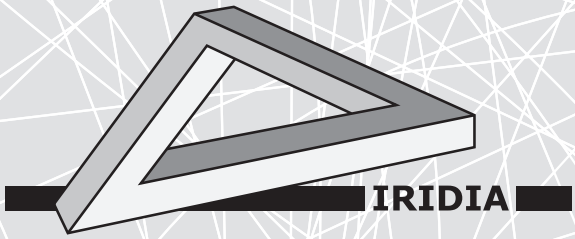


Université Libre de Bruxelles

Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

Adaptive Swarm Robotics in Rough Terrain Navigation

Rehan O'Grady

IRIDIA – Technical Report Series

Technical Report No.
TR/IRIDIA/2005-017

October 2005

IRIDIA – Technical Report Series
ISSN 1781-3794

Published by:

IRIDIA, *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*
Université Libre de Bruxelles
Av F. D. Roosevelt 50, CP 194/6
1050 Bruxelles, Belgium

Technical report number TR/IRIDIA/2005-017

Revision history:

TR/IRIDIA/2005-017.001 October 2005

Adaptive Swarm Robotics in Rough Terrain Navigation

by

Rehan O'Grady

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

Supervised by

Marco Dorigo, Ph.D.

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

August, 2005

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

A recent trend in robotics research has been the application of the social insect metaphor to groups of autonomous robots. This research field is known as swarm robotics. In this DEA thesis we investigate the application in the swarm robotics context of an important collective mechanism observed in social insects - *functional self-assembly*. Self-assembly is the process in which autonomous agents connect to one another to form larger aggregate structures. If a group of autonomous agents can self-assemble in response to environmental conditions, then they are said to display adaptive or functional self-assembly.

We present our design of a swarm robotic controller with the capacity for functional self-assembly. Our system is built on the SWARM-BOT robotic platform. We describe a series of experiments designed to test the adaptive capabilities of our self-assembling system. All experiments were performed using real robots. The task we consider requires a group of robots to navigate over an area of unknown terrain towards a target light source. If possible, the robots should navigate to the target independently. If, however, the terrain proves too difficult for a single robot, the group should self-assemble into a larger entity and collectively navigate to the target. Our results indicate for the first time that self-assembly is a valid adaptive response mechanism for a physical multi-robot system.

Acknowledgments

I would like to thank my supervisor Marco Dorigo. I would also like to thank Roderich Gross for his help and guidance. I am grateful to all my colleagues for being so helpful and for making IRIDIA such a pleasant working environment.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Overview

Swarm robotics is a rapidly expanding area in robotics research. In the last two decades multi-robot systems have already been the focus of much dedicated research. Swarm robotics involves the study of a particular type of multi-robot system. Taking inspiration from social insect behaviour, researchers in swarm robotics build robotic systems using *swarm-intelligence* [10, 9] control principles such as decentralisation of control and use of local information. Swarm robotics research provides insight into the mechanisms of swarm intelligence in the animal kingdom. At the same time, it holds the promise in the not too distant future of cheap, robust and flexible robotic systems with potential applications from nanosurgery to habitat construction in space.

In this DEA thesis we investigate the application to swarm robotics of a collective mechanism used by social insects. The mechanism we have chosen to study is *functional self-assembly* [65]. We define self-assembly as the process through which separate autonomous agents form a larger group entity by physically connecting to one another. The group is said to display functional self-assembly if the agents can autonomously choose to self-assemble in response to the demands of their task and environment.

We conduct our investigation using the SWARM-BOT robotic platform [49, 27]. This recently developed system consists of a number of autonomous robotic agents called *s-bots*. The most innovative aspect of the SWARM-BOT system lies in the *s-bots*' ability to physically connect to one another in order to form a larger group entity termed a *swarm-bot*. A *swarm-bot* has the potential to complete tasks impossible for a single *s-bot* — for

example to cross chasms into which a single *s-bot* would fall or to overcome hills too steep for a single *s-bot*.

To pursue our investigation, our first step was to design an experiment which would require a group of *s-bots* to make adaptive use of self-assembly — i.e. to display functional self-assembly. The next step was to develop a distributed swarm control mechanism which would allow the s-bots to tackle the task we gave them. Finally, we analysed the performance of our system. This analysis also provided indications of potential directions for future research.

The experiment we designed requires a group of *s-bots* to navigate towards a target light source over unknown terrain. The *s-bots* must analyse the terrain they are traversing and 'decide' whether or not it is necessary for them to self assemble. We used two different environments in our experiments — one in which the *s-bots* are capable of navigating independently to the target light source, and one in which the *s-bots* have to self-assemble in order to reach the target and complete their task (see Figure 2.7).

The structure of this DEA thesis is as follows. In the remainder of this introductory chapter we present a review of our field and of related research. In chapter two we discuss our experimental setup. We describe in detail the SWARM-BOT robotic platform on which we conducted our experiments and the nature of the task and environment. In chapter three we present results of the experiments we conducted, and analyse the performance of our controller. In chapter four we give details of our ongoing research in this area. In chapter five we present our conclusions.

1.2 Robotic Control Architectures

Swarm robotics research is conducted with the long term goal of building cheap, scalable and robust robotic systems. The design principles underlying swarm robotic systems reflect these goals. Swarm robotics control mechanisms are usually distributed, heterogeneous and allow the use of local information only. To understand the need for these design principles, and indeed to understand the rational for Swarm robotics in general, it is necessary to have an appreciation for alternative control paradigms. In this section, therefore, we present an overview of robotic control architectures - their strengths and weaknesses - and show where swarm robotics fits into the overall picture.

The Deliberative / Planning Approach

Traditional robotic control architectures rely on top-down planner-based or deliberative strategies. These strategies typically use a centralised world model for verifying sensory information and generating actions in the world [35, 19, 50, 43]. The information in the world model is used by the controller to produce an appropriate sequence of actions (or plan) for the agent.

Processing information in a complex noisy real world environment can, however, be prohibitively difficult. For this reason most purely deliberative systems have had limited success and their use has been largely restricted to simplified artificial environments. These environments are usually of limited complexity and are designed to make sensory input less noisy [53]. Even when a useful representation can be extracted from sensory information, changes in the environment can require frequent re-planning. The cost of this re-planning is often prohibitive for complex systems. Planner-based approaches have been criticised for scaling poorly with the complexity of real-world problems, and making real-time reaction to sudden world changes impossible. For a critique of this representational approach we refer the reader to two seminal articles by Brooks [13, 12].

The Reactive Approach

Several methodologies for achieving real-time performance in autonomous agents have been proposed. The purely reactive bottom-up approach is a technique that has been successfully implemented in a number of systems. This approach involves embedding the agent's control strategy into a collection of preprogrammed condition-action pairs with minimal internal state [11, 1, 20]. Reactive systems maintain no internal models. Typically, they apply a simple functional mapping between stimuli and appropriate responses, usually using some form of lookup mechanism.

These functional mappings rely on a direct coupling between sensing and action as well as fast feedback from the environment. Purely reactive strategies have proven effective for a variety of problems that can be completely specified at design-time. Such systems have proven particularly successful in terms of run-time efficiency due to their minimal computational overhead. However, a major drawback is that their limited representational power results in a lack of run-time flexibility.

Hybrid Approaches

Both deliberative and reactive systems have severe runtime shortcomings. Deliberative systems cope badly with noisy and/or dynamically changing environments. Reactive systems have little or no flexibility.

Hybrid architectures represent one mechanism to try and combine the best aspects of the reactive and deliberative approaches. Hybrid systems usually employ a reactive system for low-level control and a planner for higher-level decision making. Much research has been conducted in hybrid systems. This includes (but is not limited to) reactive planning or reactive execution used in Reactive Action Packages (RAPs), PRS (Procedural Reasoning System), Schemas [3], Internalised Plans (Payton 1990) [57], Contingency Plans [21].

The Behaviour Based Approach

The behaviour based approach has been proposed as an alternative way to avoid the run-time problems of the deliberative and purely reactive approaches [62, 44, 59, 36]. The behaviour based approach is characterised by its modular approach whereby control of the system is split among a number of different behaviour modules. Individual behaviour modules are not linked in a rigid serial or top down structure (which would mean that the overall system reduces to one that could be implemented using a more traditional centralised methodology). Instead, behaviour based systems require some form of behaviour arbitration between modules. This arbitration is usually a central feature of any behaviour based system.

In addition, behaviours tend to be relatively simple. Behaviours are, nonetheless, more complex than the stimuli-response mechanisms used in the reactive approach. A single behaviour is usually in control of the agent for a period longer than the time span of an atomic action of the controlled agent. Different behaviours interact with each other through the world rather than internally through the system [4]. An example of a behaviour based robot is Toto - a behaviour based navigating and path finding robot. Toto demonstrated some higher level reasoning capabilities at the same time as robust real-time reaction in a non-hybrid behaviour based system [47].

Controllers based on Artificial Neural Networks

Artificial neural networks (henceforth referred to as neural networks) are information processing devices that attempt to imitate the way a human brain works [6]. Their development was inspired by examination of neurons

and synapses that make up the bio-electrical networks in the brain. A neural network is a connected network of relatively simple processing elements. The global behaviour is determined by the connections between the processing elements and element parameters.

A neural network can be used as a robotic control mechanism. In such controllers, the robot's sensors provide input to the neural network. The organisation and weights of the connections determine the output of the network. The output is applied to the robot's actuators.

Finding an appropriate set of connections and parameters for a neural network is a complex non-linear task. Instead of directly programming neural networks, therefore, some form of learning mechanism is often used [55].

Recently, a technique known as artificial evolution has been applied to the development of robotic neural network controllers with some success. This technique involves using a darwinian paradigm to evolve generations of neural networks to solve a particular task. The replication of individuals from generation to generation is based on a fitness function that somehow gives a measure of well a neural network controller is able to solve the task [54]. In principle, individuals from progressive generations will be increasingly well adapted to solve the task. This technique is often used in the development of swarm-robotic controllers.

1.3 Multi Robot Systems

Much research has been conducted in multi-robot systems in the last two decades. Some tasks are particularly well suited to being carried out in parallel by teams of robots. Even for tasks that can be successfully carried out by traditional 'monolithic' robots, the multi robot approach can often be valuable, providing gains in robustness, flexibility and costs. The design of multi robot systems, however, presents a whole raft of new challenges. In particular, the interactions between the individual agents must be considered.

1.3.1 Potential Advantages of Multi Robot Systems

Parallelism

Multi robot systems are capable of doing several (sometimes different) things at the same time. This can greatly increase the efficiency of a system especially when the task itself is inherently parallel.

Geographical Distribution

Geographical distribution is one type of inherent parallelism that can be exploited by multi-robot systems. Multi robotic systems can be effective in disparate locations at the same time. This can be useful when the task itself contains many subparts, each of which is distributed. Even when the task is not distributed, geographical distribution of robotic agents can allow the system to get more meaningful sensory input not available to a single robot examining the task from a single location or perspective [41].

Task Decomposition

Task decomposition is a strategy that can enable parallel task execution. Some tasks can be divided into sub-tasks. Multi robot systems are well suited to tasks which can be broken down like this. Of course, methods for task decomposition and task allocation must be incorporated into the system [34].

Design Costs and Versatility

Multi-robot systems are often made up of numerous simple robots with identical hardware specifications. Manufacturing many identical simple robots is usually cheaper than creating a single complex robot. Furthermore, a single complex robot will often be tailor made for a particular task. A well designed multi robot system will make use of modular generic components that may be applicable in different problem domains. As well as adding versatility, this can provide cost savings in the longer term.

Robustness and Fault Tolerance

A distributed robotic system may be better able to adapt to changing task requirements and environmental factors than a single robot. In addition, multi robot systems also often incorporate an element of redundancy. A single robot system will often fail if a single component fails. Multi robot systems can often continue to function when parts of robots or even whole robots fail.

Swarm robotics research, in particular, is dedicated to the design of systems with the characteristics of flexibility and robustness. To achieve this goal, swarm robotics mimics the mechanisms that govern social insect colonies.

1.3.2 Control Architectures for Multi Robot Systems

Centralised vs Distributed Control

Multi robot systems can be divided into those which are controlled centrally, and those in which the control is decentralised. The latter class of system can be referred to as distributed control architecture systems. In a centrally controlled system only a single control agent is present. This agent is solely responsible for deciding the actions of all of the individual robotic agents. By contrast, in a distributed system each agent is responsible for its own control. It is possible to have a control system that uses both centralised and distributed control mechanisms. For example each agent can be capable of acting independently to some degree, while still subordinate to a centralised control system that can override the individual agents when necessary [32].

Heterogeneous and Homogeneous Systems

A further distinction can be drawn between multi agent systems that are comprised of identical agents (homogeneous) or different specialised agents (heterogeneous). Systems can be considered heterogeneous if the robotic agents differ either mechanically or behaviourally (different control). The majority of multi robot research has been carried out to date using mechanically and behaviourally homogeneous systems. Balch investigated the performance enhancement potential of using heterogeneous controllers (the robots he used were still mechanically homogeneous) [8]. An example of a mechanically heterogeneous system is the ALLIANCE multi robot system. ALLIANCE consists of a distributed control architecture for robot teams where team members can be either legged or wheeled robots [56].

1.3.2.1 Swarm Robotic Control Architecture

Swarm robotics is an area of collective robotics that takes inspiration from social insect behaviour. Swarm robotics tries to take advantage of the mechanism of self-organisation and emphasises *swarm intelligence* [10] control principles such as decentralisation of control and use of local information. Swarm robotics controllers tend to be decentralised and homogeneous.

The controller presented in this DEA thesis is an example of a swarm robotic controller. It is implemented on the SWARM-BOT robotic platform (see section 2.1).

1.3.3 Communication Strategies for Multi Robot Systems

When designing a multi robot system, another important consideration is the form of communication that will be used between individual agents. Of course, communication in any form only becomes meaningful when at least some measure of decentralisation is present.

Direct Communication

Direct Communication is conceptually the simplest inter-robot communication mechanism, but nonetheless tends to require a high level of technical sophistication on the part of the communicating agents. When using direct communication an agent either broadcasts a message or directly communicates with other individual agents. Direct communication can be used in a number of different ways, including task coordination [18, 33], learning and sensing. Jones and Mataric use communication to allow each individual in a group of robots to build an overall picture of the state of task completion [41].

Sensing based communication

Sensing based communication relies on robotic agents perceiving the presence and/or actions of other robotic agents. Mataric coined the term *Kin Recognition* to describe the ability to distinguish fellow agents from other objects in the environment. Sensing of other agents' actions has been studied by Kuniyoshi et al. [42]. Examples of sensing based group behaviour in nature include herding, flocking and schooling behaviours. Swarm robotic systems often make use of sensing based communication. Various studies have been made of flocking, dispersing and pattern formation in the context of multi-robot systems [7, 64].

