Morphologically Responsive Self-Assembling Robots

by

Rehan O'Grady

Université Libre de Bruxelles
Faculté des Sciences Appliquées, CoDE, IRIDIA
rogrady@ulb.ac.be

Supervised by

Marco Dorigo, Ph.D.

Directeur de Recherches du FNRS
Université Libre de Bruxelles
Faculté des Sciences Appliquées, CoDE, IRIDIA
mdorigo@ulb.ac.be

June, 2010

A thesis submitted in partial fulfilment of the requirements of the *Université Libre de Bruxelles*, *Faculté des Sciences Appliquées* for the doctoral degree (PhD) in

Sciences de l'Ingénieur

Abstract

We investigate the use of self-assembly in a robotic system as a means of responding to different environmental contingencies. Self-assembly is the mechanism through which agents in a multi-robot system autonomously form connections with one another to create larger composite robotic entities. Initially, we consider a simple response mechanism that uses stochastic self-assembly without any explicit control over the resulting morphology — the robots self-assemble into a larger, randomly shaped composite entity if the task they encounter is beyond the physical capabilities of a single robot. We present distributed behavioural control that enables a group of robots to make this collective decision about when and if to self-assemble in the context of a hill crossing task. In a series of real-world experiments, we analyse the effect of different distributed timing and decision strategies on system performance. Outside of a task execution context, we present fully decentralised behavioural control capable of creating periodically repeating global morphologies. We then show how arbitrary morphologies can be generated by abstracting our behavioural control into a morphology control language and adding symbolic communication between connected agents. Finally, we integrate our earlier distributed response mechanism into the morphology control language. We run simulated and real-world experiments to demonstrate a self-assembling robotic system that can respond to varying environmental contingencies by forming different appropriate morphologies.

Acknowledgments

First and foremost, I would like to thank my supervisor Marco Dorigo. I am very grateful for the opportunity he gave me to work at IRIDIA. I feel that my time in this laboratory has allowed me to grow as a scientist and as a person. Marco leads by example, and from him I have learnt much more than robotics. I would also like to thank Roderich Gross and Anders Lyhne Christensen for their close collaboration. More recently, working with Carlo Pinciroli, Nithin Mathews, Mauro Birattari and Arne Brutschy has also been a pleasure. In general, I am grateful to all of my colleagues at Iridia for always being so helpful and for making IRIDIA such a pleasant working environment. Particular thanks go to Hugues Bersini, who helped to create and nurture this environment. I would also like to acknowledge the essential role played by Francesco Mondada's group at the EPFL, who designed and built the robotic platform used in this thesis. Finally, I would like to thank Elena for her support and for not getting too upset (most of the time) about the many late nights spent in the intimate company of robots.

Contents

Chapter 1

Introduction

Self-assembling robotic systems are multi-robot systems in which the robots can autonomously form physical connections with one another. The long term goal of robotic self-assembly research is to create systems that can adapt to new or changing environments by creating teams of appropriately shaped composite robotic entities. Based on the environmental context, the size and shape of each self-assembled entity could be chosen to maximise the efficiency of the overall system. Without self-assembly, autonomous robotic systems are by definition limited by morphological constraints of size and/or shape.

In this thesis, we investigate behavioural mechanisms that autonomously control the timing and nature of the self-assembly process. To date, self-assembly research has focused on the problem of autonomously creating physical connections between components. However, without higher level behavioural control over the morphologies of the resulting composite entities, many of the potential adaptive benefits of self-assembly are lost. And even if a system can autonomously form physical connections, its overall autonomy remains limited if it does not have control over when to deploy its self-assembly capability, and how to allocate its resources. Together, decisional autonomy and morphology control make a self-assembling system *morphologically responsive*. A morphologically responsive self-assembling system can respond to its environment by changing its morphological characteristics.

Over the course of this thesis, we develop and analyse a morphologically responsive self-assembling system. Because of the limited sensory apparatus of the robotic agents we use, our system responds either to single variable parameters of the environment (e.g., steepness or orientation of a hill), or to cues that are explicitly placed into the environment to make identification of its salient features easier for the robots (e.g., shiny ground surface). The morphologies and behaviours that our system displays in response to different environments are a priori mapped to specific environmental cues or parameter values. In contrast, truly adaptive systems of the future will be able to assess complex environments without the need for explicit cues, and should be able to produce new behaviours on the fly in response to previously unseen environments. However, the self-assembling research community is still far from being able to create such a truly adaptive system.

The morphologically responsive system we investigate in this thesis is a first step towards these truly adaptive systems of the future and by itself already represents a significant advance of the state of the art in self-assembling robotics. Indeed, to the best of our knowledge, this is the first study to investigate morphlogical responsiveness using a physically embodied robotic platform. While the problems of decisional autonomy and morphology control we address are generically applicable to all self-assembling systems,

we ground our work on the swarm-bot robotic platform. Over the course of this thesis, we use the swarm-bot platform to develop and analyse self-assembly response mechanisms of gradually increasing sophistication. Wherever possible, we abstract the behavioural control away from the underlying platform to make our results potentially applicable to a wider range of robotic platforms.

Initially, we focus on decisional autonomy. We present distributed behavioural control that gives the system decisional autonomy over when and if to self-assemble, based on parameters of a simple hill crossing task — a low hill can be crossed by a single robot, while a steeper hill causes a single robot to topple, and therefore requires the robots to self-assemble in order to cross it. In a series of real-world hill crossing experiments, we show how different self-assembly timing and decision strategies can enhance system efficiency and performance. This allows us to also consider the costs of self-assembly, which are often neglected in self-assembly studies. Although our analysis is focused on the hill crossing task, we nonetheless show that our self-assembly strategies are generic by applying them to two other real-world tasks — hole crossing and robot rescue. In the context of the robot rescue task, we investigate the problem of resource allocation in a self-assembling system.

We then investigate morphology control. We present behavioural control that allows the swarm-bot platform to control the angle of connections between components. On top of this low level control, we build a distributed morphology control system — *swarmorph* — that can generate periodically repeating morphologies. We abstract swarmorph into a scripting language *swarmorph-script* and add instructions that enable symbolic communication between connected robots. Swarmorph-script can generate arbitrary morphologies. Its inherently distributed nature makes it scalable and robust to local failures. Furthermore, because swarmorph-script instructions are based on abstract representations of the swarm-bot angular connection mechanism, the system should be directly applicable without modification to any robotic platform for which a similar angular connection mechanism can be implemented.

We demonstrate in both simulated and real-world experiments on the swarm-bot platform how a self-assembling robotic system can respond to different tasks by forming appropriate morphologies. We also present some initial results of ongoing research that we are conducting with the goal of creating a system capable of displaying genuine adaptivity when faced with never before seen environments.

1.1 Goals of This Thesis

The primary goal of this thesis is to advance the state of the art in self-propelled self-assembling systems to enable a morphological response on a real-world platform.

The swarm-bot platform on which we conduct our research was developed over the course of the eponymous project funded to the tune of 2 million euros by the European Commission under the Future and Emerging Technologies initiative (FET). The stated goal of the swarm-bots project was to create *"an artefact composed of a number of simpler... robots... capable of self-assembling and self-organising to adapt to its environment"*. The work conducted in this thesis started as the swarm-bots project was drawing to a close. At this stage, the mechatronics and behavioural control necessary to achieve autonomous stochastic self-assembly had already been developed. The project was selected as one of the success stories of the FET programme. However, due to the limited time frame of the project, the project based research had not yet tackled the problem of morphological responsiveness — an essential component of adaptivity for self-assembling systems (as

discussed above).

The goal of this thesis was thus to take up the gauntlet of the early swarm-bot vision. In practice, that meant taking a robotic platform capable of self-assembly, and expanding the behavioural control to the point at which self-assembly could be used in response to real-world environmental conditions. As the thesis progressed, additional goals emerged: to use this enhanced behavioural control to investigate circumstances under which self-assembly adds value to a robotic system, and to generate a morphology generation logic that could also be potentially applicable to other robotic platforms.

1.2 Scientific Contribution of This Thesis

In this section, we enumerate the contributions of the research in this thesis, and list the scientific publications that resulted.

1. Demonstration of a distributed decision making mechanism that allows a multi-agent system to 'choose' when and if to self-assemble on the basis of environmental contingencies.

   - **R. O'Grady**, R. Groß, F. Mondada, M. Bonani, and M. Dorigo. Self-assembly on demand in a group of physical autonomous mobile robots navigating rough terrain. In *Proceedings of the 8th European Conference on Artificial Life, ECAL 2005*, volume 3630 of *Lecture Notes in Artificial Intelligence*, pages 272–281. Springer-Verlag, Berlin, Germany, 2005
   - **R. O'Grady**. Adaptive swarm robotics in rough terrain navigation. Mémoire de DEA, Université Libre de Bruxelles, Bruxelles, Belgium, 2005
   - M. Dorigo, E. Tuci, V. Trianni, R. Groß, S. Nouyan, C. Ampatzis, T. H. Labella, **R. O'Grady**, M. Bonani, and F. Mondada. *SWARM-BOT: Design and Implementation of Colonies of Self-assembling Robots*, chapter 6, pages 103–135. IEEE Computational Intelligence Society, New York, 2006

2. Analysis of the costs and benefits that a multi-robot system gains from self-assembly. Identification of different types of benefit and quantitative analysis of these benefits.

   - **R. O'Grady**, R. Groß, A. L. Christensen, F. Mondada, M. Bonani, and M. Dorigo. Performance benefits of self-assembly in a swarm-bot. In E. Grant and T. C. Henderson, editors, *Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-2007)*, pages 2381–2387. IEEE Computer Society Press, Los Alamitos, CA, 2007

3. Conceptual division of autonomous self-assembly mechanism into different self-assembly strategies. Demonstration of how the strategies can be applied to different real world tasks.

   - **R. O'Grady**, R. Groß, A. L. Christensen, and M. Dorigo. Self-assembly strategies in a group of autonomous mobile robots. *Autonomous Robots*, 28(4):439–455, 2010
   - R. Groß, **R. O'Grady**, A. L. Christensen, and M. Dorigo. *Handbook of collective robotics: Fundamentals and challenges*, chapter The swarm-bot experience: What you can get from a group of physically cooperating robots. Pan Stanford Publishing, Singapore. To appear

- **R. O'Grady**, C. Pinciroli, R. Groß, A. L. Christensen, and M. Dorigo. Swarmbots to the rescue. In *10th European Conference on Artificial Life, ECAL 2009*. Springer-Verlag, In Press

4. Design and implementation of a distributed morphology control mechanism (*swarmorph*) for self-assembling systems.

   - **R. O'Grady**, A. L. Christensen, and M. Dorigo. SWARMORPH: Multi-robot morphogenesis using directional self-assembly. *IEEE Transactions on Robotics*, 25(3):738–743, 2009
   - A. L. Christensen, **R. O'Grady**, and M. Dorigo. Morphology control in multirobot system. *IEEE Robotics and Automation Magazine*, 14(4):18–25, 2007
   - A. L. Christensen, **R. O'Grady**, and M. Dorigo. A mechanism to self-assemble patterns with autonomous robots. In *Advances in Artificial Life, Proceedings of ECAL 2007*, volume LNAI 4648 of *Lecture Notes in Artificial Intelligence*, pages 716–725. Springer-Verlag, Berlin, Germany, 2007
   - **R. O'Grady**, A. L. Christensen, and M. Dorigo. Self-assembly and morphology control in a swarm-bot. In E. Grant and T. C. Henderson, editors, *Video Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2551–2552. IEEE Computer Society Press, Los Alamitos, CA, 2007
   - A. L. Christensen, **R. O'Grady**, and M. Dorigo. Morphogenesis: Shaping swarms of intelligent robots. In *AAAI-07 Video Proceedings*. AAAI Press, Menlo Park, CA, 2007. Winner of the "Best Video" award

5. Design and implementation of a scripting language (*swarmorph-script*) to allow for rapid implementation of new morphologies, including the ability to replicate scripting instructions from one robot to another.

