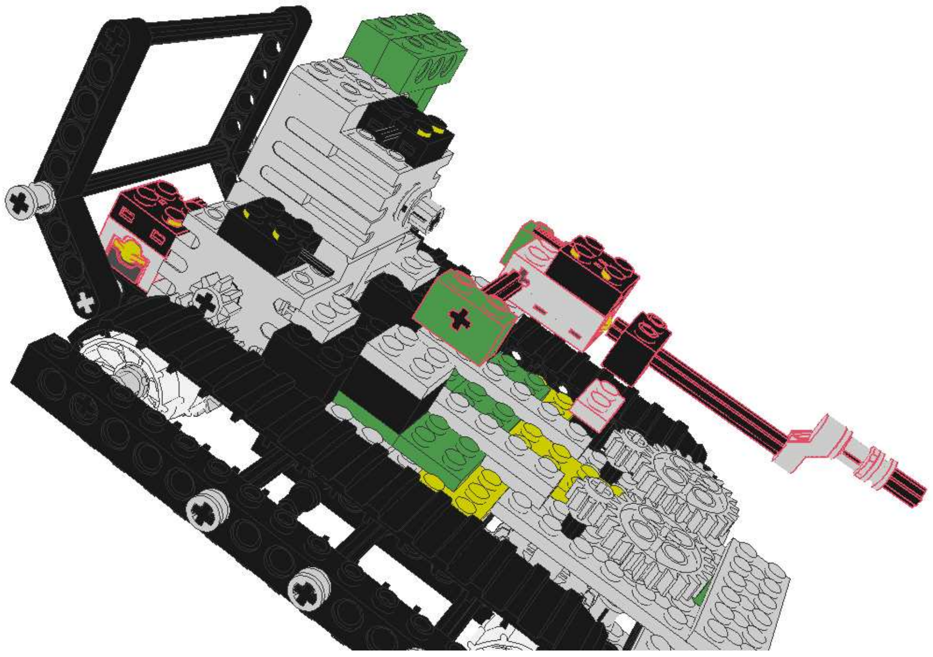
	Descr	**Light Green**	**Yellow**	**Black**	**Silver**
	Plate 1 x 2 with Door Rail (32028)	-	-	-	2
	Plate Technic Gear 8 Tooth (3647)	-	-	-	2
	Technic Plate 2 x 8 with Holes (3738)	-	-	-	2
	Plate 2 x 10 (3832)	-	-	-	1
	Technic Bush 1/2 Smooth (4265C)	-	-	-	2

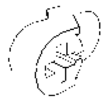
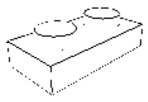
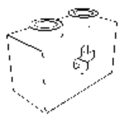
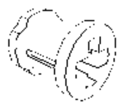
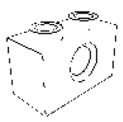
Step 9

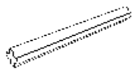
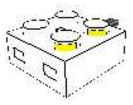
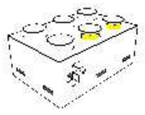


	Descr	Light Green	Yellow	Black	Silver
	Brick 2 x 2 (3003)	-	-	2	-
	Plate 2 x 4 (3020)	-	1	-	-
	Plate 2 x 2 (3022)	-	-	-	2
	Plate 2 x 8 (3034)	2	-	-	-
	Technic Gear 24 Tooth (3648)	-	-	-	2
	Plate 1 x 6 (3666)	-	-	-	2