Stigmergic communication

Stigmergy is a method of communication via modification of the environment [38]. There is no direct communication between the robots. Instead, agents are indirectly aware of each others actions via their effects on the environment. Pheromone trails in social ant colonies is one example of stigmergic communication. Swarm-robotic systems often make use of stigmergic communication. For examples of robotic systems that use this form of communication see [37, 5, 58].

1.3.4 Cooperation in Multi Robot Systems

An important aspect of multi robot systems is the extent to which the individual agents cooperate in order to achieve the given task [46]. We give a brief summary of the two main types of cooperation.

Parallel Execution

Parallel Execution is the simplest form of cooperation. Here the robots do not directly cooperate. The robots merely perform (parts of) the same task at the same time.

Non-Physical Cooperation

Non-Physical Cooperation is cooperation where the robots affect each other's actions (for example through stigmergy or direct communication) during task execution. However no direct physical interaction takes place between the robots.

Physical Cooperation

Physical Cooperation is cooperation where robots physically help each other to carry out tasks or sub tasks. Martinoli and Mondada carried out an experiment where a group of robots were required to pull sticks out of the ground. Due to constraints based on the limited physical abilities of the individual robots, two robots were required to cooperate to remove a single stick [45].

1.3.4.1 Self-Assembly

Self-assembly is a type of physical cooperation that is particularly relevant to this DEA thesis. Self-assembly is the physical assembly of separate autonomous agents into a larger group entity. Systems which make use of a connection mechanism have been studied in the context of *self-reconfigurable robots* [17, 51, 68]. The robotic units (also called modules) are usually designed to operate within a configuration of pre-attached modules; in most of the current systems, individual modules are not capable of autonomous motion.

Functional Self Assembly is the type of self-assembly that we investigate in this thesis. The term was coined by Trianni et al. [65] to describe a key cooperation mechanism of distributed systems. *Functional self-assembly* is

defined as the capacity of a group of autonomous agents to choose whether or not to self-assemble on the basis of task related and environmental factors.

Functional self-assembly has been observed in a number of natural systems. Several species of social insect utilise self-assembly to solve problems collectively that are too large or complex for a single insect [2]. (See section 1.4.2.) Note that self-assembly in the animal kingdom is always functional self-assembly. Social insects invariably self-assemble in response to the demands of a specific problem.

1.4 Self-organising Systems

In the previous two sections we have seen where swarm robotic controllers fit in the larger world of robotic control mechanisms. As mentioned previously, swarm robotics takes inspiration from behaviours seen in the animal kingdom and particularly among the social insects. The principal concept borrowed from the social insects is that of self-organisation.

An understanding of self-organisation is thus essential in order to understand swarm robotics. In this section we present an overview of self organisation, and some key examples of self-organisation in the animal kingdom.

1.4.1 The self-organisation phenomenon

Camazine et al. [14] define Self-Organisation as follows:

> Self-organisation is a process in which pattern at the global level of a system emerges solely from numerous interactions among the lower level components of the system. Moreover, the rules specifying interactions among the system's components are executed using only local information, without reference to the global pattern.

The phenomenon of self-organisation has been investigated in many scientific fields, including chemistry, physics, biology, cybernetics and economics. In each case, the common characteristic is that system properties at the global level are not specified directly but emerge from the interactions of lower level components with each other and with the environment.

Self-organised systems have the following features in common:

Figure 1.1: Self-assembly in a natural system. *Œcophilla longinoda* worker ants connect to one another to pull together large leaves during nest construction.

- Without any change of the characteristics of the underlying low level components, such systems may switch between different semi-stable states due to either *intrinsic factors* such as random fluctuations within the system or *extrinsic factors* such as environmental changes [23].
- Little or no knowledge of the global system and or environment is needed by the lower level components. This is because rules specifying interactions among these components only use local information.
- Self-organised systems are usually regulated by positive and negative feedback. Positive feedback can be thought of as an amplifying effect, whereby the existence of a particular condition encourages that same condition to become more prevalent. An example of positive feedback can be seen in stock market investment bubbles. Negative feedback is a mechanism which can stabilise a self organised system, often acting as a counterbalance to keep positive feedback mechanisms in check. Population explosions in animal populations, for example, are often subject to the negative feedback mechanism of food supply being exhausted.

1.4.2 Self Organisation in Natural Systems

Self-organisation has been observed many times in natural systems, particularly in social insect colonies [15, 16, 22, 26, 25, 31, 60, 61]. All of these systems conform to the basic principles of self organisation: interactions

among individuals based on rules of thumb that involve (i) limited cognitive ability and (ii) limited knowledge of the environment. Also observable are random fluctuations as well as positive and negative feedback mechanisms.

Two types of self-organisation in the animal kingdom are particularly relevant to this report: aggregation and self-assembly. An example of aggregation is observed in the bark beetle larvae Dendroctonus micans [24]. These larvae search independently and randomly for a rich feeding site. Once an individual has found a good feeding location, it emits a chemical signal that diffuses in air (this is an example of stigmergic communication). This triggers the aggregation process: in presence of a pheromone gradient, larvae react by moving in the direction of higher concentration of pheromone. As they start to emit pheromone themselves, they reinforce the chemical signal coming from the aggregation site (positive feedback mechanism). The aggregation ends when all the larvae have clustered in one location (negative feedback through exhaustion of larvae).

Self-assembly mechanisms are also observed in many social insects. A review of these observations is given by Anderson et al. [2]. During nest construction, for example, *Œcophylla longinoda* worker ants form connected pulling chains by gripping each other with their mandibles. Examples of rough terrain navigation with the aid of self-assembly include the ant species *Solenopsis germinata*, members of which link together to form floating rafts when their nest is flooded [52]. Collectively, they can pull leaves together that are too large and stiff for a single ant to manipulate (see Figure 1.1). *Dolichoderus cuspidatus* ants have been observed forming living bridges of connected ants that other colony members then traverse.

1.5 Rough Terrain Navigation

Rough terrain navigation has always been an important goal for designers of mobile robotic systems. This problem domain is also central to this DEA thesis — the main experiment we conduct requires a group of robots to navigate over terrain of variable roughness. The most important quality of our system is the ability of the group of robots to adapt to the roughness of the terrain. In this section we present an overview of previous and ongoing research into robotic rough terrain navigation.

1.5.1 Rough terrain navigation in specialised single robot systems

Much research effort has been focused on developing specialised articulated rovers. Examples include shrimp robot [29] and the Pathfinder rover used on Mars [63]. Such systems are usually designed for operation by remote control. Some research has also been done on autonomous control [66]. The high level of interest in this area is demonstrated by the yearly DARPA Grand Challenge. This yearly competition is based on a field test and is intended to accelerate research and development in autonomous rough terrain ground vehicles.

1.5.2 Rough terrain navigation with reconfigurable systems

Research in self-reconfigurable robots focuses on building modular systems that are flexible and can walk, creep, and roll in rough environment conditions. Examples of self-reconfigurable systems include PolyBot [67] and CONRO - designed for earthquake search-and-rescue operations [17]. Again, the focus has been on the mechanical capabilities of the systems rather than sensory and autonomous action capabilities.

1.5.3 Rough terrain navigation in multi agent systems

Some researchers have also investigated multiple rovers for all-terrain exploration [30, 28]. These systems try to leverage distributed design paradigms to get higher levels of system robustness and thus better exploration performance.

The system we present in this DEA thesis falls into this category. We develop a multi-robot system which uses swarm mechanisms to detect and overcome rough terrain.

Chapter 2

Experimental Setup

2.1 The S-bot

2.1.1 Overview

This study was conducted on the SWARM-BOT robotic platform [48, 49]. This system consists of a number of mobile autonomous robots (called *s-bots*) which have the ability to physically connect and disconnect from one another. When the *s-bots* are physically connected to each other, the resulting group artifact is referred to as a *swarm-bot*. The physical abilities of a *swarm-bot* increase with the number of constituent s-bots.

2.1.2 *S-bot* mechanical composition

A close up photograph of an *s-bot* can be seen in Figure 2.1. The mechanical structure of the *s-bot* is illustrated in Figure 2.2.

The *s-bot* is close to 12 cm in diameter, and weighs 700 g. It's body is comprised of two main parts — the chassis and the turret. The *s-bot* chassis houses the battery and the traction system. The traction system is made up of tracks and wheels. The tracks on the left and right hand sides of the *s-bot* are independently controlled by different motors. This setup provides the *s-bot* with high stability, efficient on the spot rotation and mobility on terrain of moderate roughness.

The turret makes up the main body of the *s-bot* and is mounted above the traction system. This turret can rotate with respect to the traction system by means of a motorised axis. The turret houses the majority of the *s-bot* sensing systems. A transparent T-shaped around the *s-bot* turret contains 24 coloured LEDs in 8 different positions. In each position there

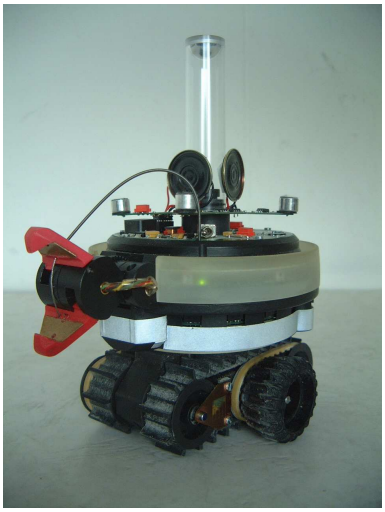


Figure 2.1: The *s-bot*.

is a blue, red, yellow and green LED. This T-shaped ring performs a dual function - it is also shaped so as to be graspable by other *s-bot* grippers.

2.1.3 Sensors

Each *s-bot* is a fully autonomous mobile robot. This is in stark contrast with the majority of self-reconfigurable robotic systems where the basic units include only one or two degrees of freedom and are usually centrally controlled.

To enable autonomous control, each *s-bot* is equipped with an array of sensors. These sensors include an omni-directional colour camera, 16 lateral and 4 down-facing infra-red proximity sensors, 24 light sensors, a 3-axis accelerometer, microphones, two humidity sensors as well as incremental encoders and torque sensors on each of the nine degree of freedom. The accelerometers can be used to detect if the *s-bot* is in danger of toppling over.

The omnidirectional camera is located inside the *s-bot* turret. It takes 360 degree pictures of the *s-bot*'s surroundings by means of a semi-spherical mirror mounted in a transparent cylinder above the turret.

The combination of the camera and the *s-bot* LED ring allows an *s-bot* to communicate its presence and even its internal state to other nearby *s-bots*. Inside the gripper there is an optical light barrier that can detect the presence of objects to be grasped. Other sensors provide the *s-bot* with

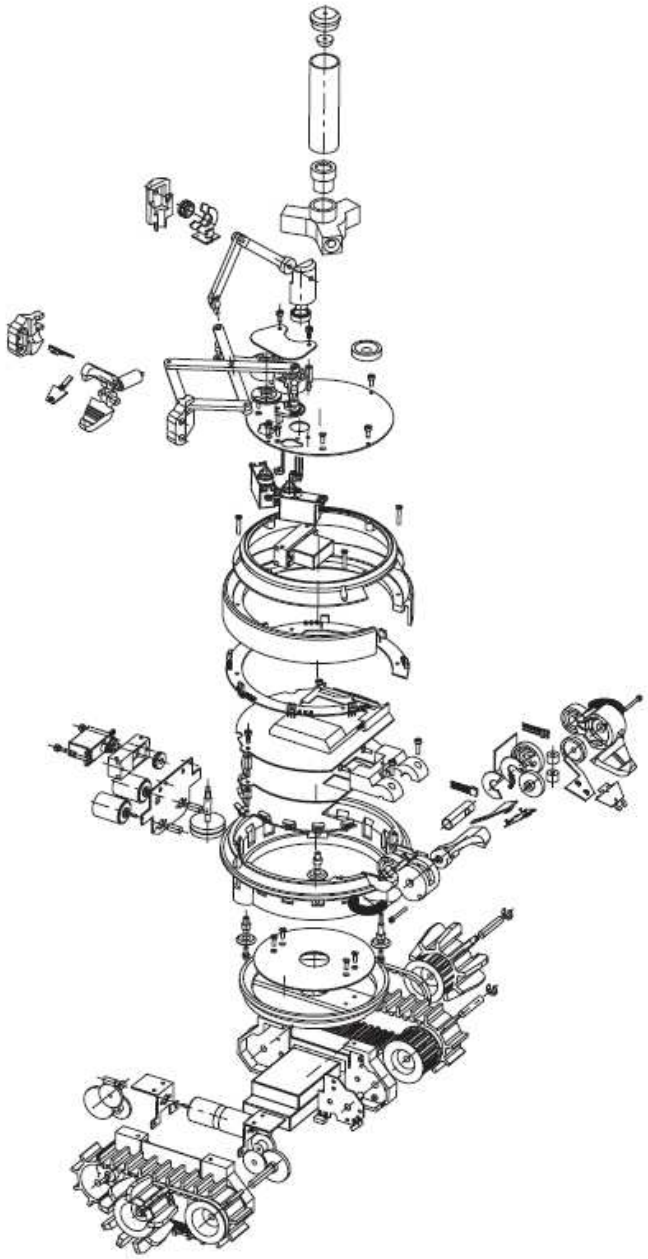


Figure 2.2: Mechanical structure of an *s-bot*. Each *s-bot* is composed of about 100 major parts.

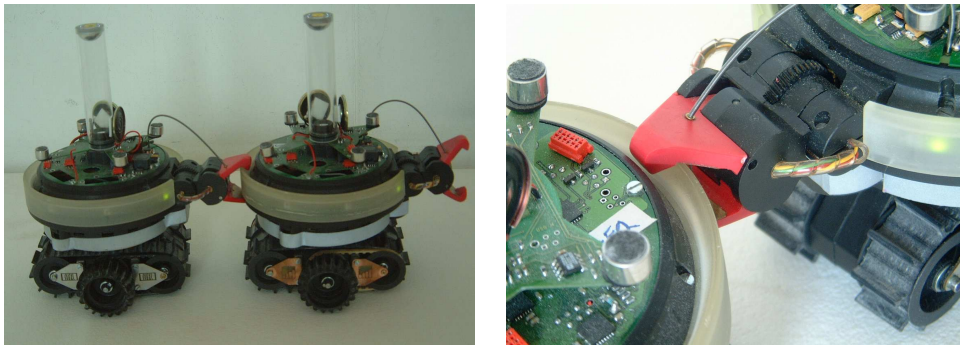


Figure 2.3: Connected *s-bots*

information about its internal motors. This includes positional information (e.g., of the rotating turret) and the torque information (e.g., of forces acting on the tracks).