   - **R. O'Grady**, A. L. Christensen, and M. Dorigo. *Morphogenetic Engineering: Toward Programmable Complex Systems*, chapter SWARMORPH: Morphogenesis with Self-Assembling Robots. "Studies on Complexity" Series (To Appear). Springer-Verlag
   - **R. O'Grady**, A. L. Christensen, and M. Dorigo. Autonomous reconfiguration in a self-assembling multi-robot system. In M. Dorigo, M. Birattari, C. Blum, M. Clerc, T. Stützle, and A. F. T. Winfield, editors, *Ant Colony Optimization and Swarm Intelligence, Sixth International Conference, ANTS 2008*, volume 5217 of *Lecture Notes in Computer Science*, pages 259–266. Springer-Verlag, Berlin, Germany, 2008
   - A. L. Christensen, **R. O'Grady**, and M. Dorigo. SWARMORPH-script: A language for arbitrary morphology generation in self-assembling robots. *Swarm Intelligence*, 2(2-4):143–165, 2008

6. Demonstration of a self-assembling system responding to different tasks by forming specific, appropriate morphologies.

   - **R. O'Grady**, A. L. Christensen, C. Pinciroli, and M. Dorigo. Robots autonomously self-assemble into dedicated morphologies to solve different tasks. In van der Hoek, Kaminka, Lespérance, Luck, and Sen, editors, *Proceedings*

of the 9th international Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2010, pages 1517–1518. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2010,

- A. L. Christensen, **R. O'Grady**, and M. Dorigo. Parallel task execution, morphology control and scalability in a swarm of self-assembling robots. In *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions, Robótica 2009*, pages 127–133, Castelo Branco, Portugal, 2009. IPCB-Instituto Politcnico de Castelo Branco

1.3 Other Scientific Contributions

Whilst researching the material presented in this thesis, I also contributed to other research topics. These contributions are briefly presented in this section.

1. Investigation of a distributed group size regulation mechanism involving flying and ground based swarms of robots.

   - **R. O'Grady**, C. Pinciroli, A. L. Christensen, and M. Dorigo. Supervised group size regulation in a heterogeneous robotic swarm. In *9th Conference on Autonomous Robot Systems and Competitions, Robótica 2009*, pages 113–119. IPCB-Instituto Politcnico de Castelo Branco, Castelo Branco, Portugal, 2009
   - C. Pinciroli, **R. O'Grady**, A. L. Christensen, and M. Dorigo. Self-organised recruitment in a heteregeneous swarm. In *Fourteenth International Conference on Advanced Robotics – ICAR 2009*, pages 1–8, 2009. Proceedings on CD-ROM, paper ID 176

2. Investigation of a model-free technique to evolve neural network controllers for endogenous and exogenous fault detection.

   - A. Christensen, **R. O'Grady**, M. Birattari, and M. Dorigo. Fault detection in autonomous robots based on fault injection and learning. *Autonomous Robots*, 24(1):49–67, 2008
   - A. L. Christensen, **R. O'Grady**, and M. Dorigo. Synchronization and fault detection in autonomous robots. In Raja Chatila, J. P. Merlet, and C. Laugier, editors, *Video Proceedings of the 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4139–4140. IEEE Computer Society Press, Los Alamitos, CA, 2008
   - A. L. Christensen, **R. O'Grady**, M. Birattari, and M. Dorigo. Automatic synthesis of fault detection modules for mobile robots. In *Proceedings of the 2007 NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2007)*, pages 693–700. IEEE Computer Society Press, Los Alamitos, CA, 2007
   - A. L. Christensen, **R. O'Grady**, M. Birattari, and M. Dorigo. Exogenous fault detection in a collective robotic task. In *Proceedings of the 9th European Conference on Artificial Life, ECAL 2007*, volume LNAI 4648 of *Lecture Notes in Artificial Intelligence*, pages 555–564. Springer-Verlag, Berlin, Germany, 2007

3. Investigation of a firefly-like synchronisation mechanism to aid fault communication and detection.

- A. L. Christensen, **R. O'Grady**, and M. Dorigo. From fireflies to fault tolerant swarms of robots. *IEEE Transactions on Evolutionary Computation*, 13(4):754–766, 2009

4. Investigation of spatially targeted communication in a heterogeneous swarm using camera and LEDs.

   - N. Mathews, A. L. Christensen, E. Ferrante, **R. O'Grady**, and M. Dorigo. Establishing spatially targeted communication in a heterogeneous robot swarm. In van der Hoek, Kaminka, Lespérance, Luck, and Sen, editors, *Proceedings of the 9th international Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2010*, pages 939–946. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2010

1.4 Publication Summary

6 Journal papers (2 as first author – *, 4 based on thesis content – †)

- 2010, *Autonomous Robots* (*†)
- 2009, *Swarm Intelligence* (†)
- 2009, *IEEE Transactions on Evolutionary Computation*
- 2009, *IEEE Transactions on Robotics* (*†)
- 2008, *Autonomous Robots*
- 2007, *IEEE Robotics & Automation Magazine* (†)

11 Conference papers (5 as first author – *, 6 based on thesis content – †)

- AAMAS 2010
- ECAL 2009 (*†)
- ICAR 2009
- Robotica 2009 (†)
- Robotica 2009 (*)
- ANTS 2008 (*†)
- IROS 2007 (*†)
- ECAL 2007
- ECAL 2007 (†)
- AHS 2007
- ECAL 2005 (*†)

3 Book chapters (1 as first author – *, 1 based on thesis content – †, 2 partially based on thesis content)

- 2010 (to appear), *"Studies on Complexity" Series*, Springer-Verlag (*†)
- 2010 (to appear), *Handbook of collective robotics*, Pan Stanford Publishing
- 2006, *Computational Intelligence: Principles and Practice*, IEEE Computational Intelligence Society

3 Video Proceedings (1 as first author – *, 2 based on thesis content – †)

- IROS 2008
- AAAI 2007 – Winner of the "Best Video" award (†)
- IROS 2007 (*†)

Media (Television)

- German Channel WDR, *Science Series: Quarks & Co*, 2007, 'Das Geheimnis des Schwarms'

1.5 Thesis Structure

The thesis is structured as follows. In Chapter 2, we discuss related research. We discuss related work in the areas of inspiration from nature, multi-robot systems, modular reconfigurable robotic systems, self-assembly, and morphology control.

In Chapter 3, we present the swarm-bot robotic platform that was used in our experimentation. We discuss some of the actuation mechanisms we used to achieve gripping and collective motion, and some of the sensing mechanisms used to detect features of the environment.

In Chapter 4, we investigate decisional autonomy in a self-assembling system. We present distributed behavioural control that allows a group of robots to choose when and if to self-assemble on the basis of environmental contingencies. The task we consider is a hill crossing task, parameterised by the steepness of the hill. A simple hill can be crossed individually, while a steep hill requires the robots to self-assemble and navigate over the hill as a collective entity.

In Chapter 5, we investigate the benefits (and costs) of self-assembly. We show how the behavioural control used in chapter 4 can be genericised into a set of self-assembly strategies. By comparing the performance of these different strategies in a more sophisticated version of the hill crossing task (with a hill whose orientation is also variable), we perform the first quantitative analysis of the benefits of self-assembly to an autonomous robotic system.

In Chapter 6, we investigate resource allocation in a self-assembling system by applying one of the self-assembly strategies to a different problem domain — a 'robot rescue' scenario. In this scenario, the system is required to dynamically allocate resources to perform group transport of broken robots of a priori unknown sizes.

In Chapter 7, we present a low level mechanism that gives robots control over the timing and direction of connections formed during the self-assembly process. This directional self-assembly mechanism enables the higher level work on morphology control in subsequent chapters.

In Chapter 8, we present *swarmorph*, a distributed morphology control system that relies on directional self-assembly. Morphologies are grown through the repeated application of distributed rules. No symbolic communication is used.

In Chapter 9, we present an extension of our work on morphology control—a scripting language called *swarmorph-script* that is dedicated to the generation of robotic morphologies. The scripting language allows for rapid prototyping of new morphologies. The generation of arbitrary morphologies is enabled by the inclusion of primitives in the scripting language that allow for symbolic communication between connected robots.

In Chapter 10, we integrate work from earlier chapters. Using this integrated system, we conduct experiments in simulation and on real-robots where the robotic system must tackle a sequence of tasks, in which each task requires a different dedicated morphology.

In Chapter 11, we describe ongoing work that has the goal of making our system truly adaptive (instead of relying on pre-determined morphologies). We discuss possible future research directions and present our conclusions.

Chapter 2

Context and Related Work

In this chapter, we present the research context of this thesis and discuss related research studies. In Section 2.1, we describe the field of swarm robotics. In Section 2.2, we discuss related work in the field of self-assembling systems — a particularly relevant type of swarm robotics system. In Section 2.3, we discuss related work in the field of morphologically controllable robotic systems. Finally, in Section 2.4, we discuss related work in using distributed robotic systems in task application scenarios.

2.1 The Principles of Swarm Robotics

The research in this thesis is conducted using the swarm-bot robotic platform (see Chapter 3). As the name suggests, this robotic system belongs to a class of robotic system known as a swarm robotics system. Such systems emphasise distributed control of large numbers of robots, and these concepts underlie all of the research in this thesis.

Swarm robotics systems rely on the principles of self-organisation to allow large groups of relatively simple agents to carry out relatively complex tasks [35, 42, 82, 85, 90, 91]. In the following section (Section 2.1.1), we discuss the fundamental principals of self-organisation. In Section 2.1.2, we present some examples of self-organising systems in Nature, that have been used as inspiration for swarm robotics systems. In Section 2.1.3, we describe the typical characteristics of swarm robotics systems. Relevant instances of swarm robotics systems are discussed in Sections 2.2, 2.3 and 2.4.

2.1.1 Self-Organisation

Camazine *et al.* [11] define self-organisation as: *"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."* Camazine uses the word pattern to mean: *"a particular, organized arrangement of objects in space or time."*

Patterns in self-organised system are generated without external guidance or templates. The rules specifying interactions among the lower-level components of the system use only local information. Garnier *et al.* [58] identify four fundamental dynamics that underly most self-organisation processes:

1. *Positive feedback that results from the execution of simple and local behavioural rules. Positive feedback is a recurrent influence that amplifies an initial state.*
2. *Negative feedback, that leads to stabilization, for example, when available resources are exhausted.*
3. *Randomness of the individual decision making process, that allows the colony to find new solutions.*
4. *Multiple direct or stigmergic interactions among individuals that lead to the appearance of a global and enduring structure.*

Self-organising swarm robotics systems have the advantage that each robotic agent need only have comparatively limited cognitive abilities and knowledge of the environment (if any). Such distributed self-organising systems also tend to avoid the problems inherent with more traditional approaches that rely on a centralised management of the robots activities and on comprehensive information exchange. These traditional approaches rapidly become infeasible as the size and complexity of the robotic agents, robotic populations and tasks grow. By contrast, in swarm robotics systems, the emergence of global behaviour from local interactions allows such systems to scale well as one increases the number of participating agents.