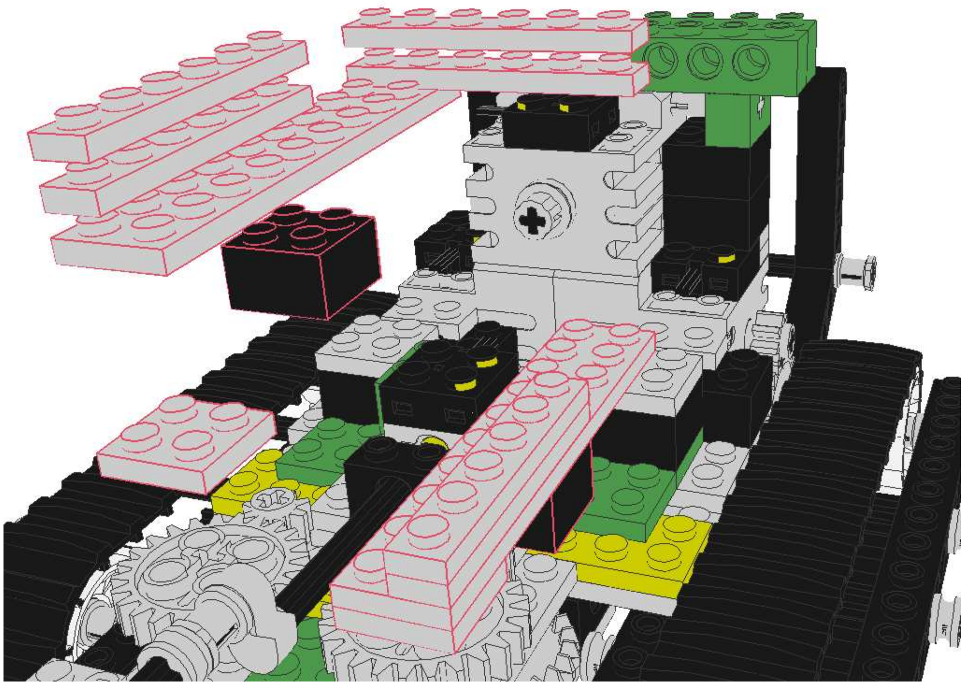
Step 10

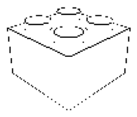
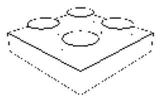
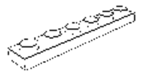
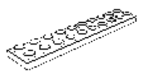


	Descr	Light Green	Black	Silver
	Technic Engine Crankshaft Centre (2854)	-	-	1
	Plate 1 x 2 (3023)	-	-	1
	Technic Brick 1 x 2 with Axle-hole (32064)	2	-	-
	Technic Bush (3713)	-	-	1
	Technic Brick 1 x 2 with Hole(3700)	-	1	-

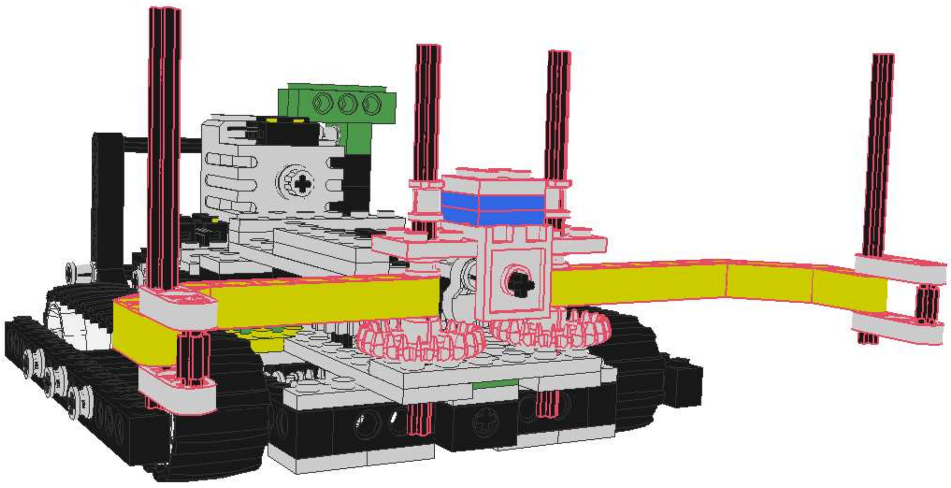
	Descr	**Light Green**	**Black**	**Silver**
	Technic Axle 4 (3705)	-	1	-
	Technic Axle 6 (3706)	-	1	-
	Technic Bush 1/2 Smooth (4265C)	-	-	1
	Technic Axle 3 (4519)	-	1	-
	Brick 2 x 2 x 2/3 Electric Wire End (5306)	-	2	-
	Electric Touch Sensor Brick 3 x 2 (879)	-	-	2

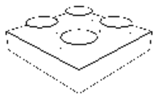
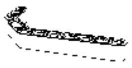
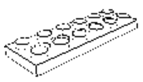
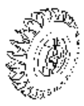
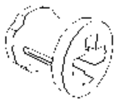
Step 11

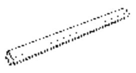
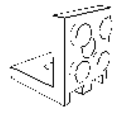
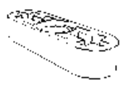
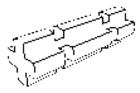