The *s-bot* sensors operate at a variety of different ranges. Infra-red proximity (active) sensors are limited to very short range operation. The microphones can operate over a much longer range. The camera can be used both for long and short range sensing, depending on the feature extraction algorithm used.

2.1.4 Gripping Mechanism

Rigid connections between s-bots are implemented by a gripper mounted on a horizontal active axis on the turret. The gripper is designed to mesh with the T-shaped connection ring around the *s-bot* turret. If not completely closed, the connection leaves some degrees of freedom. If completely closed, the gripper ensures a rigid connection.

The shape of the gripper enables a very large acceptance area for grasping. This large acceptance area is an important feature of the SWARM-BOT system, as it allows *s-bots* freedom to connect at different angles and in less controlled situations. Again this is in contrast with interconnecting modules in previous self-reconfigurable robotic systems where the exact position of individual modules needs to be known or calculable, thus allowing for precise positioning during interconnection.

Photographs of two connected *s-bots* and a close up of the gripping mechanism can be seen in Figure 2.3. A diagrammatic representation of the *s-bot* gripper is shown in Figure 2.4.

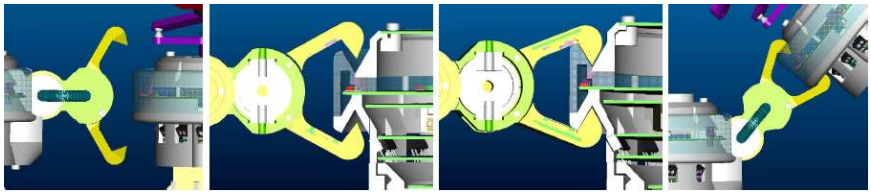


Figure 2.4: *S-bot* gripper. The mechanism by which one *s-bot* connects to another.

2.1.5 CPU, Control Electronics and Software

Distributed swarm-robotics controllers are not usually computationally intensive. However, *s-bots* have many sensors which require fast preprocessing. In addition, sophisticated monitoring and data collection capabilities facilitate software development and experimental analysis. The *s-bots* have thus been equipped with a network of eleven processors, each of them responsible for a sub-task in the system. The most powerful processor, an ARM based processor running LINUX, is in charge of the management of the system, of the processing of the most complex sensors and of the communication with a base station for monitoring purposes.

S-bot control software is written in C or C++. It is copied using ssh and wi-fi protocols onto the *s-bot* before being executed on the *s-bots* native Linux. The controller presented in this DEA thesis was implemented in C++. Each behavioural module (see chapter 3) was implemented in a separate C++ class.

2.2 The Experiment

2.2.1 The Environment

We conduct experiments in two different arenas, referred to as *Environment A* and *Environment B*. Both have dimensions of 240 cm x 120 cm and consist of three distinct areas: two areas of flat terrain (a starting area and a target area) separated by an area of rough terrain (see Figure 2.5). *Environments A* and *B* differ only in the nature of the rough terrain. In *Environment A*, the rough terrain consists of a hill two centimetres high which can be overcome by a single *s-bot* (see Figure 2.7a). In *Environment B* the rough terrain consists of a hill 5 cm high which a single *s-bot* cannot overcome alone (see Figure 2.7b).

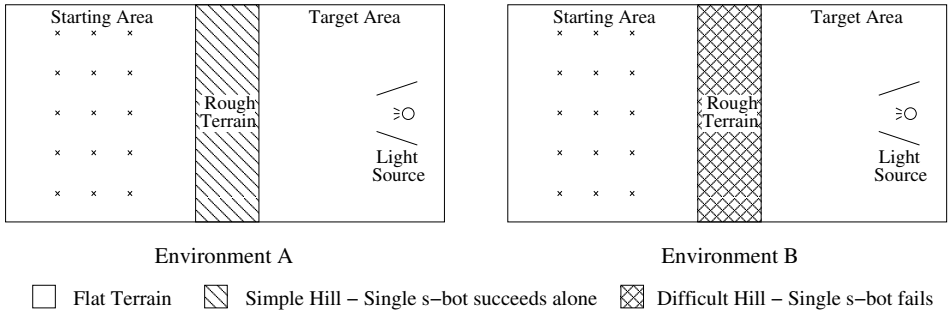
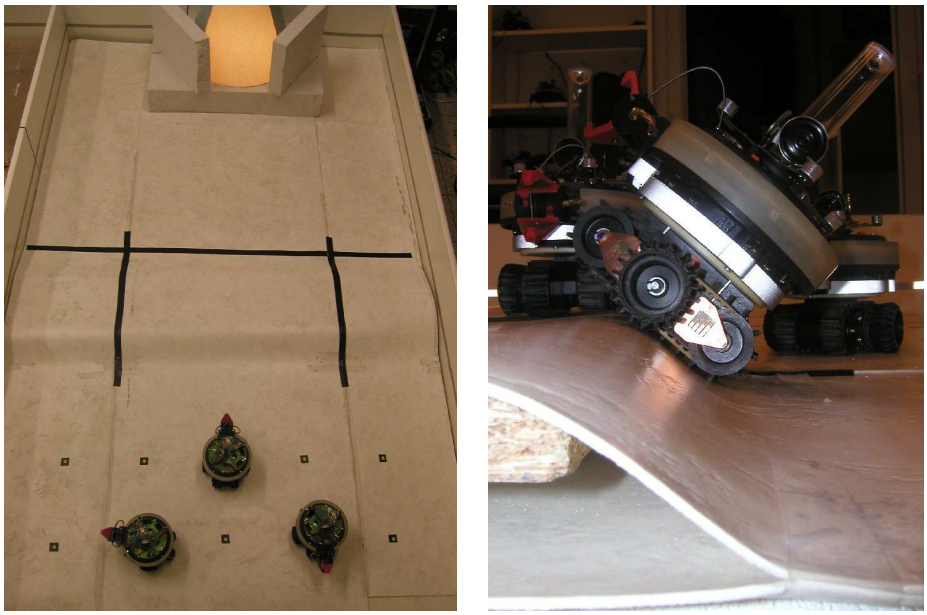
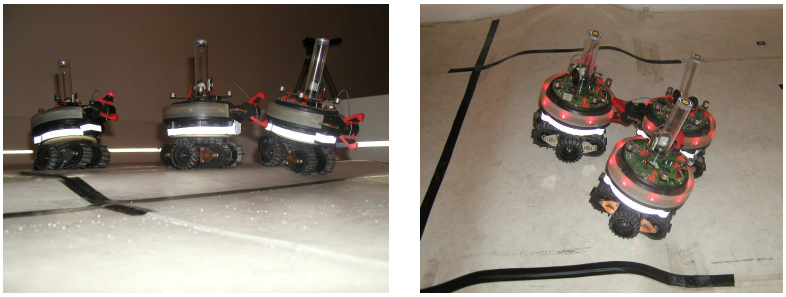


Figure 2.5: Scale diagram of the two experimental environments (view from above). Initially the *s-bots* are placed in the starting area (candidate positions marked by crosses). To complete the task the *s-bots* must enter the target area. In *Environment A* (left figure) the *s-bots* are capable of accomplishing the task independently. In *Environment B* (right figure) it is not possible for the *s-bots* to complete the task unless they self-assemble.



(a) (b)

Figure 2.6: (a) Environment B from above. *S-bots* in random initial positions and orientations. (b) Cross section of Environment B hill. The *s-bot* in the foreground is about to topple backwards.



(a) (b)

Figure 2.7: The Task. (a) When faced with a simple hill the *s-bots* should overcome the hill and navigate to the target independently. (b) When faced with a hill too difficult for a single *s-bot*, the *s-bots* should self-assemble and navigate over the hill to the target source in a connected swarm.

2.2.2 The Task

At the beginning of each trial, the *s-bots* are positioned in the starting area. The initial position of each robot is assigned randomly by uniformly sampling without replacement from a set of 15 specific starting points. The *s-bot*'s initial orientation is chosen randomly from a set of 4 specific directions (Figure 2.6a shows *S-bots* in a random initial configuration). To complete the task the *s-bots* must reach the target area without toppling over.

Figure 2.6b shows a cross section close up of the environment B hill. An *s-bot* has been placed on the hill in a position where it is about to topple over backwards.

The *s-bots* have no a priori knowledge of the environment in which the trial takes place. In *Environment B* the task cannot be accomplished by the *s-bots* independently. To complete the task the *s-bots* must aggregate, self-assemble and coordinate their movements over the rough terrain. In *Environment A* single *s-bots* are capable of accomplishing the task independently. It is unnecessary and inefficient for the *s-bots* to aggregate and self-assemble.

Figure 2.7 shows the different successful task execution strategies for 3 s-bots in Environment A and Environment B. In Figure 2.7a (left figure) the *s-bots* are in Environment A and are navigating independently over the hill to the target light source. In Figure 2.7b (right figure) the *s-bots* are in Environment B and have chosen to self-assemble in order to overcome the hill and reach the target light source.

Chapter 3

Control Strategy

Swarm robotic systems tend to be scalable and robust. Because they rely on the complexities of interactions between individual robots, however, such controllers are often difficult to construct. In this section we present the design and implementation of our swarm robotic controller.

This is the first time that a swarm robotics controller that demonstrates functional self-assembly has been implemented on physical robots. Due to the complexities of the real world environment, we used a building block approach - we implemented a collection of simple basic behaviours corresponding to the different phases of functional self-assembly as seen from the perspective of the individual *s-bot*. We developed a behaviour based controller by combining these building blocks.

In keeping with swarm robotic principles, each *s-bot* is fully autonomous. The same controller is executed on all of the s-bots. Our controller is thus a distributed behaviour based controller (see sections 1.2 and 1.3.2).

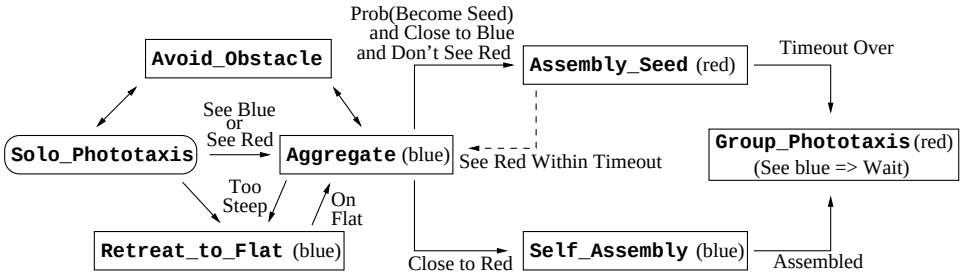


Figure 3.1: Behaviour Based Architecture - Control Flow

Detected Colour	**Possible interpretation**
YELLOW	Target Light Source
BLUE	*S-bot* executing `Retreat_To_Flat` behaviour
BLUE	*S-bot* executing `Aggregate` behaviour
BLUE	*S-bot* executing `Self_Assembly` behaviour
RED	*S-bot* executing `Assembly_Seed` behaviour
RED	*S-bot* executing `Group_Phototaxis` behaviour

Table 3.1: Possible interpretation of camera detected colour objects.

The control flow of the behaviour based *s-bots* is illustrated in Figure 3.1. Limited local communication between the *s-bots* is implemented through the use of colour. Some behaviours have an associated colour. When in one of these states the *s-bot* lights up its coloured LED ring with the appropriate colour. *S-bots* thus gain an indication of the presence and internal states of nearby *s-bots* through camera based colour detection. If an *s-bot* can detect any blue objects, for example, it means that there is at least one *s-bot* either aggregating or assembling in its vicinity. Possible interpretations for detected objects of a given colour are shown in table 3.1.

The *s-bots* always start in `Solo_Phototaxis` behaviour — the *s-bots* start by independently performing phototaxis towards the target light source. Based on its sensory input, an *s-bot* will start trying to aggregate if it determines that the task requires cooperation. This will happen either if it detects a hill it cannot pass alone or if it detects a blue object which indicates the presence of another *s-bot* executing `Retreat_To_Flat` behaviour, `Aggregate` behaviour or `Self_Assembly` behaviour. The assumption is that if another *s-bot* is executing one of these behaviours, it must already be aware of the presence of such a hill. `Self_Assembly` behaviour is triggered when one aggregating robot probabilistically becomes the seed for the assembly (initiates `Assembly_Seed` behaviour and turns red). Assembled *s-bots* switch to `Group_Phototaxis` behaviour. *S-bots* will start navigating collectively to the target area once they can no longer detect any blue objects (aggregating or assembling *s-bots*).

3.1 Camera Image Processing

The *s-bot* camera plays a key role in the controller we have developed. As mentioned above, coloured object detection is essential for the sensing based communication that enables the swarm behaviour. In this section we describe how the controller processes and uses the images received from the camera.

3.1.1 Coloured Object Detection

The Linux camera drive produces JPEG images. Using the following steps, these JPEG images are converted into an array of objects with associated colour and direction.

- Receive JPEG image (640 pixels x 480 pixels)
- Perform colour segmentation of each pixel in RGB colour space. Each pixel is associated with one of three colours - red, yellow or blue.
- Divide image into grid of 16 pixel x 16 pixel blocks. Resulting grid has dimensions: 40 blocks x 30 blocks.
- Perform majority voting algorithm for pixels of each block. As a result each block is associated with a single colour.
- Use erosion and dilation based algorithm to refine block-colour association
- Return array of blocks with associated tuple: (colour, direction of block relative to centre of image).

The last step of the above process thus produces the required output: an array of objects. Each object has an associated colour (red, yellow or blue) and an associated direction (0 degrees - 360 degrees).

3.1.2 Target Direction Noise Filtering

Whenever the *s-bot* has to perform phototaxis, it must be able to determine the direction of the target light source. The light source is identified by the above coloured object detection process as a cluster of yellow objects. However, reflections in the arena (from arena walls, the arena floor and other *s-bots*) are also often detected as yellow objects by the coloured object detection process.

Algorithm 1 Camera Target Direction Noise Filter Algorithm

1: getObjectsFromCamera( )
2: *biggestSegment* ← getSegmentWithMostObjects( )
3: *closestObject* ← getClosestObjectInSegment(*biggestSegment*)
4: *direction* ← getObjectDirection(*closestObject*)

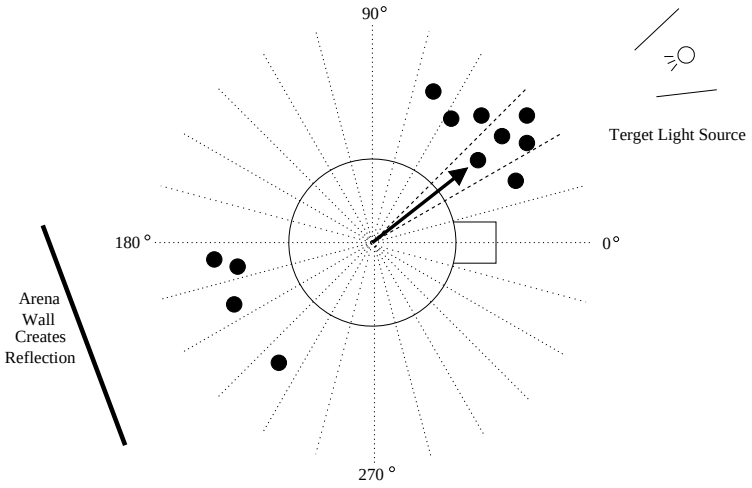


Figure 3.2: Camera based object detection and target direction filtering. The black circles represent detected yellow objects. The calculated target direction is shown by the bold arrow.