Furthermore, the relative simplicity of individual agents in swarm robotics systems makes such systems potentially easier and cheaper to manufacture than systems comprising monolithic robotic agents. At the same time, the lack of a central point of coordination encourages robustness and flexibility. There is no single point of failure, and individual agents tend to be interchangeable, leading to inherent redundancy.

2.1.2 Inspiration From Nature

Self-organisation processes are responsible for the generation of much of the order that we observe in natural systems. In biological systems, examples include the formation of cell membranes and multi-cellular structures, information processing in brains, the synchronous flashing of fireflies, flocks of birds, and the division of labor in social insects [12, 13, 14, 34, 37, 38, 45, 130, 133, 134]. In chemical and physical systems examples include molecular self-assembly, reaction-diffusion systems, sand dunes, stars, and galaxies.

Robotics has traditionally borrowed heavily from observation of natural processes and continues to rely heavily on biological inspiration [122]. Some researchers even use robotics as a means to better understand natural processes [126]. Swarm robotics (along with other so called *swarm intelligence* research) particularly emphasises its inspiration from self-organising processes in Nature [7]. In the remainder of this section we analyse three types of naturally occurring self-organisation mechanisms that are particularly relevant to this thesis.

2.1.2.1 Self-Assembly

Self-assembly is a central topic in this thesis. Here, we describe some of the biological inspiration for some of the robotic self-assembling systems that are discussed below in Section 2.2.

Self-assembly is a widely observed phenomenon in social insects [1, 134]. Anderson *et al.* [1] identified 18 distinct types of self-assembled structures that insects build: *bivouacs,*

Figure 2.1: Self-assembly in a natural system. *Œcophilla longinoda* worker ants connect to one another to pull together large leaves during nest construction.

bridges, curtains, droplets, escape droplets, festoons, fills, flanges, ladders, ovens, plugs, pulling chains, queen clusters, rafts, swarms, thermoregulatory clusters, tunnels, and walls. In some cases (e.g., an ant raft) the individuals assemble into *"a formless random arrangement"*, whereas in other cases (e.g., an ant ladder) the insects show a degree of morphology control in the self-assembly process, that is, individuals assemble into a *"particular (required) arrangement"*. Pulling structures have been observed in a few ant species (e.g., Eciton burchellii) as well as in honey bees (*Apis mellifera*) [1]. The structures generate torque, for instance, to fix a large prey to the floor or to bend a leaf during nest construction.

During nest construction, *Œcophylla longinoda* worker ants form connected pulling chains by gripping each other with their mandibles [89, 88]. Collectively, they can pull leaves together that are too large and stiff for a single ant to manipulate (see Figure 2.1). 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 [97]. *Dolichoderus cuspidatus* ants have been observed forming living bridges of connected ants that other colony members then traverse.

Theraulaz *et al.* [142] investigated self-assembly processes in *Linepithema humile*. They demonstrated a 'droplet' forming behaviour in which ants aggregated at the end of a rod and formed droplets containing several assembled ants that eventually detached from the rod. They modelled the system with distributed agents, and arrived at a model that successfully reproduced various properties of the ant behaviour, including droplet size and inter-drop interval. This droplet behaviour has yet to be observed in the wild, and its biological function is still unknown.

2.1.2.2 Group Transport

Group transport is highly relevant to this thesis as it forms the basis of the 'Robot Rescue' task that we investigate in Chapter 6.

Sudd [141] studied the transport of prey by single ants and by groups of ants of the species *Pheidole crassinoda*. He reported on two types of reconfiguration in groups of ants pulling a prey object — realignment and repositioning. In realignment, an ant changes the orientation of its body without releasing its hold on the prey, while in repositioning, an ant releases its hold and grasps the prey again at a different position. Group transport behaviour has been observed in at least 39 different species of ant [94].

Super-linear efficiency in group transport has been reported in some ant species [46, 72].

In the genus *Pheidologeton*, for instance, on average an ant engaged in group transport held at least 10 times the weight it did as a solitary transporter [93].

Some species of social spider have also been observed performing group transport. It has been theorised that the spiders coordinate their actions by means of stigmergic communication transmitted through the object being transported [146].

2.1.2.3 Aggregation

The self-assembly mechanism initially presented in Chapter 4 and all subsequent self-assembly strategies (Chapters 5 - 6) use an aggregation module to allow randomly moving robots to come together in a single location as a precursor to self-assembly. The directional self-assembly mechanism that we present in Chapter 7, which underlies all of our subsequent morphology control research (Chapters 8 - 10) also uses an aggregation-like mechanism to ensure that robots stay within visual range.

An example of aggregation is observed in the bark beetle larvae Dendroctonus micans [36]. 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).

Cockroach aggregation behaviour has also been much studied, and also used as explicit inspiration for robotic studies. In a series of experiments in a white circular arena, Jeanson *et al.* [77] derived a probabilistic behavioural model for the German cockroach *Blattella germanica*. They showed that individuals switch probabilistically between two alternative behaviours: random walk and resting. Analysis revealed that the probability for a cockroach to switch to (or remain in) one of these two states depends on the number of resting cockroaches in its neighbourhood (in direct antenna contact): as this number increases, the stopping probability also increases, while the probability of leaving an aggregate decreases. Garnier *et al.* [59] used Jeanson's behavioural model to show that a group of cockroach-like robots can achieve a collective choice between two different shelters in the environment through simple local interactions. In a separate study, Halloy *et al.* [69] demonstrated that robotic agents could actually interact with cockroaches and influence their aggregation behaviour.

2.1.3 Characteristics of Swarm Robotics Systems

Dorigo and Sahin [40] identified four criteria to give an approximate measure of the degree to which a robotic system can be considered a swarm robotics system:

1. *The study should be relevant for the coordination of large numbers of robots.*

2. *The system being studied should consist of relatively few homogeneous groups of robots, and the number of robots in each group should be large.*

3. *The robots being used in the study should be relatively simple and incapable, so that the tasks they tackle require the cooperation of the individual robots.*

4. *The robots being used in the study should only have local and limited sensing and communication abilities.*

In some swarm robotics systems, the individual agents can autonomously form physical connections with one another. This class of so called *self-assembling* system is particularly relevant to this thesis, and is discussed in the following section (Section 2.2).

2.2 Self-assembling Robotic Systems

In the past 50 years, many researchers have designed and studied modular systems whose components — ranging from passive mechanical parts to mobile robots — can self-assemble into physically connected structures [65].

Self-assembling systems can be classified as self-propelled or externally propelled. Agents in self-propelled systems use an internal power source to create the kinetic energy required to move. By contrast, in order to move, externally propelled systems rely on kinetic energy from external sources in the environment. The swarm-bot robotic platform that we use in this thesis is a self-propelled system. However, in the literature more morphology control work has been done with externally propelled systems (see Section 2.3). In the following two sections we present the systems and their self-assembly capabilities. Applications of these systems for morphology control are described in Section 2.3 and for task execution in Section 2.4.

2.2.1 Self-propelled Systems

Almost half a century ago, Jacobson [76] designed a simple system for self-assembly and self-replication—the Reproductive Sequence Device One (RSD I). The system resembled a model railway, with two types of modules, head modules and tail modules, moving on the rails. A 'seed' object consisting of a connected head and tail module pair could be manually assembled on a siding, whereupon the system would autonomously replicate the seed object on an adjacent siding.

Fukuda *et al.* have proposed several implementations of the so called cellular robotic system or CEBOT [48, 54]. The fundamental concept involves a series of modules with different functions that would connect together to create higher level robotic entities with different capabilities. The CEBOT Mark I module was cuboid with active connectors (latch) on one side and passive connectors (pins) on the opposite side [49, 50]. One module could approach and bind to another module, but only if very precisely pre-aligned by the experimenter. CEBOT Mark III [55] used a similar latch and pin binding mechanism, but with 3 active and 3 passive connecters per module, resulting in a hexagonal module shape. CEBOT Mark II [51, 52, 47] and CEBOT Mark IV [53, 56] were characterised by the use of a mechanical hook as the connecting mechanism, and the geometry of the modules was modified to encourage a higher approach alignment tolerance. In each of the Mark II, Mark III and Mark IV systems, Fukuda *et al.* demonstrated the successful docking of a mobile module with a stationary module. Fukuda *et al.* also showed how connected modules could communicate and coordinate their actions so as to approach and connect with an additional module [47].

Hirose *et al.*'s Gunryu concept [71] consists of a number of autonomously mobile units each equipped with an actuator that allows the modules to form physical connections with each other. However, only two prototype modules connected by a passive arm have been built.

Rubenstein *et al.* demonstrated that the CONRO self-reconfigurable system (see Section 2.3.2 is capable of autonomous docking (self-assembly) [125]. Two robotic chains were used, each consisting of two linearly-linked CONRO modules. To ensure that both chains perceive each other, the initial placement of the two robots had to be no more that 15 cm apart, and the robots needed to be facing each other with with an angular displacement of no more than 45 °. In contrast with the research presented in this thesis, the behavioural control used by Rubenstein *et al.* was heterogeneous, both at the level of the individual constituent modules, and at the higher level used to control the two composite robotic entities.

Super-Mechano Colony (SMC) [32, 70] is composed of one parent unit and several child units that are responsible for locomotion when they are attached to the parent unit. In [149], Yamakita *et al.* demonstrated how the child units could rearrange themselves for optimal tracking performance at different speeds. Three child modules were manually arranged into a chain pulling the parent module. The two child modules at the end of the chain were able to disconnect, follow a predefined path, and reconnect to the parent module in the appropriate configuration. In the SMC Rover [96], the child units are called Uni-Rovers. Each Uni-Rover is composed of a wheel and a single manipulator. The Uni-Rovers can attach to the parent unit at one of six locations. When six Uni-Rovers connect to the parent unit, the artifact effectively becomes a six wheeled rover. Recently, Groß *et al.* [61] ported a control algorithm for autonomous self-assembly from the swarm-bot platform to the SMC platform. The ported controller was capable of letting a child module approach and assemble with another module, for approaching angles up to 150 °.

The docking of a mobile modular robot with a stationary modular robot has been demonstrated with the self-reconfigurable M-TRAN III platform [98] (see Section 2.3.2). The docking was supported by sensory feedback from a dedicated camera module mounted on the stationary robot. Both image processing and control were performed on an external PC. Docking succeeded with a variety of initial positions and orientations. The authors also demonstrated an integrated sequence comprising both self-assembly and subsequent self-reconfiguration. Post-assembly reconfiguration occurred through the movement of modules within the assembled structure.

The swarm-bot platform used in this thesis is also a self-propelled self-assembling system. It differs from the above systems primarily in its connection mechanism, which consists of a gripper and a ring around the body of each robot that enables a robot to receive connections on up to two thirds of its perimeter, with a relatively high misalignment tolerance. Details of the swarm-bot platform are given in Chapter 3. Recently, self-assembly has also been demonstrated with the swarm-bot system. A group of robots was programmed to form a single connected composite entity. Experiments were conducted with up to 16 physical robots [62]. This work provided the launching point for the research presented in this thesis.

2.2.2 Externally Propelled Systems

Penrose and Penrose [120] built a system of modules made of wood that represents the first mechanical analogue for self-replication, and thereby showed *"how reproduction can be demonstrated by an exceedingly simple mechanism"*. In their system, passive mechanical parts moved on a linear holding tray that was agitated laterally. In their default position, the parts do not link under the influence of shaking alone. If a seed object composed of two parts, mechanically linked to each other, is added, it replicates by physically interacting with the other parts on the track.