	Descr	Black	Silver
	Brick 2 x 2 (3003)	2	-
	Plate 2 x 2 (3022)	-	2
	Plate 1 x 6 (3666)	-	6
	Plate 2 x 10 (3832)	-	2

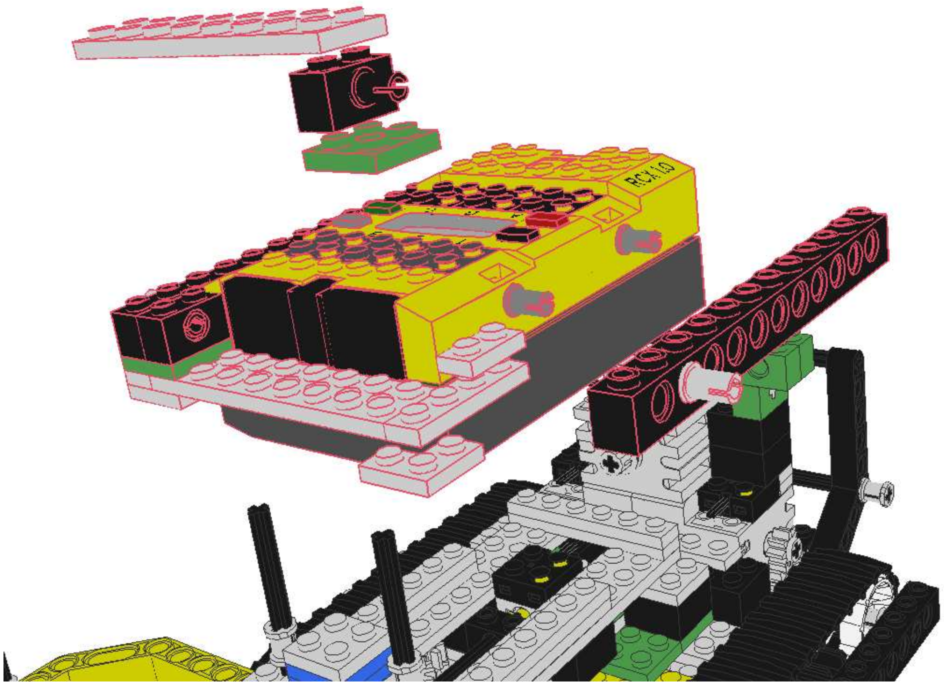
Step 12

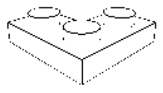
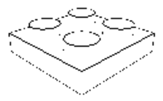
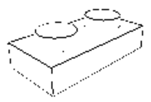
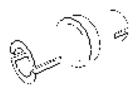
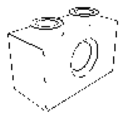


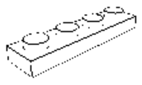
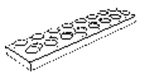
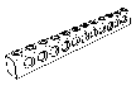
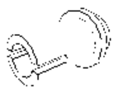
	Descr	Light Blue	Yellow	Black	Silver
	Plate 2 x 2 (3022)	2	-	-	1
	Technic Liftarm 1 x 11.5 Double Bent (32009)	-	2	-	-
	Technic Plate 2 x 6 with Holes (32001)	-	-	-	1
	Technic Gear 24 Tooth Crown (3650A)	-	-	-	2
	Technic Bush (3713)	-	-	-	2

	Descr	**Light Blue**	**Yellow**	**Black**	**Silver**
	Technic Axle 8 (3707)	-	-	2	-
	Technic Axle 10 (3737)	-	-	2	-
	Bracket 2 x 2 - 2 x 2 (3956)	-	-	-	1
	Technic Bush 1/2 Smooth (4265C)	-	-	-	2
	Technic Liftarm 1 x 3 (6632)	-	-	-	4
	Technic Axle 2 Notched (32062)	-	-	2	-