To cope with this noise, a target filtering algorithm was used (see Algorithm 1. This is illustrated in Figure 3.2. For simplicity we only represent detected yellow objects. These are represented in the diagram as small black circles. The algorithm divides the *s-bot*'s horizontal plane into 24 segments of 15 degrees each. The number of yellow objects in each segment is counted, and only the segment with the most yellow objects is considered. More yellow objects are usually detected from the the light source than from any of the reflections. This is because the light source is naturally brighter than any of its reflections.

Having established the plane segment containing the light source, the filtering algorithm picks the nearest object to the *s-bot* inside that segment. The direction of this object is taken to be the direction of the target light source. This resulting direction is shown in Figure 3.2 by the bold arrow.

3.2 Behaviour Arbitration

Behaviour arbtitration is handled by an independent control module (not represented in Figure 3.1). The logic for this simple module is presented in Algorithm 2. Note that the individual behaviours choose when to hand control over to another behaviour. Note also that the control step time length is a tenth of a second.

3.3 `Solo_Phototaxis` Behaviour

`Solo_Phototaxis` is the starting behaviour for each *s-bot*. The *s-bot* performs phototaxis towards the target light source (the direction of which it determines using its camera). On flat terrain the *s-bot*'s maximum track speed is constant. On rough terrain the *s-bot* reduces its maximum track speed as a linear function of its inclination (as measured by the accelerometers). This is to prevent the s-bot toppling before `Retreat_to_Flat` behaviour has time to be initiated.

The algorithm used in this behaviour is presented in Algorithm 3. The hard turn in the algorithm is performed by rotating both *s-bot* tracks in opposite directions. The soft turn is performed by rotating one *s-bot* track at MAX-SPEED, and the other *s-bot* track at MAX-SPEED - SOFT-TURN-INCREMENT.

The actual values used for MAX-SPEED and SOFT-TURN-INCREMENT were optimised on the basis of pre-experimental evaluation. Table 3.2 shows the exact values used for these constants, as well as for constants used in other behaviours.

Algorithm 2 The Behaviour Arbitration Module

1: $currentBehaviour \leftarrow$ `Solo_Phototaxis`
2: **loop**
3: executeBehaviour($currentBehaviour$)
4: wait(maximum(100 milliseconds, $behaviourExecutionTime$))
5: $currentBehaviour \leftarrow currentBehaviour$.getNextBehaviour( )
6: **end loop**

Constant Name	Constant Value
MAX-SPEED	22
SOFT-TURN-INCREMENT	5
MAX-SLOPE	30
RETREAT-TIMEOUT	5
ASSEMBLY-SEED-TIMEOUT	5
PROXIMITY-THRESHOLD	10
BECOME-SEED	0.04

Table 3.2: Value constants used in *s-bot* control. Generated manually through trial and error optimisation.

3.4 `Avoid_Obstacle` Behaviour

`Avoid_Obstacle` behaviour is initiated from `Solo_Phototaxis` behaviour or `Aggregate` behaviour when the readings from the *s-bot*'s 14 proximity sensors exceed a certain threshold. The *s-bot* determines the direction of the obstacle by comparing values from the different proximity sensors. The *s-bot* moves away from the obstacle until the obstacle is no longer detected.

More precisely, the spatial relationship of the obstacle to the *s-bot* is represented with a vector integrating the direction and magnitude of all 14 of the *s-bot*'s proximity sensors. This vector is calculated using equations 3.1 and 3.2.

The control logic for this behaviour is presented in Algorithm 4.

$$ObstacleVector_x = \sum_{prox=1}^{14} -\cos\left(Direction_{prox}\right) * Magnitude_{prox} \quad (3.1)$$

$$ObstacleVector_y = \sum_{prox=1}^{14} -\sin\left(Direction_{prox}\right) * Magnitude_{prox} \quad (3.2)$$

Based on this vector the *s-bot* determines if the obstacle is ahead and to the left, ahead and to the right, behind and to the left or behind and to the right. The *s-bot* then executes a soft turn (see section 3.3) away from the obstacle

Algorithm 3 `Solo_Phototaxis` Behaviour

```
1: slope ← getSlope( )
2: if slope > MAX-SLOPE then
3:    switchBehaviour( Aggregate )
4: else if detectColourObject( BLUE ) or detectColourObject( RED )
   then
5:    switchBehaviour( Retreat_To_Flat )
6: else
7:    getTargetDirection( )
8:    if targetHeading > 20 deg then
9:       hardTurnToTarget( )
10:   else
11:      speed ← MAX-SPEED * ( MAX-SLOPE - slope ) / MAX-SLOPE
12:      softTurnToTarget( speed )
13:   end if
14: end if
```

Algorithm 4 `Avoid_Obstacle` Behaviour

```
1: repeat
2:    proximityReadings ← getProximityReadingsFromSensors()
3:    obstVectX ← getObstacleVectorX( proximityReadings )
4:    obstVectY ← getObstacleVectorY( proximityReadings )
5:    softTurnAwayFromVector( obstVectX, obstVectY )
6: until max( proximityReadings ) < PROXIMITY-THRESHOLD
```

3.5 `Retreat_to_Flat` Behaviour

`Retreat_to_Flat` behaviour is initiated from `Solo_Phototaxis` behaviour or `Aggregate` behaviour when information from the *s-bot*'s accelerometers indicate that the *s-bot* is in danger of toppling over. The *s-bot* receives information about its inclination in two planes from two separate accelerometers. The *s-bot* adds these two values together. If this combined value is greater than MAX-SLOPE, the *s-bot* determines that it is on a hill too steep to traverse alone. Constant values have been manually optimised (see Table 3.2).

Once in `Retreat_to_Flat` behaviour, the *s-bot* determines the direction of the slope with respect to its heading. The *s-bot* uses this information to reverse down the slope as directly as possible - using soft turns to try and keep the slope of the hill directly ahead. Once the *s-bot* is again on flat terrain, the *s-bot* reverses away from the rough terrain, then rotates so that it is facing away from the slope.

The control logic for this behaviour is presented in Algorithm 5. Note that the $downHillVect$ variable will always be calculated since the `Retreat_To_Flat` behaviour will only be executed if the steepness threshold has already been exceeded.

Algorithm 5 `Retreat_To_Flat` Behaviour

```
1:  activateColourRing( BLUE )
2:  loop
3:    pitch ← getFrontBackInclination()
4:    roll ← getLeftRightInclination()
5:    totalInclination ← pitch + roll
6:    if totalInclination > MAX-SLOPE then
7:      downHillVect ← calculateDownHillVector( pitch, roll )
8:      setTracksMoveDirection( downHillVect )
9:    else
10:     setTracksMoveDirection( downHillVect, RETREAT-TIMEOUT )
11:     switchBehaviour( Aggregate )
12:   end if
13: end loop
```

3.6 `Aggregate` Behaviour

The control logic for `Aggregate` behaviour is presented in Algorithm 6. While executing `Aggregate` behaviour the *s-bots* locate and then approach

each other as a precursor to self-assembly.

The *s-bot* conducts a random walk until it detects either a blue or a red object. If the *s-bot* detects a red object, (i.e. another *s-bot* that is executing `Assembly_Seed` behaviour, `Self-Assembly` behaviour or `Group_Phototaxis_Behaviour`) the *s-bot* will switch to `Self_Assembly` behaviour. If the *s-bot* detects a blue object, (i.e. another *s-bot* that is executing `Retreat_To_Flat` behaviour or `Aggregate` behaviour), the *s-bot* will approach the blue object, stop moving and wait until it sees a red object.

Algorithm 6 `Aggregate` Behaviour

```
activateColourRing( BLUE )
loop
  if detectColourObject( RED ) then
    Close to Red → switchBehaviour( Self_Assembly )
    Far from Red → approachRed( )
  else if detectColourObject( BLUE ) then
    Prob( BECOME-SEED )→ switchBehaviour( Assembly_Seed )
    Prob( 1 - BECOME-SEED )→ approachBlue( )
  else
    randomWalk( )
  end if
end loop
```

3.7 `Self_Assembly` Behaviour

The control logic for this behaviour is presented in Algorithm 7

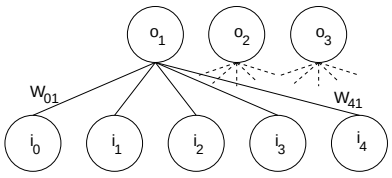
Function f maps sensory input to motor commands. It is implemented by a neural network which has been designed by artificial evolution in a previous work [40]. The `Self_Assembly` behaviour has been extensively tested with swarms of up to 16 physical s-bots in another previous work [39].

3.8 `Assembly_Seed` Behaviour

`Assembly_Seed` behaviour is initiated probabilistically from `Aggregate` behaviour. Aggregating *s-bots* will switch to `Self_Assembly` behaviour and attempt to self-assemble as soon as they detect a red object. Once aggregating *s-bots* have approached each other, therefore, one of the aggregating

Algorithm 7 `Self_Assembly` Behaviour

```
1: activateColourRing( BLUE )
2: loop
3:    (i1, i2) ← featureExtraction(camera)
4:    (i3, i4) ← sensorReadings(proximity)
5:    (o1, o2, o3) ← f(i1, i2, i3, i4)
6:
7:    if graspingRequirementsFulfilled( o3 ) then
8:       grasp( )
9:       if successfullyConnected( ) then
10:         switchBehaviour( Group_Phototaxis )
11:      else
12:         openGripper( )
13:      end if
14:   end if
15:   applyValuesToTracks( o1, o2 )
16: end loop
```



(a)

$$o_j = \frac{1}{1 + e^{-x_j}}$$

$$x_j = \sum_{n=0}^{4} \omega_{nj} i_n$$

(b)

Figure 3.3: (a) A graphical representation of the feed-forward two-layer artificial neural network (i.e., a perceptron) of the assembly module. i_1, i_2, i_3, and i_4 are the nodes which take input from the *s-bot*'s sensors. i_0 is the bias term. o_1, o_2, and o_3 are the output nodes. (b) The equations used to compute the network output values.

s-bots must become red by switching to `Assembly_Seed` behaviour in order for the self-assembly process to begin.

To prevent two *s-bots* from simultaneously entering `Assembly_Seed` behaviour, the *s-bot* waits for an initial period of 3 seconds to check that no other red objects become visible. If the *s-bot* does see a red object in this initial period, control is passed back to `Aggregate` behaviour. (If two *s-bots* in the same vicinity switch to `Assembly_Seed` behaviour, both will revert to `Aggregate` behaviour). If no red object is seen in this initial period, control is passed to `Group_Phototaxis` behaviour.

The control logic for this behaviour is presented in algorithm 8.

Algorithm 8 `Assembly_Seed` behaviour

```
1: activateColourRing( RED )
2: loop
3:   if withinTimeout( ASSEMBLY-SEED-TIMEOUT ) then
4:     if detectColourObject( RED ) then
5:       switchBehaviour( Aggregate )
6:     end if
7:   else
8:     switchBehaviour( Group_Phototaxis )
9:   end if
10: end loop
```

3.9 `Group_Phototaxis` Behaviour

`Group_Phototaxis` behaviour is initiated from `Self_Assembly` behaviour or `Assembly_Seed` behaviour. If the *s-bot* can see blue objects in the vicinity it remains stationary (the assumption being that other *s-bots* are trying to assemble to the swarm-bot). Otherwise the *s-bot* performs phototaxis to the target.

The control logic for this behaviour is presented in algorithm 9. Group phototaxis is more complicated than the phototaxis required in `Solo_Phototaxis` behaviour. Because the *s-bot* is now part of a *swarm-bot*, the orientation of the turret is fixed. To move towards the target the *s-bot* continually rotates the traction system with respect to the turret so that the tracks remain oriented towards the target. Depending on the angle between the *s-bot* chassis heading and the target direction, the *s-bot* can either move forwards or in reverse towards the target. This means that the

turret/chassis rotation never exceeds 90 degrees in either direction.

The tracks speeds are set so that the chassis moves towards the target. The target direction relative to the chassis is determined before the turret/chassis rotation mentioned above is carried out. Thus the turret/chassis rotation and the track motion work together to ensure that the chassis remains headed towards the target. The process of calculating the *s-bot* turret/chassis rotation and track speeds can be seen in lines 6-11 of Algorithm 9. Details of the rotation and track speed calculation are illustrated in Figure 3.4.

Note that the control mechanism has a behaviour discontinuity point when the target direction is close to 90 degrees away from the chassis heading. This discontinuity is indicated with a dotted line in Figure 3.4. As the target direction passes this point the *s-bot* needs to rotate the chassis so that it is heading backwards instead of forwards to the target (or vise versa). At the same time it reverses the rotation of the tracks. [1]

Algorithm 9 `Group_Phototaxis` behaviour

```
 1: activateColourRing( RED )
 2: loop
 3:   if detectColourObject( BLUE ) then
 4:     setTrackSpeeds( 0, 0 )
 5:   else
 6:     turretTargDirn ← getTargetDirection( )
 7:     rotation ← getTurretRotation( )
 8:     chassisTargDirn ← getRelativeDirection(
        turretTargDirn, rotation )
 9:     newRotation ← calculateRotation( targDirn )
10:     rotateTurret( newRotation )
11:     setTracksMoveDirection( chassisTargDirn )
12:   end if
13: end loop
```

[1] In rare cases this can result in inefficient behaviour due to noisy sensor input when the angle between the target direction and the chassis heading is close to 90 degrees. In this case the *s-bot* can end up constantly changing the direction of motion and having to repeatedly rotate the chassis almost 180 degrees.