Hosokawa *et al.* [73] analysed the dynamics of a system of self-assembling hexagons. Their system was composed of simple, homogeneous modules, placed in a flat box, which was then rotated in a vertical plane. The modules do not have any state. Simple logic is implemented through anisotropic binding preferences. Geometrically, each module is an equilateral triangle with permanent magnets of opposite polarization in two of its sides. Consequently, at most six modules can bind together, thus forming a hexagon.

Breivik [8] developed a system of template-replicating polymers. The system uses two types of modules, A and B. The modules could bind in two different ways, based on logic coded into the hardware binding mechanism. Breivik describes the formation of new sequences based on previously formed polymers acting as templates in an experiment with 70 modules freely floating in a liquid two-dimensional environment that was externally agitated.

White *et al.* studied both two dimensional and three dimensional systems with passively moving modules. In two-dimensions [148], White *et al.* used either triangular or square modules that floated freely on an air table attached to an orbital shaker. Programmable logic in the form of switchable electromagnets was used to control the self-assembly process. For both types of shape, the system displayed self-reconfiguration through controlled self-assembly and disassembly. The authors present analytical and computational models of their system, both of which indicate quadratic growth for unconstrained growth of self-assembled entities (assuming the number of available modules is not a limiting factor). Constrained growth (into particular shapes) was shown to depend on the particular shape and algorithm. In three dimensions [147], cuboid modules freely floating in an agitated liquid medium bound to each other either through switchable electromagnets, or (in a different system) through a system of valves inside the modules combined with an external pressure in the liquid medium. Controlled reconfiguration with electromagnetic bindings was demonstrated with two modules and a success rate of 24 %. Random, irreversible self-assembly was demonstrated with up to four modules. Using fluid pressure, reconfiguration with two modules was also demonstrated.

Bishop *et al.* [5] created a system of trianglular modules, that float freely on an air table. Connections are established through a system of controllable permanent magnets on each side of the triangle. Self-assembly logic is coded using a graph grammar which is distributed to each module. Once connected, communication between modules is established to allow both connected modules to determine whether they should remain bound or detach. Experiments were performed with up to six modules. The problem of finding a grammar associated with a desired target shape is discussed in [81] (see Section 2.3).

Much of the morphology control work in self-assembling systems has been conducted with externally-propelled systems (see Section 2.3). In this thesis, we also study morphology control in a self-assembling system. However, the system we use (the swarm-bot platform) is self-propelled.

2.3 Morphology Control in Distributed Robotic Systems

Morphology control research is motivated by the pursuit of adaptability. Any biological or mechanical agent can carry out its task more effectively if its morphology is well suited to the task. A robotic system with morphological flexibility is theoretically, therefore, able to carry out a wider range of tasks more effectively than a robotic system with a fixed morphology.

Morphology control research is relevant to this thesis, as one of our chief contributions

is to show how a self-assembling system can firstly form different morphologies, and secondly how such a system can use these morphologies to solve different tasks. Two main bodies of research are particularly relevant to this thesis. The first body of research is the study of morphology control algorithms for distributed robotic systems, which is discussed in Section 2.3.1. The second body of research is into the hardware of so called *modular reconfigurable robotic systems*, which is discussed in Section 2.3.2. Note that some morphologically motivated research that has been conducted into the hardware of self-assembling systems is not discussed here, as it was discussed separately in Section 2.2 above.

2.3.1 Morphology Control Algorithms

Various approaches to controlling and assembling modular robots have been proposed. Challenges include trying to form appropriate morphologies and getting numerous autonomous units to operate together efficiently. In this section we discuss algorithms for creating and maintaining particular morphologies. The use of such morphologies in task-execution scenarios is discussed in Section 2.4.

Rus and Vona [128, 129] have proposed a centralized control algorithm to allow a robotic system composed of *Crystalline Atom* units to reconfigure its shape. Although a physical prototype of a *Crystalline Atom* unit has been built, the results with the proposed control algorithm were obtained analytically and in simulation.

White *et al.* [147] suggested an approach in which individual units have unique IDs and predefined locations in the target structure. A scripting language is used to specify where each unit should connect.

Jones and Matarić [78] have developed a *Transition Rule Set* compiler that takes as input the desired morphology and outputs a set of rules. Agents with limited and local sensing can follow these rules and assemble into the desired morphology. The approach proved scalable and capable of producing a large class of structures in a simulated 2D lattice world populated by simulated unit square agents.

Butler *et al.* [10] have studied generic decentralized algorithms for lattice-based self-reconfigurable robots. The algorithms are inspired by cellular automata and control is based on geometric rules. Related research into the autonomous detection of a manually assembled configuration using graph theory has been conducted on the CKbot platform [118] (see Section 2.3.2 below).

Klavins *et al.* [81] tackled the problem of defining a class of graph grammars that can be used to model and direct distributed robotic self-assembly. They show how a grammar can be synthesized that generates a desired, pre-specified target morphology.

Shen *et al.* [137] have demonstrated a bio-inspired control method based on virtual hormone for controlling swarms of robots. The virtual hormone can be propagated through the swarm causing the robots to generate patterns and/or reconfigure. The authors show results from experiments in simulation and discuss how virtual hormone could be propagated in real-world scenarios either through radio or through infrared communication.

Again in simulation, Støy and R. Nagpal [139, 140] have demonstrated algorithms for self-reconfiguration and directed growth of cubic units based on gradients and cellular automata.

Bojinov *et al.* [6] have shown how a simulated modular robot (Proteo) can self-reconfigure into useful and emergent morphologies when the individual modules use local sensing and local control rules.

Bhalla *et al.* [5] self-assembled specific shapes with passively moving two dimensional components constructed out of foam board. They used dedicated concave and convex

shapes combined with embedded permanent magnets to implement a key-lock-neutral style binding mechanism. To self-assemble each specific global shape, the authors initially used a highly abstract graph-like representation, which was then translated into a highly simplified simulation environment using spheres that could bind to each other as components, before being finally realised in the physical foam board incarnation.

Mytilinaios *et al.* [100] used the Molecube platform (see Section 2.3.2 below) to investigate the use of evolutionary algorithms to design self-replicating morphologies in a 2-D simulation environment. Zykov *et al.* [162] used a physical instantiation of the system to demonstrate the self-replication of a 4-module entity provided with an ordered supply of additional modules. The system executed a predetermined sequence of actions, successful connections being confirmed through explicit communication.

Schultz *et al.* [132] developed a potentially generic scripting language for self-reconfiguring robots that they implemented on a real-world instance of the ATRON platform, and on a simulated version of the MTRAN platform (see Section 2.3.2 below). The language can be used to code reconfiguration sequences. Unlike our swarmorph-script approach, the script relied on global state being shared across all participating modules, and for all modules to contain full reconfiguration instructions before the start of each experiment.

A related research field is that of multi-robot formation control [3, 33, 86, 87]. In formation control research, groups of robots in a multi-robot system steer themselves into one or more pre-specified formations. Formations are particular arrangements of robots, but without physical connections between the robots. Proposed approaches to formation control include the use of virtual structures, leader-follower schemes, and decentralised, behaviour-based methods. Mechanisms to maintain these formations while the group is in motion are also studied. By contrast, in morphogenesis research the shape of the composite entity is maintained automatically as a result of the physical connections between the robots. Recently, Rubenstein *et al.* [127] modelled simple simulated robots that could form and hold a desired swarm shape, independent of the total number of agents. In their system, when the shape is damaged by the removal of some of the robots, the remaining agents recover the former shape, but on a smaller scale. Bachrach *et al.* [2] propose a control language 'Protoswarm' that allows one to specify spatial behaviours at a group level, which by treating the environment as a vector field are then translated into individual robot behaviours.

The swarmorph morphology control approach that we present in Chapters 8 - 10 of this thesis is unique because it can autonomously create arbitrary physically connected morphologies, while still operating within the restrictions imposed by a physically embodied robotic system. Indeed, we demonstrate all of our algorithms on real robots. Much of the morphology control research mentioned in this chapter takes place in simulation. For many studies, the goal is to investigate high-level properties of a novel morphology control mechanism. In these studies, the simulation environments tend not to incorporate many of the physical constraints inherent in embodied robotic systems [78, 81, 137, 139, 140]. Some simulation based studies have tried to take into account physical constraints, or to test particular aspects of their systems in limited real-world experiments [6, 10, 100, 118, 128, 129]. But direct real-world morphology control research remains the exception rather than the rule, and has usually had very different aims from the swarmorph approach — either the demonstration of a specific sequence of actions [162], or generating morphologies using unique IDs and predefined locations for individual components [147].

2.3.2 Modular Reconfigurable Robotic Systems

No study on morphology control in the context of robotics would be complete without mentioning self-reconfigurable modular robotic systems. These modular systems are motivated by the investigation of morphological flexibility. They explore morphological flexibility through the use of connected modular components. The past 20 years has produced a large body of research in the area of self-reconfigurable modular robotic systems and associated connection mechanisms [154]. The individual modules vary in autonomy and independence of control from system to system. However, in contrast with multi-robot systems (including self-assembling systems), the individual modules are usually quite simple and can seldom carry out meaningful tasks independently.

In the 1980's, Fukuda and Nakagawa [49], inspired by (biological) cellular organisms, proposed the concept of "dynamically reconfigurable robotic system"—a pioneering work that laid the foundation for subsequent research on modular robotic systems and multi-robot systems. The authors anticipated potential applications of such modular systems *"in many fields, e.g. maintenance robots, more advanced working robots, free-flying service robots in space, more evolved flexible automation, etc."*.

Hirose *et al.* [71] investigated a modular robot concept for planetary exploration, and described potential benefits of such systems in the context of autonomous all-terrain locomotion. For example, *"a single unit by itself will fall off into the crevice, but if it is a connected body, falling can be prevented"*. On the other hand, *"the torso may be separated into several groups, and each of those groups can function as an autonomously distributed group robot"*.

Yim *et al.* [151] predicted that such systems would be particularly suited to applications in which versatility is critical. *"Typically, these are situations in which some information about the environment is not known a priori. Thus, a system cannot be designed specifically for a task, since the task that is needed is not known"*.

Fukuda *et al.*'s CEBOT system [47, 80] is one of the first realizations of a reconfigurable modular system. The mobile architecture consists of heterogeneous modules with different functions, e.g. to rotate, move, and bend. Various prototypes of the CEBOT system comprising different shapes and connection mechanisms have been studied.

PolyBot [150, 151, 152, 155, 161] is a modular chain robot in which each module has one degree of freedom. The PolyBot system uses hermaphroditic connection plates with four grooved pins and four holes on each plate. When two modules connect, the grooved pins on the connection plate of one module match up with the four holes on the connection plate of the other module. Connections are formed and released by a shape-memory alloy latching mechanism. It has been demonstrated with both the G2 [156] and G3 [152, 156, 161] versions of the PolyBot system that an arm consisting of multiple PolyBot modules is capable of operating in 3D space and that such an arm can grasp and dock with additional modules. Individual PolyBot modules remain incapable of autonomous motion or action, however.

The CONRO [15, 16, 119] system consists of a number of roughly shaped rectangular boxes with a female connector on one face and male connectors on three of the other faces. Each module comprises a processor, power supply, sensors, and actuators. The connectors can be rotated with respect to the body with two degrees of freedom by means of two motorized joints. A shape memory alloy actuator integrated in the active connector can rotate a latch to catch lateral grooves in the pins from the plate of the mating passive connector. IR emitters and detectors are integrated in the binding plates to support the docking and to enable communication between connected modules.