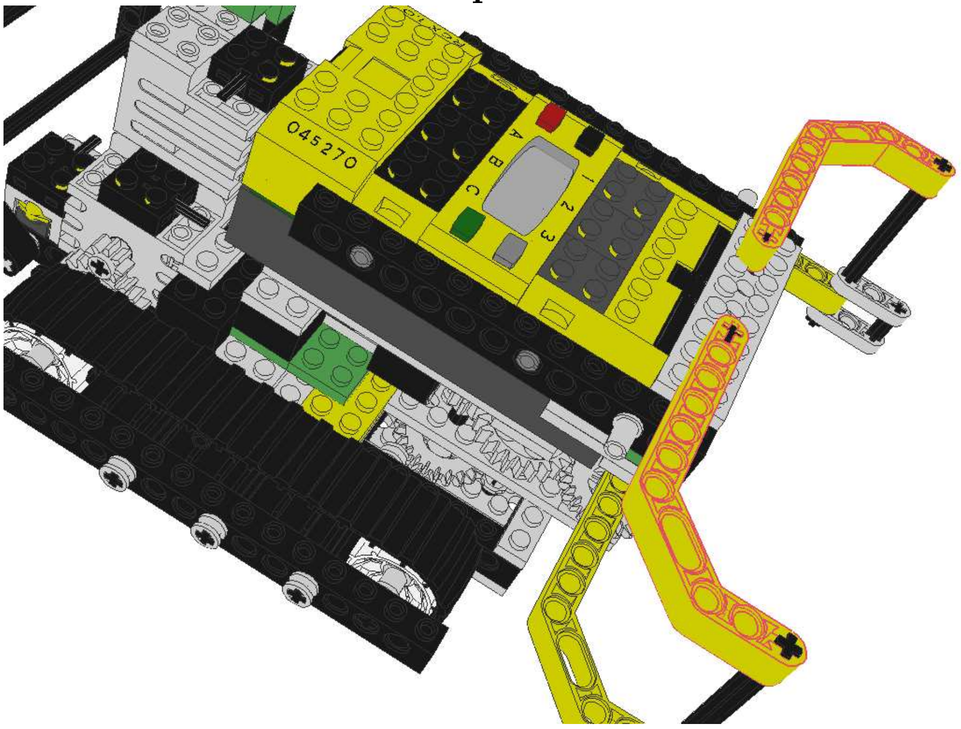
Step 13

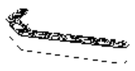


	Descr	Light Green	Black	Gold	Light Gray	Silver
	Plate 2 x 2 Corner (2420)	-	-	-	-	2
	Plate 2 x 2 (3022)	2	-	-	-	-
	Plate 1 x 2 (3023)	-	-	-	-	2
	Technic Pin (3673)	-	2	-	-	2
	Technic Brick 1 x 2 with Hole (3700)	-	2	-	-	-

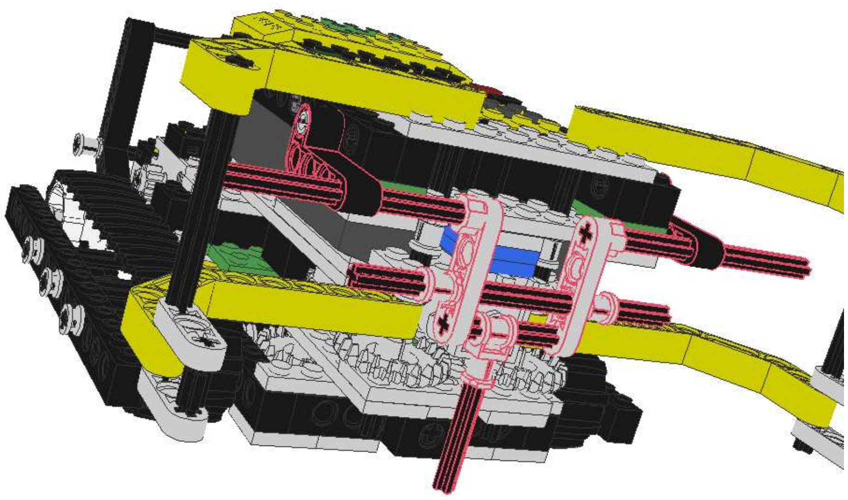
	Descr	Light Green	Black	Gold	Light Gray	Silver
	Plate 1 x 4 (3710)	-	-	-	-	2
	Technic Plate 2 x 8 with Holes (3738)	-	-	-	-	2
	Technic Brick 1 x 12 with Holes (3895)	-	2	-	-	-
	Technic Pin 1/2 (4274)	-	-	-	4	-
	Electric Mindstorms RCX (884)	-	-	1	-	-