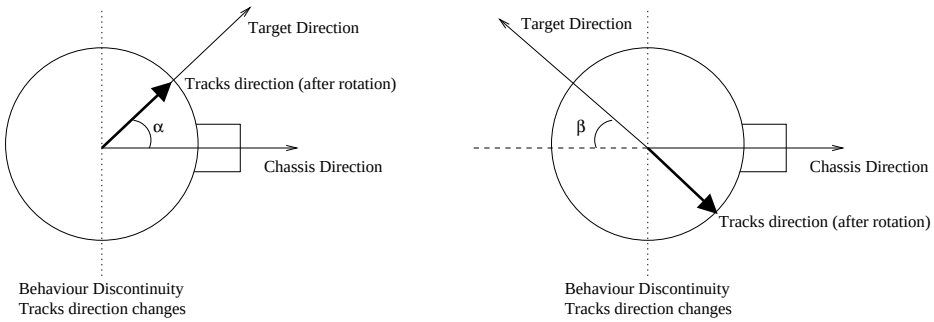


Figure 3.4: *S-bot* turret/chassis rotation based on target direction (view from above). (Left) *S-bot* rotates turret α degrees clockwise. Tracks going forwards. (Right) *S-bot* rotates turret β degrees anticlockwise. Tracks in reverse.

Chapter 4

Results

4.1 Overview

We conducted a series of experiments in two different environments (see Figure 2.5) with groups of 1, 2 and 3 *s-bots*.

4.1.1 Trials with 3 *s-bots* in Environment A

We conducted 20 trials. In every trial all 3 *s-bots* reached the target zone. In 19 out the 20 trials the *s-bots* correctly navigated independently to the target. In a single trial the *s-bots* self-assembled on the down slope of the hill and then performed collective phototaxis to the target. The decision to self-assemble was triggered when one of the *s-bots* misperceived a non-existent blue object.[1]

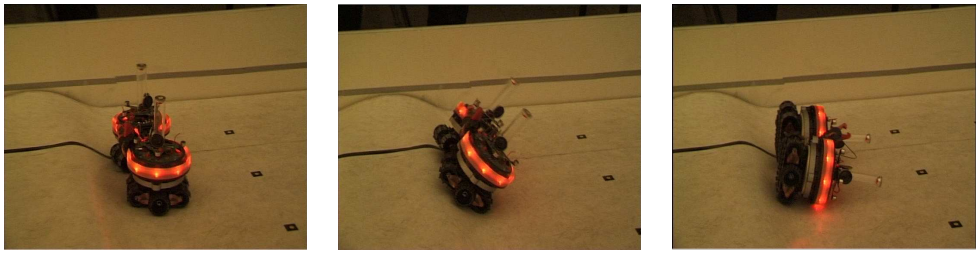
4.1.2 Trials with a single *s-bot* in Environment B

We modified the controller so that the *s-bot* was prevented from switching out of `Solo_Phototaxis` behaviour. The *s-bot* was thus limited to performing phototaxis towards the target taking no account of the terrain encountered.

We conducted 20 trials with a single *s-bot*. The *s-bot* failed to overcome the hill in 20 out of 20 trials. In each trial the *s-bot* reached the hill and then toppled backwards due to the steepness of the slope.[2]

[1] We observed that such misperceptions can occur when the *s-bots* are in the immediate vicinity of the target light source.

[2] We checked that a single *s-bot* was failing due to the intrinsic properties of the slope by repeating this experiment at a number of different constant speeds.



(a) (b) (c)

Figure 4.1: A 2 *s-bot swarm-bot* fails to overcome the Environment B hill. The failure is because the orientation of the *swarm-bot* is parallel to the orientation of the hill. (a) The *swarm-bot* approaches the hill (b) The *swarm-bot* climbs the hill and is starting to topple. (c) The *swarm-bot* has toppled backwards.

4.1.3 Trials with 2 *s-bots* in Environment B

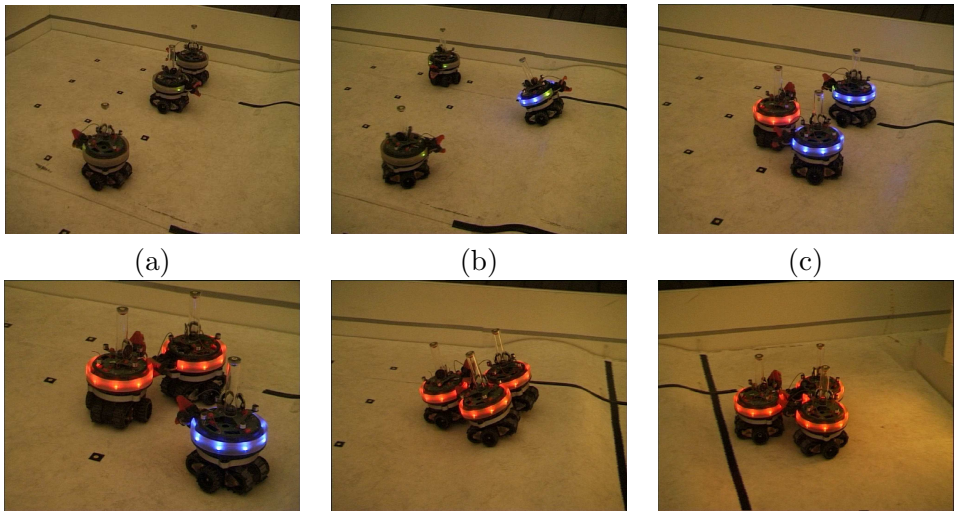
We conducted 20 trials. The *s-bots* successfully detected the slope in every trial. Furthermore the *s-bots* always succeeded in assembling into a 2 *s-bot swarm-bot*. In 13 trials (65%) the *swarm-bot* succeeded in overcoming the hill. In the other 7 trials (35%) the assembled *swarm-bot* failed to overcome the hill. These failures happened when the *swarm-bot* approached the hill with an orientation parallel to that of the hill. Figure 4.1 shows a sequence of photos from one of these unsuccessful trials.

4.1.4 Trials with 3 *s-bots* in Environment B

We conducted 20 trials. The *s-bots* successfully detected the slope in every trial. In 16 trials (80%) all of the *s-bots* successfully self-assembled into a 3 *s-bot swarm-bot*. In each of these 16 trials the 3 *s-bot swarm-bot* went on to successfully reach the target area.

In the remaining 4 trials (20%) the *s-bots* still managed in each case to self-assemble into a *swarm-bot* of 2 *s-bots*. In two of these 4 trials the 2 *s-bot swarm-bot* went on to successfully reach the target area. In the two other trials the 2 *s-bot swarm-bot* was obstructed by the third *s-bot* which failed to self-assemble.

Figure 4.2 shows a sequence of images taken from a typical successful trial.



(a) (b) (c)

(d) (e) (f)

Figure 4.2: (a) The *s-bots* start in a random configuration. (b) A single *s-bot* detects a slope it cannot overcome alone and turns blue (c) Other *s-bots* detect blue colour, turn blue themselves and aggregate. One *s-bot* then seeds the assembly process by turning red. (d) One *s-bot* has assembled to the seed and thus turns red. (e) All *s-bots* are assembled, they collectively overcome the hill. (f) The *swarm-bot* arrives in the target area.

Table 4.1: Percentage of *s-bots* succeeding for stages Self-Assembly (**A**) and Completion of task (**C**). The first row shows the percentage of successful *s-bots*. Subsequent rows show the percentage of *s-bots* that completed stages in groups of 1, 2 or 3 *s-bots* or that failed.

	1 *s-bot* trials		2 *s-bots* trials		3 *s-bots* trials	
	A	**C**	**A**	**C**	**A**	**C**
% Successful (total)	N/A	0.00	100.00	65.00	93.33	86.67
% Successful alone	N/A	0.00	N/A	0.00	N/A	0.00
% Successful in 2 *s-bot swarm-bot*	N/A	N/A	100.00	65.00	13.33	6.67
% Successful in 3 *s-bot swarm-bot*	N/A	N/A	N/A	N/A	80.00	80.00
% Failed	N/A	100.00	0.00	35.00	6.67	13.33

4.2 Analysis

4.2.1 Success Rate

Table 4.1 presents a summary of the results achieved in the experiments described above. The table shows the percentage of *s-bots* that successfully self-assembled (A) and the percentage of *s-bots* that successfully completed the entire task (C). The three columns distinguish between trials with 1 *s-bot*, 2 *s-bots*, and 3 *s-bots*.

The first row shows the total percentage of successful *s-bots*. Subsequent rows show the percentage of *s-bots* that completed stages alone, or as part of a 2 *s-bot swarm-bot* or as as part of a 3 *s-bot swarm-bot*, or that failed. The 'failed' row represents the percentage of *s-bots* that did not succeed in arriving in the target area without toppling over.[3]

The success rate for task completion increases with the number of robots. A single robot always fails. In 2 *s-bot* trials, 65% of *s-bots* complete the task. The 3 *s-bot* trials show a further clear improvement — 86.67% complete the task.

The fourth row (% Successful in 3 *s-bot swarm-bot*) shows that in the 3-*s-bot* trials 80% of *s-bots* successfully self-assemble into a 3 *s-bot swarm-bot*. The same row shows us that 80% of *s-bots* also complete the task in 3 *s-*

[3]For example in the 3 *s-bot* trials 6.67% of *s-bots* completed the task as part of a 2 *s-bot swarm-bot*.

bot swarm-bot. Thus in 3 *s-bot* trials, whenever all the 3 *s-bots* successfully self-assemble into a 3 *s-bot swarm-bot* they always go on to successfully overcome the rough terrain. By contrast, in the 2 *s-bot* trials 100% of the *s-bots* self-assemble into a 2 *s-bot swarm-bot*. Despite this only 65% of the 2 *s-bot swarm-bots* successfully overcome the hill.

As mentioned above, the failure of the 2 *s-bot swarm-bot* always depended on the angle at which the assembled *s-bots* approached the hill. In particular, whenever the *s-bots* approached the hill in parallel or close to parallel, they toppled over backwards (see Figure 4.1). Any linear *swarm-bot* that approaches the hill in parallel is likely to topple. Linear *swarm-bot* formations become less likely with increasing numbers of *s-bots*. In our 3 *s-bot* trials this never happened. Whenever the 3 *s-bots* successfully assembled into a 3 *s-bot swarm-bot*, the *swarm-bot* always overcame the hill.

4.2.2 Timing analysis of 2-sbot trials in Environment B

We have identified five phases of task execution for the 2 *s-bot* trials in Environment B. Figure 4.3 shows how the timing of task execution in the 20 trials is broken down between these phases. Note that these phases represent the state of the system (both *s-bots*) rather than the state of individual *s-bots*.[4] The five phases of task execution are discussed individually below.

- **Independent Phototaxis**. This initial phase is represented in Figure 4.3 by the black bar segment. During this phase all the *s-bots* are performing independent phototaxis to the target light source. All of the *s-bots* are executing `Solo_Phototaxis` behaviour. This phase begins when the trial starts. The phase ends at the moment when the hill is first detected by one of the *s-bots*.

  The duration of this phase is fairly consistent between trials (between 3.6 s and 18.5 s), and is dependent on the random initial configuration of the *s-bots*.

- **Group Hill Detection**. This phase is represented in Figure 4.3 by the white bar segment. During this phase the second *s-bot* becomes aware of the presence of the hill. The phase ends when the second *s-bot* becomes aware of the hill and switches to `Aggregate` behaviour. This phase takes between 0.1 s and 16.6 s.

[4] A phase is a system (group) level property. Each phase represents a particular state of the system. Phases are not inherent to the system - they are states we have identified for analysis purposes. A behaviour is an inherent part of an individual *s-bot* controller. At any moment in time an *s-bot* is unambiguously executing a single behaviour.

- **Aggregation & Assembly Seeding** This phase is represented in Figure 4.3 by the dark grey bar segment. In this phase the *s-bots* are all executing `Aggregate` behaviour, or have temporarily switched to `Assembly_Seed` behaviour. This phase takes between 3.0 s and 58.8 s.

  During this phase the *s-bots* approach each other. This phase ends when the assembly has been successfully seeded. That is, when one of the two *s-bots* has successfully become the seed for the assembly - i.e. switches to `Assembly_Seed` behaviour and remains in `Assembly_Seed` behaviour beyond the initial timeout.

- **Self-Assembly** This phase is represented in Figure 4.3 by the grey segment. In this phase the *s-bots* self-assemble. This tends to be the longest phase, as we would expect, given that self-assembly is the most complex part of the task. The phase ends when all of the *s-bots* have

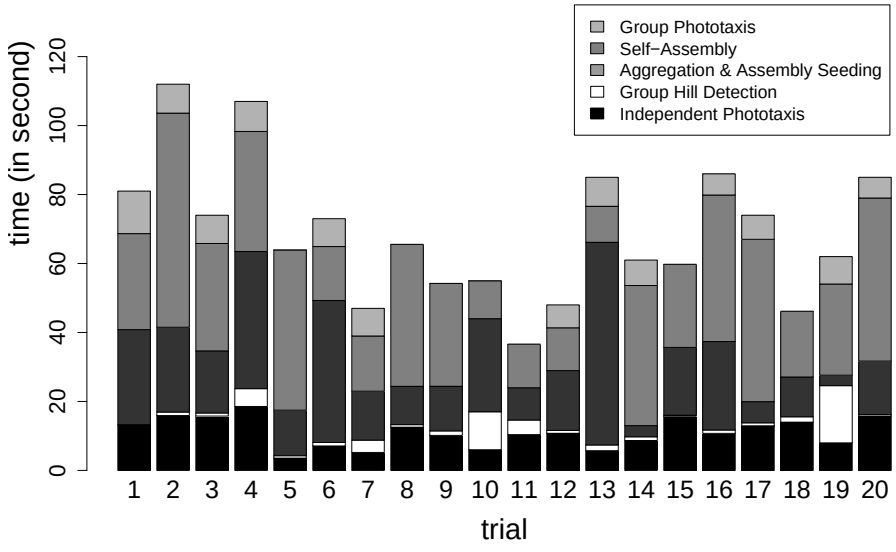


Figure 4.3: Timing analysis of 2 *s-bot* trials in Environment B (for further explanation see text).

switched to `Group_Phototaxis_Behaviour`. This phase takes between 10.4 s and 62.1 s.

- **Group Phototaxis** This phase is represented in Figure 4.3 by the light grey segment. In this phases all of the *s-bots* navigate collectively towards the target light source. In 7 trials the assembled *s-bots* failed to accomplish this phase. In all of the successful trials this phase took less than 13 s. The phase ends when the entire *swarm-bot* is inside the target area.

  In some trials (5,8,9,10,11,15,18) this last phase is not shown in Figure 4.3. These are the trials in which the *s-bots* failed to reach the target area. Note that even in these unsuccessful trials, the two *s-bots* still succeeded in self-assembling, so the representation of the other phases is still meaningful.

4.2.3 Timing analysis of 3-*s-bot* trials in Environment B

We have identified three phases of task execution for the 3 *s-bot* trials in Environment B. Figure 4.4 shows how the timing of task execution in the 20 trials is broken down between these phases. The three phases of task execution are discussed individually below.