In the Millibot Train system [9] multiple mobile robots can form linear structures by connecting to each other using a pinhole-based connection mechanism. The resulting connected robotic entity can cross obstacles impassable by a single robot.

The systems presented above all employ connection mechanisms based on penetration and shape matching. This requires precise alignment of the modules when connections are formed. Furthermore, connections between modules can only be made at predefined locations on the module bodies, sometimes only at a single location.

M-TRAN [79, 99, 157] is a hybrid system in which each module consists of three parts: an active block with three active surfaces, a passive block with three passive surfaces and a link between them. Both active and passive blocks are semi-cylindrical. The connection surfaces are composed of a combination of permanent and electrical magnets. A module is equipped with an on-board microprocessor, inter-module communication/power transmission devices and inter-module connection mechanisms. M-TRAN is able to metamorphose into different robotic configurations. Examples include a legged machine for which a coordinated walking motion was generated. Self-assembly experiments conducted with the M-TRAN platform are discussed in Section 2.2.1.

ATRON [117] is a lattice-based self-reconfigurable robot with modules composed of two hemispheres joined by a single revolute joint. The system is explicitly designed to make reconfiguration less complex, especially in terms of the number of actuator commands required. Research into control using the ATRON platform is discussed in Section 2.4.

CKbot [118] is another chain like reconfigurable robot, distinguished by its small size — each module resembles a cube of side 6cm. Modules are either screwed together, or can form connections using magnets. Research into morphology control using the CKbot platform is discussed above in Section 2.3.1.

SuperBot [135] is a recently developed deployable self-reconfigurable system for real-world applications outside laboratories. The system takes inspiration from CONRO and M-TRAN. SuperBot combines the advantages of chain-based and lattice-based robotic systems to accomplish multi-modal locomotion. Each module can dock with other modules in six different positions. A prototype consisting of six modules has been built [131].

Molecubes [100] is a homogeneous, lattice-based reconfigurable robot. The basic component module is a $10\,cm$ cube. The cube is divided along a diagonal plane, and the two halves can swivel relative to each other. Both halves have a single face with binding electromagnets, allowing the module to connect to up to two other modules. Once a connection is made data can also be transferred through the connecting face. Morphology control experiments conducted with the Molecubes platform are discussed in Section 2.3 above.

Our work in this thesis uses the swarm-bot platform, and as such is immediately distinguishable from most of the above platforms by the ability of modules (or agents) in the system to act independently. Morphology control is a central theme of this thesis, however, and as such our motivation is the same as that which drives much of the above mentioned research. Our work stands out, not just because of the differences in hardware architecture, but also because we have pushed our research to the point where real world morphologies are being formed in response to different real world tasks. In contrast, despite the progress in the field of self-reconfigurable robotics, most research remains heavily hardware focused — autonomous morphology generation is little studied (see Section 2.3.1 above), and task execution even less so (see Section 2.4 below).

2.4 Task Execution

A key focus of this thesis is taking self-assembly and morphology control mechanisms and applying them in real-world task execution scenarios. Much of the research in multi-robot systems has been abstract, or has focused on the hardware and low level control. This restriction holds particularly true in self-assembly and morphology control research, where little consideration has been given to the functional use of self-assembly or morphology control.

In this section we discuss related task execution research using self-assembling, reconfigurable and (where relevant) other multi-agent systems.

2.4.1 Coordinated Movement and Rough Terrain Navigation

Coordinated movement is a natural area to explore for multi-robot systems. In physically connected and self-assembling multi-agent systems, coordinated movement can be an essential enabling mechanism for physical cooperation. Rough terrain navigation is often cited as a key application to motivate research into such systems [135, 143]. In situations where a single rigid-body wheeled agent might fail, coordinating groups of robots might succeed in navigating rough terrain by exploiting different configurations that are better adapted to the environment encountered. Indeed, coordinated motion is a prerequisite for the physical cooperation achieved by the robots in this thesis as they perform their tasks based on navigating over obstacles (see Chapters 4, 5 and 10).

Trianni *et al.* [143] experimented with coordinated motion on the swarm-bot platform. Using artificial evolution, they created a robotic controller that allowed manually connected robots to navigate around an arena, while avoiding holes. The control was distributed. Coordination occurred through the sensing of physical forces applied to the robots bodies. The research was conducted on the same self-assembling platform we use in this thesis. However, in Trianni *et al.*'s study the focus was on the neural network controllers. The robots were pre-assembled, thus clearly differentiating Trianni *et al.*'s work from the research in this thesis, where the focus is on the timing and nature of the self-assembly process.

Motion in modular reconfigurable systems by definition involves coordination between modules. In the literature studying coordinated movement in such systems, a distinction is made between systems employing so called *lattice based* modules and systems employing so called *chain-based* modules. In lattice module based robots, different shapes are achieved through stacking the modules in different configurations. Motion in these systems tends to involve a robot 'flowing' its modules over each other. Several algorithms have been demonstrated in simulation [10]. However, practical difficulties, including mis-alignment of modules and the large number of docking and undocking steps required for movement make such motion difficult to achieve in practice. In chain module based systems, each module can bend its body to form shapes with arbitrary angles. This allow for movement based on the modules bending to change the global shape of the robot appropriately. Many such locomotion modes have already been demonstrated. A PolyBot [153] rolling track can run for 500 m on batteries and can climb stairs and fences; a M-TRAN-II robot [79] can move as a rolling track, H-walker, snake, caterpillar, and others; and a CONRO robot [136] can move as snake, caterpillar, insect, spiders, and others. Shen *et al.* made a systematic study of different types of gait achievable using the Superbot platform [135].

None of the above mentioned studies with modular reconfigurable robots tackle the problem of autonomous response — how to use these movement strategies in response to

different environmental conditions. Yu and Nagpal [158, 159, 160] provide a rare instance in the literature of a modular system that can respond to different environmental contingencies, and whose control has been tested on a real-world platform. They propose a generalized distributed consensus framework, and use it to demonstrate a variety of modular robotic systems, including an adaptive column that can adapt to external force, a modular gripper that can manipulate fragile objects, and a modular tetrahedral robot that can perform phototaxis towards a light source. They also show that control algorithms derived from this framework are provably correct, as well as being robust towards real world sensing and actuation noise. The primary difference between Yu and Nagpal's research and the work presented in this thesis is that Yu and Nagpal work with self-reconfigurable systems made of pre-assembled modules, and therefore do not have to consider the timing and nature of the self-assembly process.

On the ATRON platform, Christensen [30] showed how *'attraction points'* — virtual points in space — can be used to evolve controllers that could change shape and self-repair in response to changing environmental circumstances. Simulation based tasks included supporting and unstable roof and a self-repair of an artificial bone. Fault tolerant reconfiguration was demonstrated in a system consisting of 9 real-world modules [31]. Up to 500 simulated modules were used in other self-repair experiments.

The problem of using self-assembly as an autonomous response mechanism has been investigated by Trianni *et al.* [144] in a highly simplified simulated environment (see also [145]). In their study, a group of three simulated robots had to perform phototaxis across a terrain composed of discrete low and high temperature zones. Offline evolution was used to generated neural network controllers using a fitness function that allocated the highest fitness to the robots when they navigated individually through the high temperature zones and collectively through the low temperature zones. Our work in Chapter 5, in which we analyse the performance benefits of different self-assembly strategies, differs from this study by Trianni *et al.* in that our work is conducted using a real-world robotic system and our notions of cost and benefit have a real world meaning.

At NASA, Huntsberger *et al.* [74] investigated distributed robotic solutions for extra planetary exploration scenarios. Their method was, however, quite different from the distributed approach considered in this thesis. They used tightly coupled controllers to enable small teams of robots to carry out tasks such as coordinated transport and cliff traversal. Some researchers have also investigated multiple rovers for all-terrain exploration [43, 44]. These systems try to leverage distributed design paradigms to get higher levels of system robustness and thus better exploration performance.

The research presented in this thesis has two main features that set it apart from most of the related work described in this section. Firstly, we use coordinated movement (in conjunction with self-assembly) as an autonomous response mechanism to different environmental conditions, rather than just investigating coordinated movement in isolation. Secondly, we base our research on a real world environment and real-world physically embodied robots, rather than considering simulated abstractions of robotic systems.

2.4.2 Object Transport and Manipulation

Object transport and manipulation is highly relevant to this thesis for two reasons. Firstly, we investigate a transport problem ourselves in Chapter 6. Secondly, object transport and manipulation are considered reference tasks when studying physical cooperation between robots. Physical cooperation is, in turn, fundamentally relevant to this thesis, as it is usually cited as a key motivation for the study of robotic self-assembly.

The seminal object manipulation task in distributed robotics is clustering. Beckers *et al.* [4] conducted an experiment where robots of diameter 25 cm pushed objects using a front mounted scoop. A microswitch mounted behind the scoop ensured that the robots would continue to push up to two objects, but would stop and change direction as soon as three objects were in the scoop. The dynamics of this system resulted in the objects clustering in the arena. Similar experiments were conducted with the Khepera robotic platform by Martinoli and Mondada [90]. These studies use the simplest form of cooperation possible in a multi-robot system whereby the robots carry out different parts of a decomposable task in parallel. With this type of simple cooperation, it makes no difference if one robot carries out a task or several — it is just faster with more robots. Note that the actions of one robot may still affect those of another robot through *stigmergic* communication. However, stigmergic 'communication' refers to a robot's modifications of its environment that influence the behaviour of other robots. In fact, such modifications could just as well be used by a single robot to influence its own future behaviour. The task could, therefore, be completed using only a single robot.

Kube and Zhang [83, 84] conducted a series of experiments in which a group of physical robots was transporting a heavy object. They observed that the robots could end up pushing the same object in opposing directions. As a result of this, the transported object could become stuck. To resolve this problem, they integrated into their control policy a recovery mechanism that was inspired by the group transport of the ant species *Pheidole crassinoda* [141] (Kube and Zhang's robots mimicked the group transport rearrangement behaviour). Kube and Zhang's system has no mechanism to adjust the allocation of robots when multiple objects are present, in contrast with our work in Chapter 6, where we explicitly consider this problem.

In simulation, Perez-Uribe *et al.* [121] investigated a system in which a group of robots is required to find and transport multiple objects of two sizes: small and large. The allocation of robots to the objects was irreversible, thus creating a deadlock potential in the case of an immovable object. They investigated three primary strategies: (i) if a small object is encountered, start transporting (one robot is sufficient in this case), (ii) if a large object is encountered, call for help (by local broadcast) and wait, and (iii) if a call for help is received, go towards the emitter to engage in cooperative transport. The focus of this study was on the evolution of cooperation.

Ijspeert *et al.* [75] studied a system of physical robots in which two physically cooperating robots could pull long sticks out of the ground (the sticks could not be removed by a single robot). When a single robot tried to remove a stick, it would wait for another robot to arrive. The optimal waiting time was computed as a function of the environmental parameters.

Groß and Dorigo [63, 64, 66] used computer simulations to study the transport of heavy objects by groups of self-assembling robots. The control policies were designed by evolutionary algorithms. In [64, 66], it was assumed that only a single transportable object was present. The system in [63] could in principle cope with multiple objects, however, each robot would always attempt to transport the closest object within its perceptual field. The allocation of robots was irreversible and did not depend on the objects' resistance to motion.

Tuci *et al.* [145] conducted an experiment with physical robots that could self-assemble and transport a heavy object. The transporters were programmed to suspend their transport whenever they perceived an unconnected robot (allowing the latter to join the group). Thus, if a robot permanently failed to join the pulling structure, a deadlock occurred.