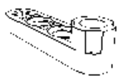
Step 14

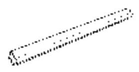


	Descr	**Yellow**
	Technic Liftarm 1 x 11.5 Double Bent (32009)	2

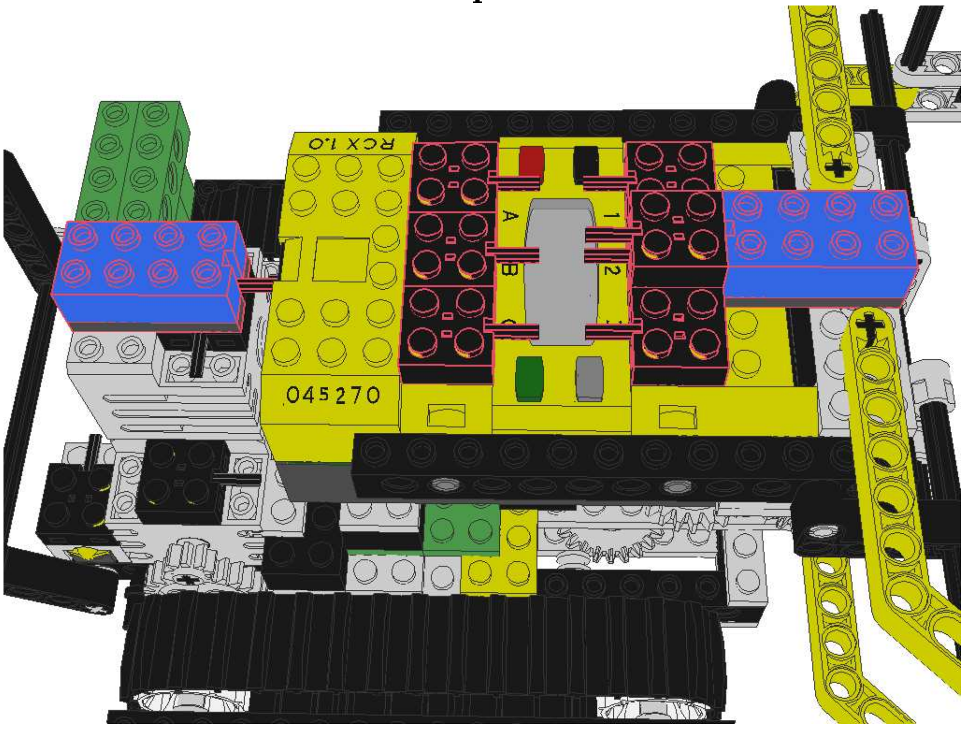
Step 15

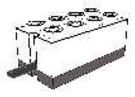
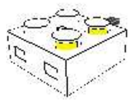


	Descr	Black	Silver
	Technic Liftarm 1 x 4 (2825)	-	2
	Technic Connector with Axle-hole (32039)	-	1
	Technic Liftarm 2 x 4 L Shape (32140)	2	-
	Technic Bush (3713)	-	2

	Descr	**Black**	**Silver**
	Technic Axle 4 (3705)	2	-
	Technic Axle 8 (3707)	2	-
	Technic Axle 10 (3737)	1	-

Step 16

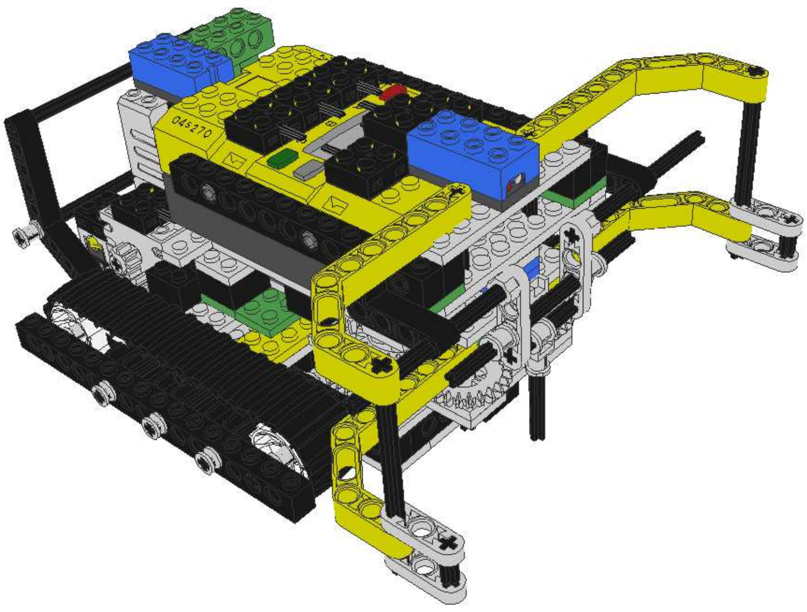


	Descr	Light Blue	Black
	Electric Light Sensor (2982C01)	2	-
	Brick 2 x 2 x 2/3 Electric Wire End (5306)	-	7