Note that in the 3 *s-bot* trials we are not able to identify five distinct phases as we could for the two *s-bot* trials in the previous section. For two *s-bots* the **Group Hill Detection** phase, the **Aggregation and Assembly Seeding** phase and the **Self-Assembly** phase were guaranteed to be time ordered and distinct. For three *s-bots* this is no longer the case. For example, two *s-bots* could in theory self-assemble and perform group phototaxis while the other *s-bot* was still trying to aggregate. Therefore, for the three *s-bot* analysis we have amalgamated these three phases into a single phase - the **Aggregation and Self-Assembly** phase. In the two *s-bot* trials the moment when knowledge of the hill spread through the group and the moment when self-assembly was successfully seeded marked phase transition boundaries. For the three *s-bot* trials as illustrated in Figure 4.4 we instead mark these moments with a 'C' and an 'S' respectively.

- **Independent Phototaxis** This phase is represented by the black bar segment. The phase lasts until one of the *s-bots* has detected the presence of the hill. The duration of this phase is fairly consistent between trials, and is dependent on the random initial configuration of the *s-bots*. It never takes longer than 17 seconds before at least one *s-bot* has detected a hill.

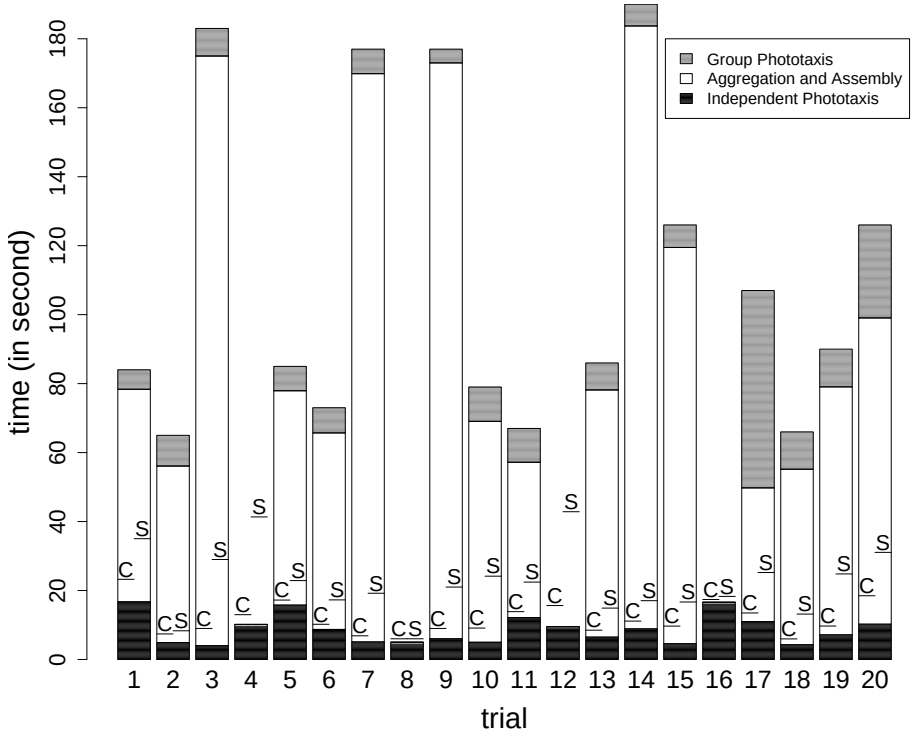


Figure 4.4: Timing analysis of 3 *s-bot* trials in Environment B (for further explanation see text).

- **Aggregation and Self-Assembly** This phase is represented by the white bar segment. In this phase the *s-bots* approach each other and self-assemble. During this phase the *s-bots* execute behaviours `Retreat_To_Flat`, `Aggregate`, `Assembly_Seed`, `Self_Assemble`. The phase ends once all *s-bots* are in `Group_Phototaxis` behaviour. This phase took between 25 s and 166 s. The large percentage of total completion time can be explained by the relatively higher level of complexity of this phase.

- **Group Phototaxis** This phase is represented by the grey bar segment. During this phase the assembled *s-bots* perform collective phototaxis to the target. In all but one trial this phase was accomplished fairly quickly taking between 4s and 20s. Trial 17 was an exception in which the *swarm-bot* got stuck for some time in a particular configuration on the hill.

  The trials in which the two last phases are not shown (4,8,12,16) are the trials in which the *s-bots* did not succeed in assembling into a 3 *s-bot swarm-bot*.

In Figure 4.4 the first time that the whole group becomes aware of the presence of the hill is marked with a 'C'. This gives us an idea of the effectiveness of communication within the group. In the trials where point 'C' is reached quickly, (e.g. trials 5 and 6), a single *s-bot* detects the rough terrain and this knowledge is communicated very quickly to the other *s-bots*, who are sufficiently close to see the blue colour of the *s-bot* that detected the slope. This type of communication is at work in the trial shown in figure 4.2. In other trials (e.g. trial 1 and 20) the *s-bots* are sufficiently far apart that two *s-bots* discover the hill independently.

Note that such local communication happens more often than if the *s-bots* were randomly distributed around the starting area. This is because they each start in `Solo_Phototaxis` behaviour where they are all independently trying to navigate towards the target light source. This increases the probability of being close to each other when the first *s-bot* detects the slope.[5] It is also interesting that the maximum length of time taken for hill awareness to be communicated to the group is greater for the two *s-bot* trials than it is for the three *s-bot* trials. This is because the greater density of *s-bots* in the three *s-bot* trials increases the efficiency of local colour based communication.

[5]Indeed we also observed that aggregation and self-assembly tended to consistently occur near the base of the hill.

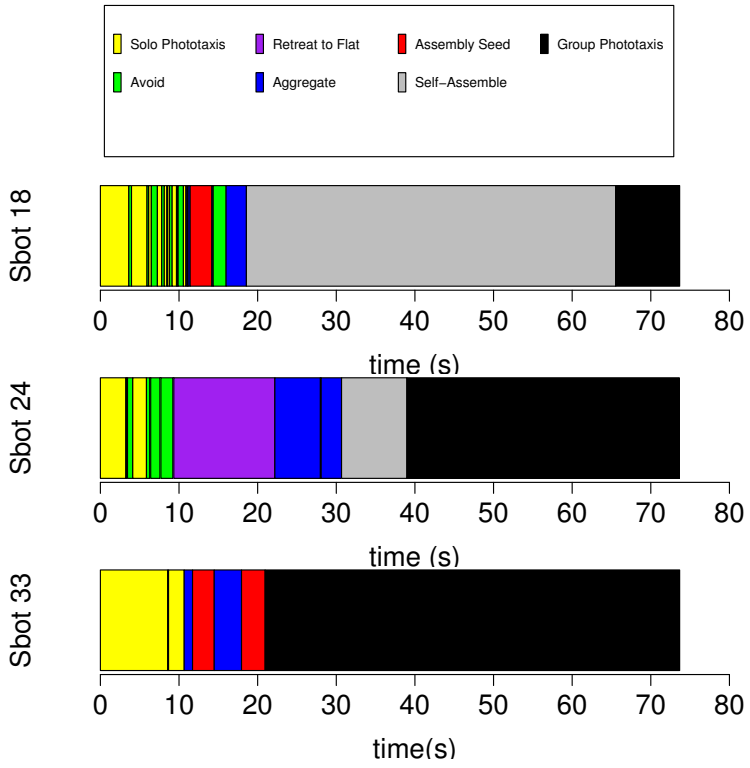


Figure 4.5: Behavioural Ananlysis of 3 *s-bot* trial 16.

The 'S' in figure 4.4 marks when self-assembly is seeded — the first time that an *s-bot* switches into `Assembly_Seed` behaviour and remains in `Assembly_Seed` behaviour beyond the initial timeout. We can see that this usually happens early in the aggregation and self-assembly phase, indicating that of these two activities, self-assembly is the most time consuming.

4.2.4 Behavioural Analysis of a single 3 *s-bot* trial (trial 16)

As with any self-organised system, the behaviour of the system as a whole arises from the interactions between the individual components of the system. In our behaviour based system, these components are not just the *s-bots*, but the individual behaviours of the *s-bots*. To get a more detailed understanding of how this complex behavioural interplay is functioning, we present in Figure 4.5 a breakdown of the behaviours of the individual *s-bots*

over the course of a single 3 *s-bot* trial in Environment B. In contrast to sections 4.2.2 and 4.2.3 above where we considered phases of the system as a whole, we now analyse the internal behavioural states of the individual *s-bots* and their interplay.

All the three *s-bots* start in `Solo_Phototaxis` behaviour. *S-bot* 18 and *s-bot* 24 repeatedly switch in and out of `Avoid_Obstacle` behaviour as they head towards the target. After 9.3 s *s-bot* 24 detects the slope and switches into `Retreat_to_Flat` behaviour. Within 2 seconds both *s-bots* 18 and 33 (10.9 s and 10.66 s respectively) have switched into `Aggregate` behaviour. This is achieved through colour based local communication (both *s-bots* notice the blue colour of *s-bot* 24).

At 11.4 s *s-bot* 18 probabilistically switches into `Assembly_Seed` behaviour. At 11.7 s *s-bot* 33 independently also probabilistically switches into `Assembly_Seed` behaviour. Both are sufficiently close that they detect each other and both switch back to `Aggregate` behaviour.

At 17.95 s *s-bot* 33 again probabilistically switches into `Assembly_Seed` behaviour. This time neither of the other two *s-bots* switch into `Assembly_Seed` behaviour within the 3 s timeout, so *s-bot* 33 switches to `Group_Phototaxis` behaviour at 20.95 s. At 18.57 s *s-bot* 18 detects the presence of the assembly seed (notices a red object) and switches into `Self_Assembly` behaviour.

S-bot 24 meanwhile finishes retreating away from the hill at 22.2 s. It then switches from `Retreat_to_Flat` behaviour into `Aggregate` behaviour. It then takes a further 8 s to find and notice the assembly seed (*s-bot* 33) at 30.71 s. At 39.0 s *s-bot* 24 has successfully assembled to *s-bot* 33.

S-bot 33 and *s-bot* 24 are both now waiting for *s-bot* 18 to assemble before they set off for the target. They don't move as long as they continue to detect a blue object (which is *s-bot* 18 still in the process of self-assembling).

S-bot 18 finally succeeds in self-assembling at 65.5 s. At this point all three successfully perform group taxis to the target. The task is completed at 73.6 s.

Chapter 5

Ongoing Research

5.1 Introduction

The two *s-bot* trials in our experiments of the last chapter failed when the two *s-bot swarm-bot* approached the hill with an orientation parallel to that of the hill (see Figure 4.1). To improve the success rate of the 2 *s-bot* trials, the assembled *swarm-bot* would need some form of control over its orientation with respect to the environment. Observation of the three *s-bot* experiments as well as initial work with larger numbers of *s-bots* also suggest that the orientation of the *swarm-bot* is an important factor that affects the ability of an assembled *swarm-bot* to traverse rough terrain.

We present some preliminary work on controlling the orientation of an assembled *swarm-bot*. We restrict our attention for the time being to (pre-assembled) *swarm-bots* connected in a linear configuration. Our goal is to produce a swarm control mechanism that will prevent such a linear *swarm-bot* from approaching an obstacle with an orientation parallel to that of the obstacle.

For experimental simplicity we use troughs (long rectangular holes) rather than hills as our rough terrain. Note that troughs create the same "angle of approach" problems. If a linear *swarm-bot* approaches a trough with an orientation parallel to that of the trough, the *swarm-bot* will fall into the trough (see Figure 5.5). If, however, a *swarm-bot* approaches a trough with an orientation perpendicular to that of the trough, the *swarm-bot* can navigate over the trough (see Figure 5.4d).

We make one further important simplifying assumption: that the trough is perpendicular to the direction of the target light source (at least from the *swarm-bot*'s starting position). This allows us to perform adaptive swarm

rotation on the basis of light source detection rather than by sensing the rough terrain directly.

The task we investigate requires a pre-assembled linear *swarm-bot* to traverse two troughs, one after the other. The two troughs are perpendicular to each other. Thus whatever its random initial orientation, the *swarm-bot* must be able to adaptively control its orientation (adaptively rotate) in order to navigate over both troughs.

It is true that the assumptions we have made seriously limit the practical use of the controller we present in this chapter. However, this chapter represents an initial investigation into swarm control mechanisms — we intend to expand this work to be more generally applicable in the future.

In the following section we describe the experimental setup used to test adaptive control of swarm rotation. In the subsequent two sections, we go on to describe the controller and present some preliminary results. Finally we discuss the limits of the controller we developed and how these limitations might be overcome to make our control mechanism more generic.

5.2 Experimental Setup

5.2.1 The Environment

Our environment consists of two joined, overlapping rectangular areas, each of dimensions 120 cm x 300 cm. Each rectangular area has its own target light source at the end furthest from the starting area. Each rectangular area contains a trough. The two troughs are both of dimension 11 cm x 120 cm (both run the whole width of their respective rectangular areas). The troughs are impassable by a two *s-bot swarm-bot*.

A diagram of the environment can be seen in Figure 5.1. A photograph of the environment can be seen in Figure 5.2.

5.2.2 The Task

The *s-bots* start pre-assembled in a linear *swarm-bot* configuration. The linear *swarm-bot* starts in a random orientation in the starting zone. At the start of every trial, Light Source 1 is illuminated and Light Source 2 is switched off. To complete the task the *swarm-bot* is required to navigate first to Target Area 1, and then to Target Area 2 (see Figure 5.1).

The linear *swarm-bot* navigates to Target Area 1 by performing phototaxis towards Light Source 1. To arrive at Light Source 1 the *swarm-bot* must

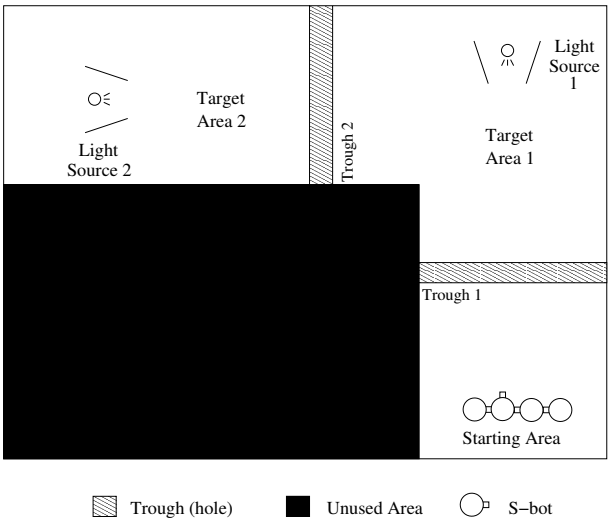


Figure 5.1: Diagram of the environment (view from above). To complete the task the pre-assembled linear *Swarm-bot* must cross both troughs and arrive in Target Area 2.