Our approach to object transport (presented as part of the 'robot rescue' task in

Chapter 6) differs from the above works in that we enable distributed reallocation of resources between tasks in a real world system, and secondly that our system has a deadlock resolution mechanism that allows the robots to detect when a task is unsolvable.

Chapter 3

The Swarm-Bot Robotic Platform

In this chapter we present the swarm-bot robotic platform which we use for all of the research in this thesis. We first give an overview of the physical characteristics of the system and its computational infrastructure (Sections 3.1 and 3.2). We then detail the sensing and actuation systems used in this thesis (Sections 3.3 and 3.4), and give details of some of the low level algorithms used to process sensory data, with a particular focus on camera data. Finally, we describe the software infrastructure we used to develop the behavioural control presented in this thesis (Section 3.5), giving details of both the simulation environment and the application programming interface that provides cross compatibility between real and simulated robots.

3.1 Physical Characteristics

The swarm-bot robotic platform was designed and built by Mondada's group at the École Polytechnique Fédérale de Lausanne (EPFL) in Switzerland. The platform is made up of multiple mobile autonomous robots called *s-bots* (see Figure 3.1) that can form physical connections with each other. 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.

The s-bot is 12 cm high without its camera turret, has a diameter of 11.5 cm without its connection mechanism and weighs close to 700 g. The main s-bot body houses most of its sensory and processing systems and can rotate with respect to the chassis by means of a motorised axis. The mechanical structure of the s-bot is illustrated in Figure 3.2.

3.2 CPU, Control Electronics and Software

The s-bot has a network of eleven processors, each of them responsible for a sub-task in the system (e.g., managing particular actuators and sensors). The most powerful processor, a 400 MHz ARM XScale CPU is in charge of the integrated management of the overall system, of the processing of the most complex sensors and of the communication with a base station for monitoring purposes.

S-bot behavioural control software is written in C++. It is compiled externally on a pc using a cross-compiler, then copied using ssh and wi-fi protocols onto the s-bot before being executed on the s-bots native Linux. Control code is executed directly on the main ARM XScale CPU. Interaction with the other processors dedicated to particular sensors and actuators is transparent to the behavioural control software, and is handled by lower

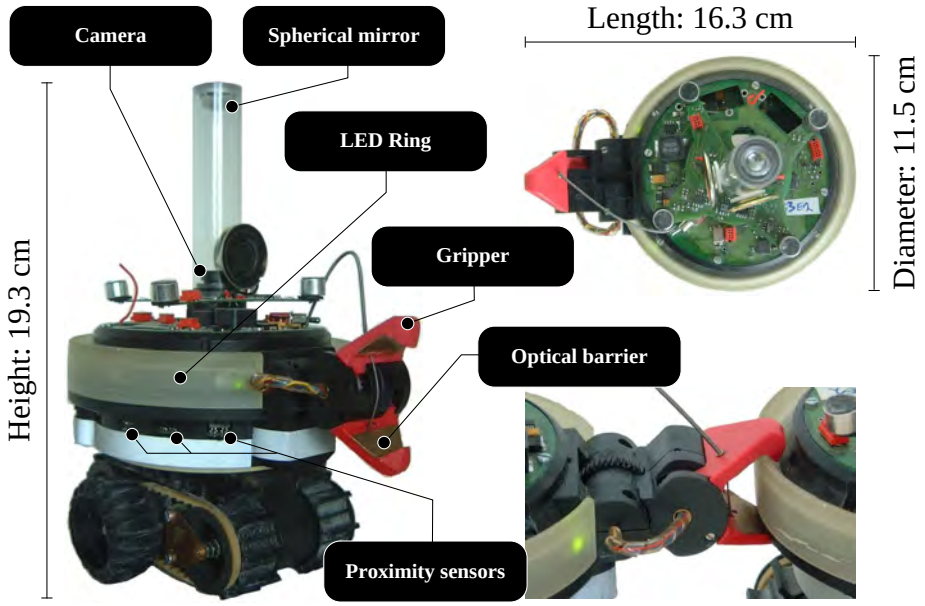


Figure 3.1: The S-bot: An autonomous, mobile robot capable of self-assembly. Weight: $\sim$ 700 g, Battery life: $\sim$ 1 h, Processor: 400 MHz XScale CPU, Operating system: Linux

level code. This lower level code forms part of the application programming interface against which the behavioural controllers in this thesis are written.

3.3 Actuation

S-bots move using a traction system that combines tracks and wheels. This system gives the s-bot good mobility on uneven terrain whilst still allowing the s-bot to rotate on the spot efficiently.

Eight sets of RGB coloured LEDs (Light Emitting Diodes) are distributed around the inside of the s-bot's transparent ring. In conjunction with the s-bot camera, these LEDs enable localised situated communication.

3.3.1 Gripping

Physical connections between s-bots are established by a gripper-based connection mechanism. Each s-bot is surrounded by a transparent ring that can be grasped by other s-bots. When closed, the gripper ensures a rigid connection. A diagrammatic representation of the s-bot gripper is shown in Figure 3.3.

The shape of the gripper and the complementary shape of the transparent ring, together ensure a large tolerance in the relative positioning of two s-bots engaged in autonomous gripping. This tolerance 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. By contrast, the majority of self-assembling systems require connecting modules to be very precisely aligned. In other self-assembling systems, therefore, the exact position of individual modules needs to be either known a priori or calculable with a high degree of precision.

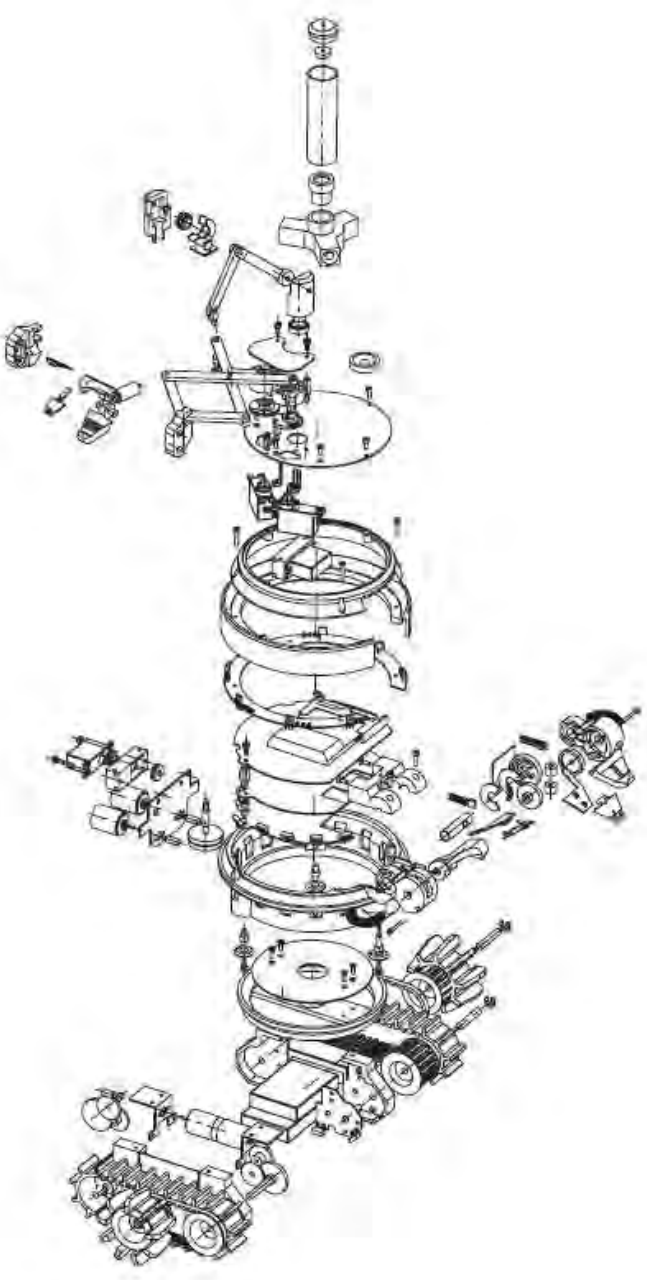


Figure 3.2: Mechanical structure of an s-bot. Each s-bot is composed of about 100 major parts.

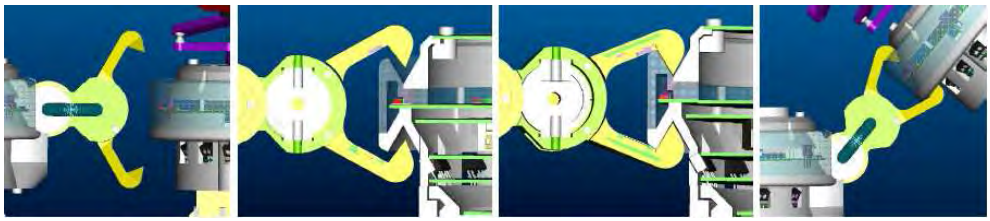


Figure 3.3: *S-bot* gripper. The mechanism by which one s-bot connects to another.

3.4 Sensing

Each s-bot is a fully autonomous mobile robot. 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 inclinometer, microphones, two humidity sensors as well as incremental encoders and torque sensors on each of the nine degree of freedom.

The 15 infra-red proximity sensors distributed around the s-bot body allow for the detection of obstacles. Ground facing proximity sensors under the tracks allow the s-bot to detect whether or not it is over a hole. Other sensors provide the s-bot with proprioceptive information about its internal motors. This includes positional information (e.g., of the rotating turret) and torque information (e.g., of forces acting on the traction system). This latter torque information is used in Chapter 6 to determine if an s-bot is successfully dragging an object. Inside the gripper there is an optical light barrier that can detect the presence of objects to be grasped. This sensor is used in conjunction with the gripper actuation described above to enable autonomous self-assembly.

In the remainder of this section we give a more detailed description of some of the sensory systems that we use in this thesis, and give brief descriptions of the low level algorithms used to process the sensory data.

3.4.1 Image Processing with the Camera

In this section we describe how the controller processes and uses the images received from the camera. All of the distributed behaviours presented in this thesis rely on local communication using a combination of the camera and the LED ring. With this combination, an s-bot can communicate its presence and its internal state to other nearby s-bots. In Chapter 9 we describe a protocol that uses the camera / LED combination to enable low bandwidth local symbolic communication.

The s-bot has omni-directional vision, achieved using a camera that points upwards at a hemispherical mirror. A transparent perspex tube holds the mirror in place. The camera records the panoramic images reflected in the hemispherical mirror. Depending on light conditions, the camera can detect s-bot LEDs up to 50 cm away, and a more powerful target light source up to 4 m away. When the s-bots operate on flat terrain, the distance in pixels from the center of an image to a perceived object can be used to estimate the physical distance between the robot and the object (see Figure 3.4).