Connections ("A", "B" and "C" are the back output slots on the Brick, "1", "2", "3" the front input slots):

Piece	Slot	Piece	Slot
left motor	A	front bumper	3
right motor	C	back bumper	2
gripper motor	B	front light	2
		back light	1

End



Bibliography

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 Intenational Joint Conference on Autonomous Agents and Multi-Agents Systems (AAMAS-02)*, volume 3, pages 1090–1097. ACM Press, New York, NY, USA, 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, USA, 2001.

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.

C. Anderson, G. Theraulaz, and J.-L. Deneubourg. Self-assemblage in insect societies. *Insectes Sociaux*, 49:99–110, 2002.

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.

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, USA, 1999.

T. Balch and R.C. Arkin. Communication in reactive multiagent robotic systems. *Autonomous Robots*, 1(1):27–52, 1994.

E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Santa Fe Institute Studies in the Sciences of Complexity. Oxford University Press, New York, NY, USA, 1999.

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, USA, 2002.

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

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.

A. Castano, W.-M. Shen, and P. Will. CONRO: towards deployable robots with inter-robot metamorphic capabilities. *Autonomous Robots*, 8(3): 309–324, 2000.

G.S. Chirikjian. Kinematics of a metamorphic robotic system. In E. Straub and R. Spencer Sipple, editors, *Proceedings of the International Conference on Robotics and Automation (ICRA94)*, volume 1, pages 449–455. IEEE Press, New York, NY, USA, 1994.

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

D. Goldber 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, USA, 2000.

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

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.

O. Holland and C. Melhuish. Stigmergy, self-organization and sorting in collective robotics. *Artificial Life*, 5(2):173–202, 1999.

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.

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, USA, March 2002.

A. Kamimura, S. Murata, E. Yoshida, H. Kurokawa, K. Tomita, and S. Kokaji. Self-reconfigurable modular robot - experiments on reconfiguration and locomotion. In *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '01)*, pages 606–612. IEEE Press, New York, NY, USA, 2001.

H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, and H. Matsubara. Robocup: a challenge AI problem. *AI Magazine*, 18(1), 1997.

K. Kotay, D. Rus, M. Vona, and C. McGray. The self-reconfiguring robotic molecule. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA-98)*, pages 424–431. IEEE Press, New York, NY, USA, 1998.

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.

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.

S. Nolfi and D. Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines.* MIT Press/Bradford Books, Cambridge, MA, USA, Cambridge, MA, USA, 2000.

A. Pamecha, C. Chiang, D. Stein, and G. Chirikjian. Design and implementation of metamorphic robots. In *Proceeding of the ASME Design Engineering Technical Conference and Computers in Engineering Conference.* ASME Press, Fairfield, NJ, USA, 1996.

W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes: The Art of Scientific Computing.* Cambridge University Press, Cambridge, UK, second edition, 1992.

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, USA, 2002.

D. Rus and M. Vona. Crystalline robots: self-reconfiguration with compressible unit modules. *Autonomous Robots*, 10(1):107–124, January 2001.

M. Schneider-Fontán and M.J. Mataric. A study of territoriality: The role of critical mass in adaptive task division. In P. Maes, M.J. Mataric, 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, USA, 1996.

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

J.H. Sudd. How insects work in group. *Discovery*, 24(6):15–19, 1963.

R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, USA, Cambridge, MA, 1998.

C. Ünsal, H. Kiliççöte, and P. K. Khosla. A modular self-reconfigurable bipartite robotic system: implementation and motion planning. *Autonomous Robots*, 10(1):23–40, 2001.

E.O. Wilson. *The Insect Societies.* Harvard University Press, Cambridge, 1971.

M. Yim. Locomotion with a unit-modular reconfigurable robot. Thesis CS-TR-95-1536, Department of Computer Science, Stanford University, CA, USA, January 1995.

M. Yim, Y. Zhang, J. Lamping, and E. Mao. Distributed control for 3D metamorphosis. *Autonomous Robots*, 10(1):41–56, 2001.