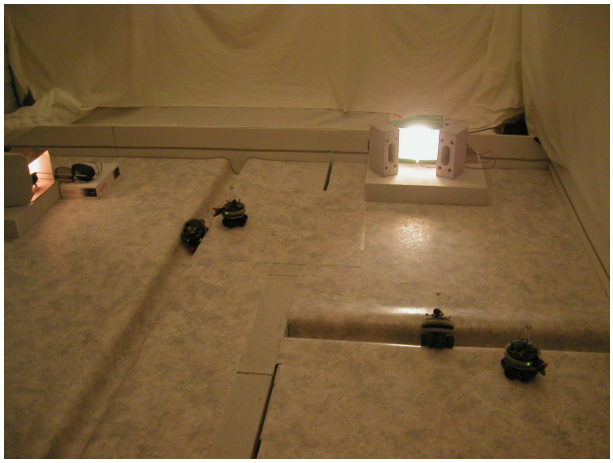


Figure 5.2: Photograph of the environment. In this photograph both light sources are illuminated. During experiments only one light source at a time is ever illuminated.

also rotate as it performs phototaxis to ensure that it approaches Trough 1 with an orientation perpendicular to that of the trough.

If the *swarm-bot* successfully reaches Target Area 1, the experimenter switches off Light Source 1 and illuminates Light Source 2. The *swarm-bot* continues to perform phototaxis to Light Source 2. Again, adaptive swarm rotation is required, this time in order to overcome Trough 2.

5.3 The Controller

5.4 High level strategy

The task as specified in the previous section requires adaptive swarm rotation. In order to produce this in a distributed controller, it is necessary that the individual *s-bots* have some way of determining the overall orientation of the *swarm-bot*.

Based on the direction of the target direction and the orientation of the *swarm-bot* each *s-bot* must independently determine a direction and speed of movement. The combined result of the individual *s-bot* movements must have a two-fold effect on the *swarm-bot*. The *swarm-bot* should move towards the target light source. At the same time the *swarm-bot* should also adaptively rotate until its orientation is parallel to the direction of motion.

5.5 Calculating the *swarm-bot*'s orientation

Calculation of the *swarm-bot*'s orientation is achieved by means of colour sensing based communication. Each *s-bot* illuminates its red LEDs for the duration of the controller execution. By detecting the density and direction of red objects in its vicinity, an *s-bot* can both determine the orientation of the *swarm-bot* and deduce its own position in the linear formation. (At least it is possible to determine whether it is at the front of the *swarm-bot*, at the rear of the *swarm-bot*, or somewhere in the middle). For noise reduction purposes, only 8 possible orientation are considered. This works in a similar fashion to the camera noise filtration algorithm (see section 3.1.2) — the horizontal plane is divided into 8 segments, and all directions are mapped to one of these segments.

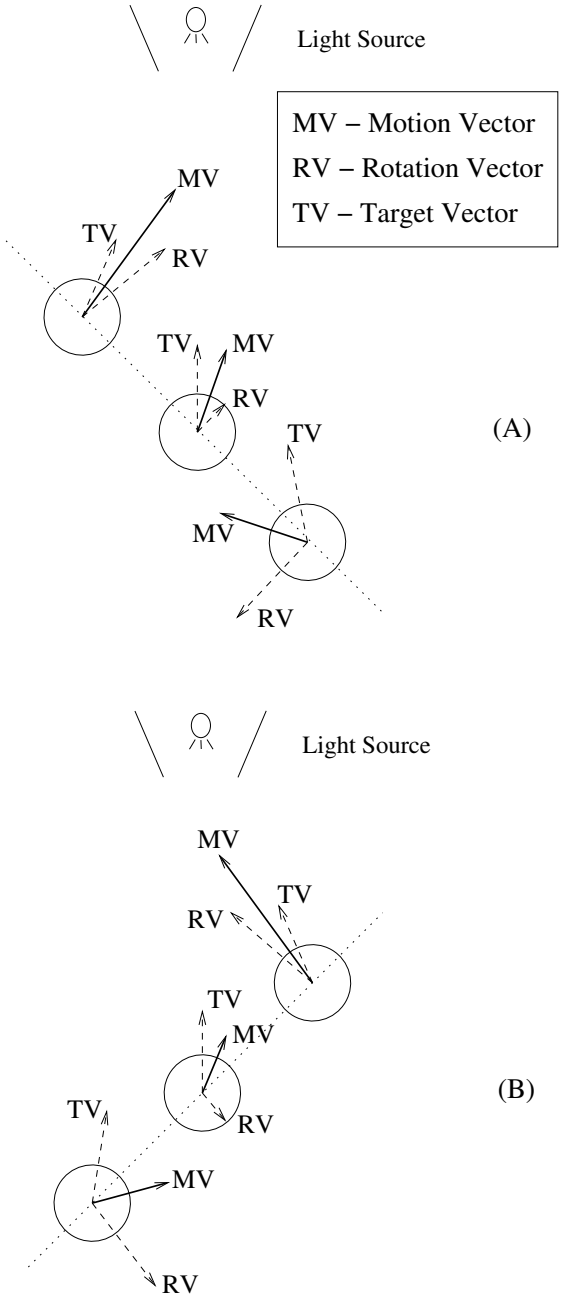


Figure 5.3: *Motion vector* calculation in a 3 *s-bot* linear *swarm-bot*. Two scenarios are shown (A and B). In both cases, the *rotation vector* direction and magnitude are determined by two key factors: (i) the position of the *s-bot* within the *swarm-bot* with respect to the direction of the light source (front, middle, rear) (ii) the angle between the *swarm-bot*'s orientation and the *target vector* direction.

5.6 Determining *s-bot* direction and speed

Each *s-bot* calculates two vectors. The first is the *target vector*. The direction of this vector is the direction the *s-bot* would move in if it were only performing phototaxis. The second is the *rotation vector*. The direction of this vector is the direction the *s-bot* would move in if it only needed to adaptively rotate the *swarm-bot*. These two vectors are then combined to give the *swarm-bot* its *motion vector*. The *s-bot* direction and speed of motion are set according to the direction and magnitude of the *motion vector*.[1] The calculation of the *motion vector* is illustrated in Figure 5.3.

The *target vector* direction is calculated using the same target direction noise filtering algorithm from our previous controller - see section 3.1.2.

If the *swarm-bot* is correctly oriented (orientation is parallel to *target vector* direction), then the *rotation vector* direction is set parallel to the *target vector* direction.

If the *swarm-bot* is incorrectly oriented, the *s-bot* first determines its position within the *swarm-bot* with respect to the direction of the target light source. If the *s-bot* can detect *s-bots* on both sides of it then it is in the middle of the *swarm-bot*. Otherwise, if the direction of the nearest *s-bot* (red object) is within 45 degrees of the direction of the target light source, the *s-bot* is at the front of the *swarm-bot*. Otherwise, the *s-bot* is at the rear of the *swarm-bot*.

Based on its position within the *swarm-bot* the *s-bot* chooses a *rotation vector* perpendicular to the orientation of the *swarm-bot*. There are two possible directions for this perpendicular rotation vector. One of these two directions would move the *s-bot* closer to the target. The other would move the *s-bot* away from the target. If the *s-bot* is at the front of the *swarm-bot* or in the middle of the *swarm-bot*, the *s-bot* chooses the rotation vector that moves it closer to the target. If the *s-bot* is at the rear of the *swarm-bot* the *s-bot* chooses the rotation vector that moves it away from the target. This choice of *rotation vector* direction based on the relationship of the *swarm-bot*'s orientation to the *target vector* can be seen in Figure 5.3.

The *target vector* is always given unit magnitude. The *rotation vector* has unit magnitude if the *s-bot* is at either the front or the rear of the *swarm-bot*. Otherwise the *rotation vector* has magnitude 0.5. This encourages rotation where it is most needed - at the ends of the *swarm-bot*. The maximum track speed for the *s-bot* is set based on the *motion vector*

[1] This is done using the same combination of turret/chassis rotation and track speeds as used for collective motion in `Group_Phototaxis` behaviour (see section 3.9)

magnitude according to equation 5.1.

$$maximumTrackSpeed = \frac{MAX\_SPEED * (motion\ vector\ magnitude)}{2} \tag{5.1}$$

5.7 Results

Preliminary experimentation with this new controller gave a success rate of roughly 40% in trials with 4 *s-bots*. When, however, adaptive swarm rotation was disabled (by setting the magnitude of the *rotation vector* to 0), the success rate dropped to 0%.

A sequence of photographs of a typical successful trial can be seen in Figure 5.4. Photographs of a failed trial with the non adaptive controller can be seen in Figure 5.5.

The relatively low success rate of 40% is partly attributable to the high difficulty level of the troughs we used. The troughs were wider than a single *s-bot*. This meant that the leading *s-bot* would actually partially descend into the trough. When the *swarm-bot* was perfectly aligned, the leading *s-bot*'s tracks would grip the far wall of the trough and the track rotation would help lift the *s-bot* over the lip of the trough. When the *swarm-bot* was even slightly misaligned, this lifting effect would not occur, the leading *s-bot* would get stuck in the trough and the *swarm-bot* would thus fail to navigate the trough.

The low success rate was also partly attributable to continuing instabilities in the system. Ideally we would like the *swarm-bot* to rotate until it is aligned correctly, then cease all rotational activity and just head towards the light source. What happened in reality was that the leading *s-bot* would often over-rotate, and then have to adjust by rotating back in the other direction. When the other *s-bots* tried to follow suite, this would result in a snake-like motion towards the light source.

As discussed above, due to the difficult nature of the trough used, even when the *swarm-bot* was only slightly misaligned it would fail to traverse the trough. We therefore believe that refinement of the controller to avoid the over rotation would result in much higher success rates.

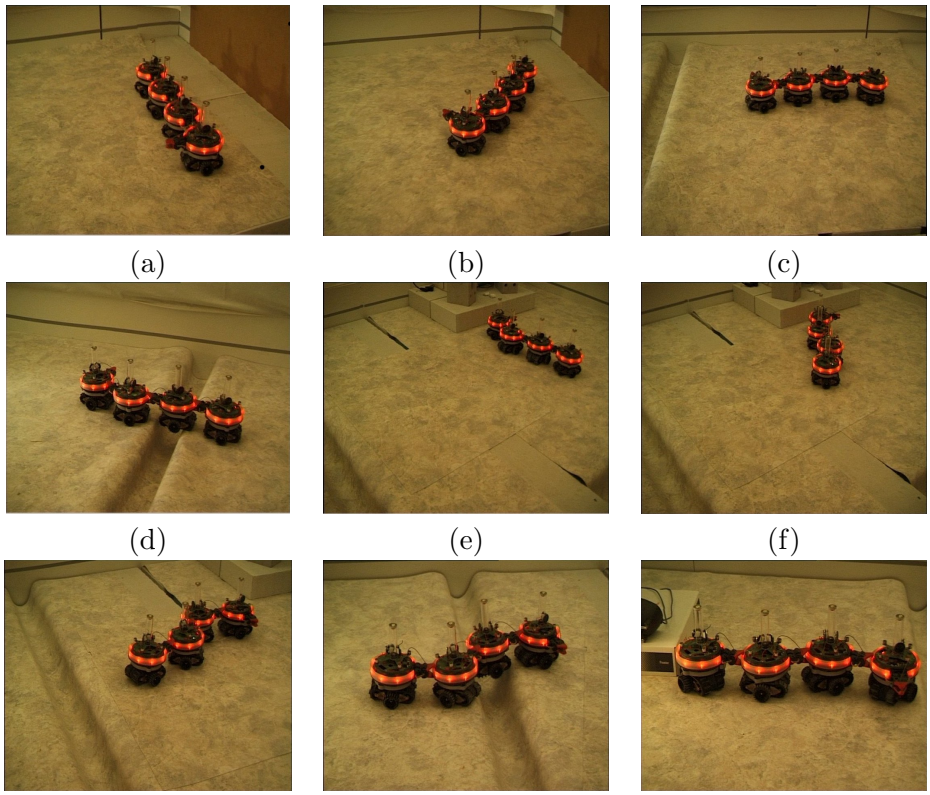


Figure 5.4: Successful task completion. The linear *swarm-bot* adjusts its rotation twice - once for each trough. In photograph (e) the *s-bot* has arrived in Target Zone 1. At this point the experimenter switches off Light Source 1 and switches on Light Source 2. The *swarm-bot* carries on to successfully complete the task by arriving in Target Zone 2.

(a) (b) (c)

Figure 5.5: *Swarm-bot* using old controller. *S-bots* use greedy algorithm and head straight towards the target without considering swarm alignment. The *swarm-bot* fails to overcome the trough.

5.8 Further Development

5.8.1 Limitations of the current controller

Our new controller makes two key assumptions about the specific nature of the task and environment we have set up. Because of these assumptions, our new controller cannot be considered generic.

The first assumption is that the *swarm-bot* has been pre-configured into a linear formation. Each *s-bot* deduces both the orientation of the *swarm-bot* and its own position within the *swarm-bot* by observing the density of red objects on either side of it. If the assembled structure were not linear, such deductions would be invalid.

Secondly, the controller is making an assumption about the relationship between the trough and the direction of the target light source. In particular, the controller assumes that the orientation of the trough is perpendicular to the direction of the target light source. If this were not the case, aligning the *swarm-bot* with respect to the light source might be futile.[2]

5.8.2 Future controller development

The limitations discussed in the previous section immediately suggest two areas of further study.

[2]This assumption may generalise to some extent. Our swarm controller tries to orient the *swarm-bot* to be parallel to the direction of motion. Even without the assumption holding true, this strategy would still make it almost impossible for the *s-bot* to approach even a randomly oriented obstacle with an orientation parallel to that of the obstacle. (Of course, the *swarm-bot* is unlikely to approach such an obstacle with an orientation perpendicular to that of the obstacle.)

The linear configuration assumption could be made valid by somehow ensuring that the *s-bots* always self-assembled into a linear structure. This could be achieved by modifying the `Self_Assembly` behaviour module so that assembling *s-bots* selectively lit up relevant portions of their LED ring to encourage linear connections.

The assumption about the relationship between the light source direction and the trough orientation is more fundamental. The *swarm-bot* needs some way of adaptively rotating on the basis of the nature of the rough terrain. This could possibly be done by sensing the rough terrain (be it hill or trough), and generating the *rotation vector* on that basis. This is by no means a trivial task, however — plenty of practical hurdles will have to be overcome, including speed of sensing (*s-bots* might only detect rough terrain when it is already too late) and differences in sensed information between *s-bots* (some *s-bots* might realise the need to rotate, others might not).

Chapter 6

Conclusions

6.1 Overview of Results

In this thesis we have demonstrated that it is possible for a group of robots to choose to self-assemble in response to the demands of their task and environment. In our study a group of robots faced with a simple hill overcome it independently. When the same robots are faced with a hill too difficult for a single robot to pass they self-assemble and overcome the hill together.