Like most sensors on mobile robots, the readings from the camera are subject to a significant amount of noise. Objects are not always perceived even when they are in range of the camera, and due to occlusions one robot cannot see all the LEDs on another robot unless the two robots are adjacent. Furthermore, as a robot moves, objects tend to “jump”

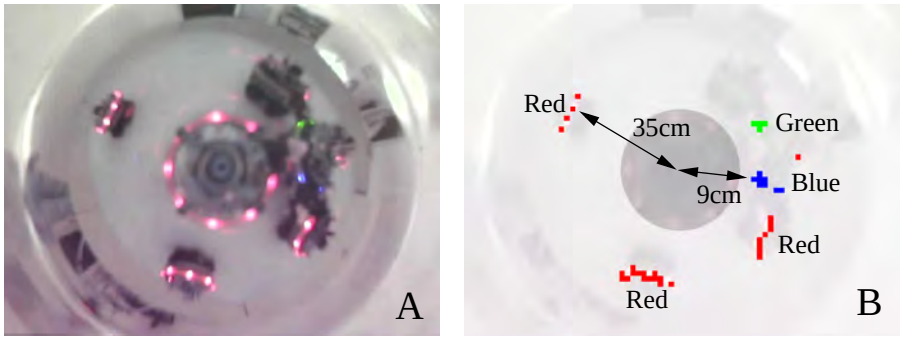


Figure 3.4: A: An example of an image captured by a robot's omni-directional camera. B: The same image after color segmentation with indications of the distance estimates from the robot that captured the image to some of the LEDs detected.

from frame to frame due to the shaking of the perspex tube housing the camera and the spherical mirror (if the camera moves even just a few millimeters between one frame and the next, it can have a considerable impact on the computed location of perceived objects). Part of the challenge in developing distributed behaviours for mobile robots is to make them robust in the presence of this type of noise.

3.4.1.1 Coloured Object Detection

The camera sensor records 640x480 pixel color images. The s-bots have sufficient on-board processing power to scan entire images and identify objects based on color. Images are divided into a grid of multi-pixel blocks and the image processor outputs the prevalent color in each block (or indicates the absence of any relevant color). We have configured the image processor to detect the location of the colored LEDs of the s-bots and discard any other information. 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). See Figure 3.4 for an example of this coloured object detection process in action.

3.4.1.2 Target Direction Noise Filtering

In all of the behaviours presented in this thesis, the s-bots must perform phototaxis. The s-bots must, therefore, be able to determine the direction of a 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*)

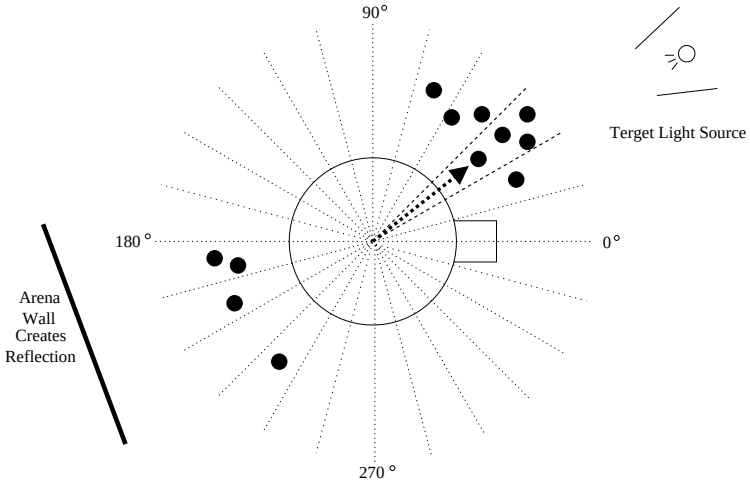


Figure 3.5: Camera based object detection and target direction filtering. The black circles represent detected yellow objects. The calculated target direction is shown by the bold arrow.

To cope with this noise, a target filtering algorithm was used (see Algorithm 1). The results of applying this algorithm are illustrated in Figure 3.5. 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 3.5 by the bold arrow.

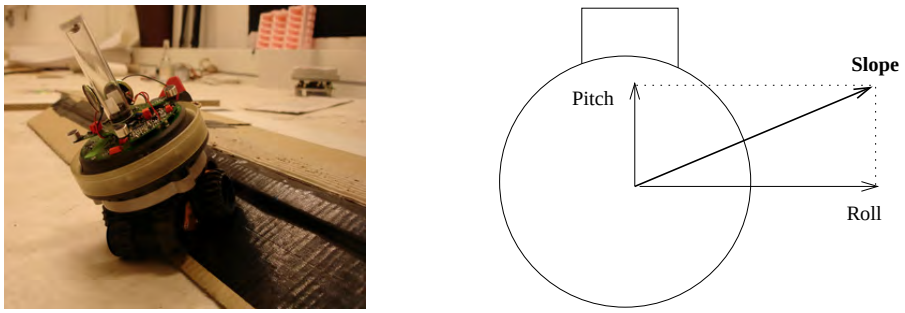


Figure 3.6: Slope direction and magnitude detection using inclinometers. Left: a real s-bot on a slope. Right: The pitch and roll inclinometer readings for the s-bot pictured on the left, and the resulting slope vector with direction and magnitude.

3.4.2 Hill Magnitude and Orientation Detection with Inclinometers

A 3-axes inclinometer provides information on the s-bot's inclination that can be used to detect if the s-bot is in danger of falling. The inclinometer returns two values - one for pitch and one for roll. By combining these values, the s-bot can determine the direction and magnitude of the slope it is on. An example of such a calculation is shown in Figure 3.6.

3.5 Behaviour Development Environment

The behaviours in this thesis were developed using a C++ API to access the s-bot sensors and actuators.

3.5.1 Common Control Interface

We used a novel *common control interface* (developed over the course of this thesis) that allowed a single behavioural controller to be compiled for execution on the real robotic platform or in simulation without any modification of source code. This common control interface allowed a rapid development model, where behavioural controllers could be prototyped in simulation and executed on the real robots, modified again in simulation and so forth.

3.5.2 Twodee Simulator

While most of the experimentation in this thesis was conducted on real robots, Chapters 7 – 10 also include experiments conducted in simulation.

The simulations in this study were conducted in a virtual environment consisting of a specialized software simulator with a custom dynamics engine tailored to our robotic platform. The simulator we used is called TwoDee [18]. TwoDee is a fast, specialized multi-robot simulator for the swarm-bots robotic platform. It has a custom rigid body physics engine, specialized to simulate only the dynamics in environments containing flat terrain, walls and holes. This restriction allows for certain optimizations and thereby reduces the computational resources necessary for running simulations significantly. TwoDee is written in C++. All the sensors and actuators that were used are simulated with reasonable accuracy by our simulation environment.

In TwoDee, particular care was taken to accurately model the camera and the proximity sensors. A sampling technique was employed using samples from the corresponding devices recorded from the real robot. These samples are collected in a set of mapped values that can afterwards be used in the simulation to determine the appropriate sensor activation for a given situation.

3.5.3 Behavioural Control Principles

Our use of the swarm-bot platform for all of the experimentation conducted in this thesis is consistent with the following behavioural control principles:

- each s-bot is autonomous in power, perception, control and action,
- each s-bot has no a priori knowledge of its environment or of its initial position and orientation,
- at the start of each experiment, an identical controller is copied onto each of the s-bots and executed on each of the s-bots independently,
- communication (when used) is visual and strictly local—the s-bots illuminate their LED rings with different colours to advertise their relative location and to provide indications of their internal state to other s-bots within visual range.

Chapter 4

Decisional Autonomy

A self-assembling system is by definition capable of autonomously forming connections between its constituent agents. Such autonomous connection forming has already been studied in the self-assembly literature. However, the problem of decisional autonomy has been largely ignored. In the context of a self-assembling system, decisional autonomy refers to the autonomous ability to choose when, if and how the self-assembly process should be triggered. The lack of research to date in this area is a serious omission, as self-assembling robots are an inappropriate choice to solve any task that does not require decisional autonomy. In scenarios where no decisional autonomy is required, the size, number and shape of required robotic entities are known in advance. Self-assembly is not appropriate in such circumstances, because it has an associated cost — to enable autonomous self-assembly, the robotic components need to be more sophisticated, and the process itself requires time, energy and computational resources. This cost renders more conventional robotic systems, either monolithic or pre-assembled, more efficient in environments whose parameters are known a priori and where no decisional autonomy is required.

Decisional autonomy is, therefore, crucial if self-assembling systems are ever to justify their use over more conventional systems. In particular, robotic entities that self-assemble make sense in situations where a priori knowledge of the environment is limited, and self-assembly is used as an autonomous response mechanism — a way for the system to flexibly respond to different environmental contingencies. The decision making mechanism presented in this chapter presents a first step towards this kind of decisional autonomy. In this chapter, we present research conducted to develop and test distributed behavioural control enabling a group of robots to classify their environment and make a corresponding decision about whether or not to self-assemble. The task we consider is a hill crossing task, parametrised by the steepness of the hill. A simple hill can be crossed individually, while a steep hill requires the robots to self-assemble and navigate over the hill as a composite entity. We present results of experiments conducted with real robots confirming the efficacy of the distributed response mechanism.

This chapter is organised as follows. In Section 4.1 we present the experimental set-up and the task. In Section 4.2 we present the distributed behavioural control we use to solve the task. In Sections 4.3 and 4.4, we present experiments conducted with real robots, and analyse the efficacy of the system. Finally, in Section 4.5, we discuss the contribution of this chapter and how it leads into the research presented in subsequent chapters.

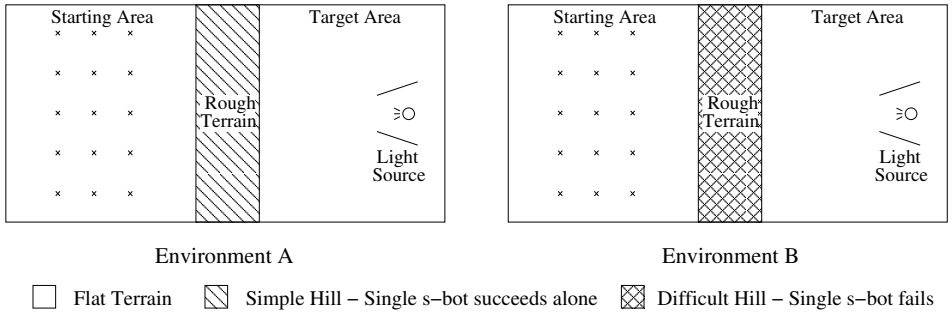


Figure 4.1: Scale diagram of the two experimental environments (view from above). Initially the s-bots are placed in the starting area (candidate positions marked by crosses). To complete the task the s-bots must enter the target area. In Environment A (left figure) the s-bots are capable of accomplishing the task independently. In Environment B (right figure) it is not possible for the s-bots to complete the task unless they self-assemble.

4.1 Experimental Setup

4.1.1 The Environment

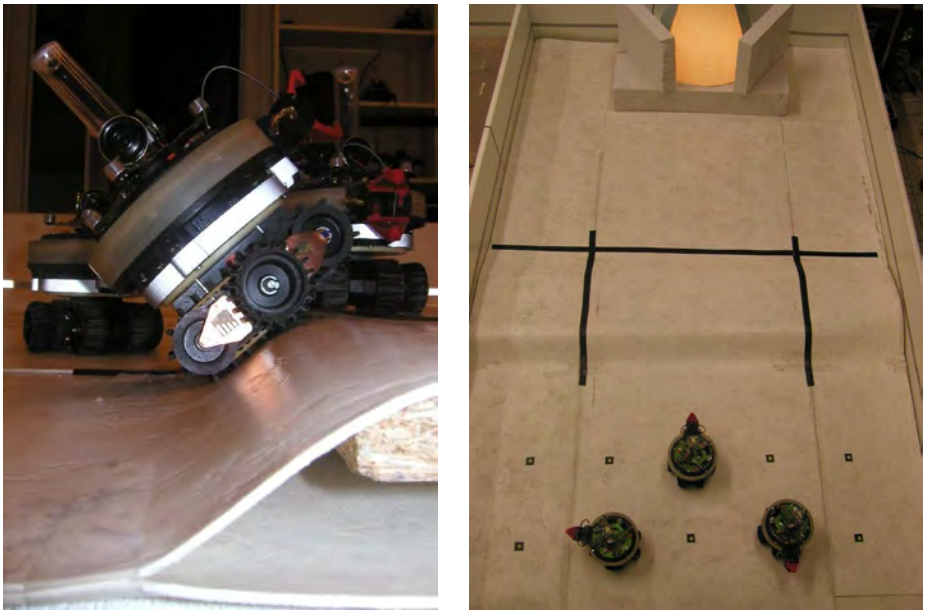
We conduct experiments in two different arenas, referred to as *Environment A* and *Environment B*. Both have dimensions of 240 cm x 120 cm and consist of three distinct areas: two areas of flat terrain (a starting area and a target area) separated by an area of rough terrain (see Figure 4.1). Environments 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. In Environment B the rough terrain consists of a hill 5 cm high which a single s-bot cannot overcome alone. Figure 4.2a 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.