We also demonstrated that the benefits of self-assembly increase with group size. In the experiments with the difficult hill no unassembled robots ever succeeded in reaching the target area. With group sizes of 2 s-bots, 65% of the *s-bots* successfully carried out the task. A further clear improvement could be seen for the 3 *s-bot* groups — 87% of the *s-bots* successfully completed the task.

6.2 Future Research Directions

6.2.1 The Evolutionary Perspective

In a previous work conducted in a simplified simulation environment, Trianni et al. [65] focused on evolving a single neural network controller integrating the different aspects of functional self-assembly. Due to the complexities involved in implementing functional self-assembly for the first time with physical robots, we used a building block approach — we implemented a collection of simple basic behaviours corresponding to the different phases of functional self-assembly seen from the individual perspective. We developed a behaviour based controller by combining these building blocks.

We believe that it would be interesting to apply the evolutionary approach to the real robots. This might yield solutions that exploit hidden properties of the robotic hardware or which make better use of the complex group dynamics of the task [54].

6.2.2 More sophisticated functional self assembly

If functionally self-assembling swarm systems are ever to become a practical reality for complex real world problems they will have to demonstrate the following autonomous capabilities:

- Self-assembly in response to environmental conditions.
- Control over the self-assembly process. This would involve the creation of different assembled patterns to meet needs of the particular problem being solved. For example, rough terrain navigation problems and object transport problems would require different assembled formations.
- Control over the assembled swarm. The assembled robots must display swarm behaviour appropriate to the task. The more complex the task, the more complex this swarm behaviour will have to be.
- Disassembly in response to the environmental conditions. Robots might, for example, need to assemble to cross some rough terrain, then disassemble in order to go through a narrow passageway.

Over the course of this thesis we have shown that the first item above is possible (albeit in a simple environment).

A sensible starting point for future research would, therefore, be to try and make similar proof of concept demonstrations for items 2 — 4 above. In the previous chapter we presented some initial work on the third item above — control of the assembled swarm.

Once each capability has been addressed individually, the next step would be to try and demonstrate all of these capabilities together in a single system.

6.3 Significance of this work

Functional self-assembly is a key adaptive response mechanism in many social insect species. Several species of social insect utilise self-assembly

to solve problems collectively that are too large or complex for a single insect [2].

Swarm robotics is a field that bases itself on principles learnt from such natural systems. It is therefore surprising, given the ubiquity of functional self-assembly as an adaptive response mechanism in the animal kingdom, how little research has been done on the subject in swarm robotics.

We believe that our results represent a significant first step on the road to utilising this important mechanism in real robotic systems. Now the same mechanism must be applied to more complex tasks.

Bibliography

[1] P.E. Agre and D. Chapman. Pengi: An implementation of a theory of activity. In *Proc. of AAAI-87*, pages 268–272, Seattle, WA, 1987. Morgan Kaufmann.

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

[3] R.C. Arkin. Towards the unification of navigational planning and reactive control. In *AAAI Spring Symposium on Robot Navigation*, pages 1–5, Stanford University, CA, 1989.

[4] R.C. Arkin. *Behavior-Based Robotics*. MIT Press, Cambridge, MA, 1998.

[5] Ronald C. Arkin. Cooperation without communication: Multiagent schema based robot navigation. *Journal of Robotic Systems*, 9(3):351–364, 1992.

[6] W.R. Ashby. *Design for a Brain*. Wiley & Sons, New York, 1952.

[7] E Bahceci, O. Soysal, and E. Sahin. A review: Pattern formation and adaptation in multi-robot systems. Technical Report CMU-RI-TR-03-43, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 2003.

[8] T Balch. Learning roles: Behavioral diversity in robot teams. Technical Report GIT-CC-97-12, Georgia Institute of Technology, Atlanta, GA, 1997.

[9] G. Beni. From swarm intelligence to swarm robotics. In E. Sahin and W.M. Spears, editors, *SAB*, volume 3342 of *Lecture Notes in Computer Science*, pages 1–9, Berlin, Germany, 2004. Springer-Verlag.

[10] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York, 1999.

[11] R.A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics & Automation*, 2(1):14–23, 1986.

[12] R.A. Brooks. Elephants don't play chess. *Robotics and Autonomous Systems*, 6(1-2):3–15, 1990.

[13] R.A. Brooks. Intelligence without reason. In J. Myopoulos and R. Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 569–595, San Mateo, CA, 1991. Morgan Kaufmann.

[14] S. Camazine, J.-L. Deneubourg, N. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau. *Self-Organization in Biological Systems*. Princeton University Press, Princeton, NJ, 2001.

[15] S. Camazine and J. Sneyd. A model of collective nectar source selection by honey bees: self-organization through simple individual rules. *Journal of Theoretical Biology*, 149:547–571, 1991.

[16] S. Camazine, J. Sneyd, M.J. Jenkins, and J.D. Murray. A mathematical model of self-organized pattern formation the combs of honeybees colonies. *Journal of Theoretical Biology*, 1:295–311, 1990.

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

[18] L. Chaimowicz, V. Kumar, and M. Campos. A paradigm for dynamic coordination of multiple robots. *Autonomous Robots*, 17(1):7–21, 2004.

[19] R. Chatila and J. Laumond. Position referencing and consistent world modeling for mobile robots. In *IEEE International Conference on Robotics and Automation*, Piscataway, NJ, 1985. IEEE Publications.

[20] J. Connell. *Minimalist Mobile Robotics: A Colony-style Architecture for a Mobile Robot*. Academic Press Professional, San Diego, CA, 1990.

[21] J. Connell. A hybrid architecture applied to robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2719–2724, Piscataway, NJ, 1992. IEEE Publications.

[22] J.-L. Deneubourg, S. Aron, S. Goss, and J.M. Pasteels. The self-organizing exploratory patterns of the argentine ant. *Journal of Insect Behaviour*, 3:159–168, 1990.

[23] J.-L. Deneubourg and S. Goss. Collective patterns and decision making. *Ethology Ecology Evolution*, pages 295–311, 1989.

[24] J.-L. Deneubourg, J. C. Gregoire, and E. Le Fort. Kinetics of the larval gregarious behaviour in the bark beetle *Dendroctonus micans*. *Journal of Insect Behavior*, 3:169–182, 1990.

[25] C. Detrain. Field study on foraging by the polymorphic ant species pheidole pallidula. *Insectes Sociaux*, 37(4):315–332, 1990.

[26] C. Detrain and J.-L. Deneubourg. Scavenging by *Pheidole crassinoda*: a key for understanding decision-making systems in ants. *Animal Behaviour*, 53:537–547, 1997.

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

[28] E.J.P. Earon, T.D. Barfoot, and G.M.T. D'Eleuterio. Development of a multiagent robotic system with application to space exploration. In *Proceedings of 2001 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, volume 2, pages 1267–1272, Piscataway, NJ, 2001. IEEE Publications.

[29] T. Estier, Y. Crausaz, B. Merminod, M. Lauria, R. Piguet, and R. Siegwart. An innovative space rover with extended climbing abilities. In *Proceedings of the ASCE Conference on Robotics for Challenging Environments*, pages 333–339, Dallas, TX, 2000. ASCE Publications.

[30] Tara A. Estlin, Alexander Gray, Tobias Mann, Gregg Rabideau, Rebecca Castano, Steve Chien, and Eric Mjolsness. An integrated system for multi-rover scientific exploration. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 613–620, Menlo Park, CA, 1999. AAAI Press.

[31] T.D. Fitzgerald. *The tent caterpillars.* Cornell University Press, Ithaca, NY, 1995.

[32] T. Fukuda, S. Nakagawa, Y. Kawauchi, and M. Buss. Structure decision method for self organising robots based on cell structures-cebot. In *Proc. of the 1989 IEEE International Conference on Robotics and Automation (Vol. 2)*, pages 695–700, Piscataway, NJ, 1989. IEEE Publications.

[33] B.P. Gerkey and M.J Matarić. Multi-robot task allocation: Analyzing the complexity and optimality of key architectures. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Piscataway, NJ, 2003. IEEE Publications.

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

[35] G. Giralt, R. Chatila, and M. Vaisset. An integrated navigation and motion control system for autonomous multisensory mobile robots. In S. S. Iyengar and A. Elfes, editors, *Autonomous Mobile Robots: Control, Planning, and Architecture (Vol. 2)*, pages 254–277. IEEE Computer Society Press, 1991.

[36] D. Goldberg and M. Matarić. Design and evaluation of robust behavior-based controllers. In T. Balch and L.E. Parker, editors, *Robot Teams: From Diversity to Polymorphism*. A K Peters, Wellesley, MA, 2002.

[37] S. Goss and J.-L. Deneubourg. Harvesting by a group of robots. In *Toward a Practice of Autonomous Systems, Proceedings of the First European Conference on Artificial Life*, pages 195–204, Cambridge, MA, 1991. MIT Press.

[38] P.-P. Grassé. La reconstruction du nid et les coordination inter-individuelles chez bellicositermes natalensis et cubitermes sp. la thérie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs. *Insectes Sociaux*, 6:41–80, 1959.

[39] R. Groß, M. Bonani, F. Mondada, and M. Dorigo. Autonomous self-assembly in mobile robotics. Technical Report IRIDIA/2005-2, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2005.

[40] R. Groß and M. Dorigo. Group transport of an object to a target that only some group members may sense. In *Parallel Problem Solving from Nature – 8th International Conference (PPSN VIII)*, volume 3242 of

Lecture Notes in Computer Science, pages 852–861, Berlin, Germany, 2004. Springer-Verlag.

[41] C.V. Jones and M.J. Mataric. Automatic synthesis of communication-based coordinated multi-robot systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 381–387, Piscataway, NJ, 2004. IEEE Publications.

[42] Y. Kuniyoshi, N. Kita, S. Rougeaux, S. Sakane, M. Ishii, and M. Kakikura. Cooperation by observation - the framework and basic task patterns. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 767–774, 1994.

[43] J.E. Laird and P. Rosenbloom. The evolution of the Soar cognitive architecture. In David M. Steier and Tom M. Mitchell, editors, *Mind Matters: A Tribute to Allen Newell*, pages 1–50. Lawrence Erlbaum Associates, Inc., 1996.

[44] P. Maes. The dynamics of action selection. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 991–997, San Francisco, CA, 1989. Morgan Kaufmann.

[45] A. Martinoli, K. Easton, and W. Agassounon. Modeling swarm robotic systems: A case study in collaborative distributed manipulation. *International Journal of Robotics Research*, 23(4):415–436, 2004.

[46] A. Martinoli and F. Mondada. Collective and cooperative group behaviours: Biologically inspired experiments in robotics. In *Proceedings of the Fourth International Symposium on Experimental Robotics*, pages 3–10, Berlin, Germany, 1995. Springer-Verlag.

[47] M.J. Mataric. Behavior-based control: Examples from navigation, learning, and group behavior. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2-3):323–336, 1997.

[48] F. Mondada, L. M. Gambardella, D. Floreano, S. Nolfi, J.-L. Deneubourg, and M. Dorigo. SWARM-BOTS: Physical interactions in collective robotics. *Robotics & Automation Magazine*, 12(2):21–28, 2005.

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

[50] H. Moravec and D.W. Cho. A bayesian method for certainty grids. In *AAAI 1989 Spring Symposium Series, Symposium on Mobile Robots*, Stanford University, CA, 1989.

[51] S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji. M-TRAN: Self-reconfigurable modular robotic system. *IEEE/ASME Transactions on Mechatronics*, 7(4):431–441, 2002.

[52] W. Mnch and W. Engels. Vorkommen der moor-knotenameise myrmica gallienii im riedgrtel des federsees (hymenoptera: Myrmicidae). *Entomologia Generalis*, 19:15–20, 1994.

[53] N.J. Nilsson. Shakey the robot. Technical Report 323, AI Center, SRI International, Menlo Park, CA, 1984.

[54] S. Nolfi and D. Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines.* MIT Press, Cambridge, MA, 2000.

[55] M. Nørgaard, O. Ravn, N. Poulsen, and L. Hansen. *Neural Networks for Modelling and Control of Dynamic Systems.* Advanced Textbooks in Control and Signal Processing. Springer-Verlag, Berlin, Germany, 2000.

[56] L. Parker. Alliance: An architecture for fault-tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, 1998.

[57] D.W. Payton. Internalized plans: a representation for action resources. *Robotics and Autonomous Systems*, 6:89–103, 1990.

[58] D.W. Payton, M. Daily, R. Estowski, M. Howard, and C. Lee. Pheromone robotic. *Autonomous Robots*, 11(3):319–324, 2001.

[59] D.W. Payton, D. Keirsey, D. M. Kimble, J. Krozel, and J.K. Rosenblatt. Do whatever works: A robust approach to fault-tolerant autonomous control. *Applied Intelligence*, 2:225–250, 1992.

[60] F. Saffre, R. Furey, B. Kraft, and J.L. Deneubourg. Collective decision-making in social spiders: Dragline-mediated amplification process acts as a recruiting mechanism. *Journal of Theoretical Biology*, 198:507–517, 1999.

[61] T.D. Seeley. *The Wisdom of the Hive*, pages 277–290. Harvard University Press, Cambridge, MA, 1995.

[62] L. Steels. The artificial life roots of artificial intelligence. *Artificial Life*, 1:75–110, 1994.

[63] H. Stone. Mars Pathfinder Microrover - A Small, Low-Cost, Low-Power Spacecraft. In *Proceedings of the 1996 AIAA Forum on Advanced Developments in Space Robotics*, Reston, VA, 1996. AIAA Publications.

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

[65] V. Trianni, E. Tuci, and M. Dorigo. Evolving functional self-assembling in a swarm of autonomous robots. In *From Animals to Animats $VIII$. Proceedings of the 8^{th} International Conference on Simulation of Adaptive Behavior*, pages 405–414, Cambridge, MA, 2004. MIT Press.

[66] N. Vandapel, S. Moorehead, W. Whittaker, R. Chatila, and R. Murrieta-Cid. Preliminary results on the use of stereo, color cameras and laser sensors in Antarctica. In *Proceedings of the International Symposium on Experimental Robotics*, Berlin, Germany, 1999. Springer-Verlag.

[67] M. Yim, D.G. Duff, and K.D. Roufas. PolyBot: a Modular Recon gurable Robot. In *Proceedings of the 2000 IEEE/RAS International Conference on Robotics and Automation*, volume 1, pages 514–520, Cambridge, MA, 2000. IEEE Publications.

[68] M. Yim, K. Roufas, D. Duff, Y. Zhang, C. Eldershaw, and S. Homans. Modular reconfigurable robots in space applications. *Autonomous Robots*, 14(2-3):225–237, 2003.