4.1.2 The Task

At the beginning of each trial, the s-bots are positioned in the starting area. The initial position of each s-bot is assigned randomly by uniformly sampling without replacement from a set of 15 specific starting points. Each s-bot's initial orientation is chosen randomly from a set of 4 specific directions. Figure 4.2b shows s-bots in a random initial configuration. To complete the task the s-bots must reach the target area without toppling over.

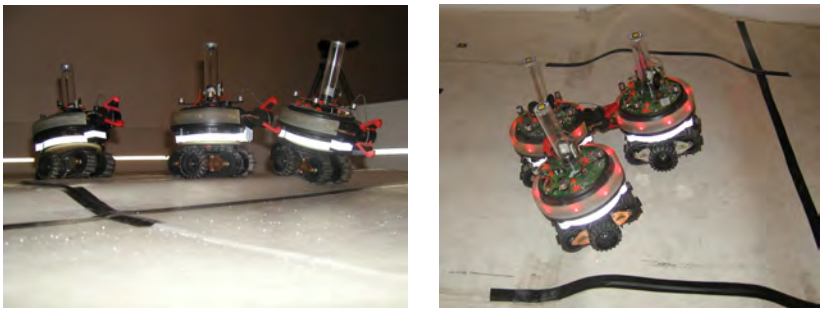
The s-bots have no a priori knowledge of the environment in which the trial takes place. To complete the task correctly, they must classify the environment they are in and act accordingly. In Environment A, the s-bots should carry out the task individually, as it is unnecessary and inefficient for the s-bots to aggregate and self-assemble. In Environment B, the s-bots must aggregate, self-assemble and coordinate their movements over the rough terrain to succeed.

Figure 4.3 shows the task being executed successfully with 3 s-bots in Environment A and Environment B. In Figure 4.3a (left figure) the s-bots are in Environment A and are navigating independently over the hill to the target light source. In Figure 4.3b (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.



(a) (b)

Figure 4.2: (a) Cross section of Environment B hill. The s-bot in the foreground is about to topple backwards. (b) Environment B from above. S-bots in random initial positions and orientations.



(a) (b)

Figure 4.3: 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 as a composite entity.

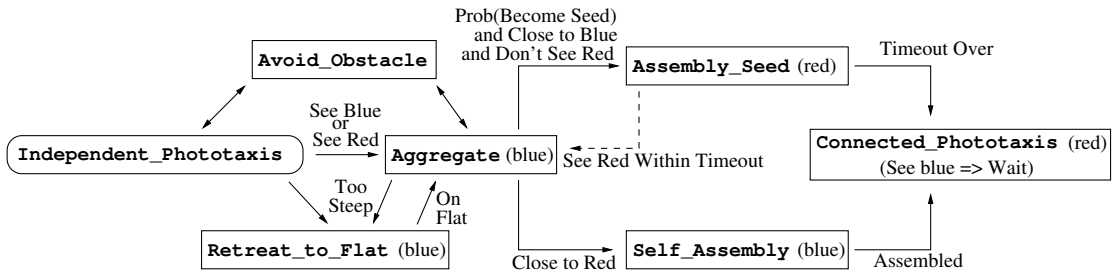


Figure 4.4: Distributed Behavioural Control: Finite State Machine

4.2 Distributed Behavioural Control

Each s-bot is fully autonomous. The same finite state machine behavioural controller is executed on all of the s-bots. The control flow is illustrated in Figure 4.4. Limited local communication between the s-bots is implemented using the camera and LEDs. Some behavioural states 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 4.1.

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 steep hill that it cannot pass alone or if it detects either a blue or a red object. The presence of a blue or a red object implies the presence of another s-bot that is either retreating from the hill, aggregating or assembling. The assumption is that if another s-bot is executing one of these behaviours, it must already be aware of the presence of a steep hill. The self-assembly process is triggered when one aggregating robot probabilistically becomes the seed for the assembly (turns red). As soon as an aggregating s-bot sees an assembled robot or the assembly seed (i.e., detects a red object), it attempts to connect to the assembled robot, and having done so turns red itself. The assembled s-bots start navigating collectively to the target area once they can no longer detect any blue objects (s-bots that are still aggregating or trying to assemble).

Behaviour arbitration is handled by an independent module (not represented in Figure 4.4). 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.

Algorithm 2 The Behaviour Arbitration Module

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

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 `Connected_Phototaxis` behaviour

Table 4.1: Possible interpretation of camera detected colour objects.

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 4.2: Value constants used in s-bot behavioural control. Generated manually through trial and error optimisation.

`Independent_Phototaxis` is the starting behaviour for each s-bot. The control logic for this behaviour is presented in Algorithm 3. 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 s-bot inclinometers). This is to prevent the s-bot toppling before `Retreat_to_Flat` behaviour has time to be initiated. 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 4.2 shows the exact values used for these constants, as well as for constants used in other behaviours.

`Avoid_Obstacle` behaviour is initiated from `Independent_Phototaxis` behaviour or `Aggregate` behaviour when the readings from the s-bot's 14 proximity sensors exceed a certain threshold. The control logic for `Avoid_Obstacle` behaviour is presented in Algorithm 4. 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

Algorithm 3 `Independent_Phototaxis` Behaviour

```
1: slope ← getSlope( )
2: if slope > MAX-SLOPE then
3:    switchBehaviour( Retreat_To_Flat )
4: else if detectColourObject( BLUE ) or detectColourObject( RED ) then
5:    switchBehaviour( Aggregate )
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
```

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 4.1 and 4.2. 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 away from the obstacle.

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
```

$$ObstacleVector_x = \sum_{prox=1}^{14} -\cos\left(Direction_{prox}\right) * Magnitude_{prox} \tag{4.1}$$

$$ObstacleVector_y = \sum_{prox=1}^{14} -\sin\left(Direction_{prox}\right) * Magnitude_{prox} \tag{4.2}$$

`Retreat_to_Flat` behaviour is initiated from `Independent_Phototaxis` behaviour or `Aggregate` behaviour when information from the s-bot's inclinometers indicate that the s-bot is in danger of toppling over. The control logic for `Retreat_to_Flat` behaviour is presented in Algorithm 5. The s-bot receives information about its inclination in two planes (pitch and roll). 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 4.2). 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. 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

```
activateColourRing( BLUE )
loop
    pitch ← getFrontBackInclination()
    roll ← getLeftRightInclination()
    totalInclination ← pitch + roll
    if totalInclination > MAX-SLOPE then
        downHillVect ← calculateDownHillVector( pitch, roll )
        setTracksMoveDirection( downHillVect )
    else
        setTracksMoveDirection( downHillVect, RETREAT-TIMEOUT )
        switchBehaviour( Aggregate )
    end if
end loop
```

`Aggregate` behaviour is initiated from `Retreat_to_Flat` behaviour (once flat ground has been reached) or from `Independent_Phototaxis` behaviour (if the robot sees any blue or red objects, that is, other s-bots that are already aggregating or assembling). 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 `Connected_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

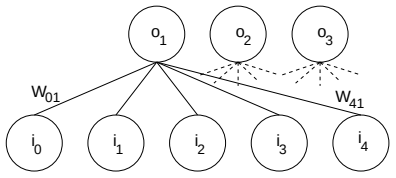
```
activateColourRing( BLUE )
loop
    if detectColourObject( RED ) then
        Close to Red → switchBehaviour( Self_Assembly )
        Far from Red → approachRed( )
    else if detectColourObject( BLUE ) then
        Prob( BECOME-SEED )→ switchBehaviour( Assembly_Seed )
        Prob( 1 - BECOME-SEED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

`Self-Assembly behaviour` is initiated from `Aggregate` behaviour once the s-bot sees a red object (i.e., an assembled robot or the assembly seed). The control logic for the `Self_Assembly` behaviour is presented in Algorithm 7. Function f maps sensory input

Algorithm 7 `Self_Assembly` Behaviour

```
activateColourRing( BLUE )
loop
    (i1, i2) ← featureExtraction(camera)
    (i3, i4) ← sensorReadings(proximity)
    (o1, o2, o3) ← f(i1, i2, i3, i4)

    if graspingRequirementsFulfilled( o3 ) then
        grasp( )
        if successfullyConnected( ) then
            switchBehaviour( Connected_Phototaxis )
        else
            openGripper( )
        end if
    end if
    applyValuesToTracks( o1, o2 )
end loop
```



(a)

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

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

(b)

Figure 4.5: (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.

to motor commands. It is implemented by a neural network which has been designed by artificial evolution in previous research not described in this thesis. As part of this previous research, the `Self_Assembly` behaviour was extensively tested with swarms of up to 16 physical s-bots. For details of these experiments and of the underlying neural network, see [62].

`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 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 `Connected_Phototaxis` behaviour. The control logic for the `Assembly_Seed` 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( Connected_Phototaxis )
9:   end if
10: end loop
```

`Connected_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 the `Connected_Phototaxis` behaviour is presented in Algorithm 9. Connected phototaxis is more complicated than the phototaxis required in `Independent_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.

In the `Connected_Phototaxis` behaviour, 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 4.6. Note that this control mechanism is discontinu-

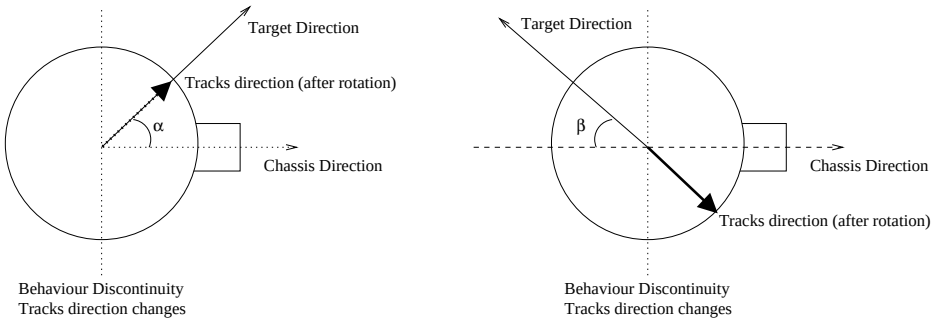


Figure 4.6: S-bot turret/chassis rotation based on target direction (view from above). (Left) S-bot rotates turret α degrees clockwise. Tracks going forwards. (Right) S-bot rotates turret β degrees anticlockwise. Tracks in reverse.

ous when the target direction is close to 90 degrees away from the chassis heading. This discontinuity is indicated with a dotted line in Figure 4.6. 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 vice-versa). At the same time it reverses the rotation of the tracks.[1]

Algorithm 9 `Connected_Phototaxis` behaviour

```
activateColourRing( RED )
loop
  if detectColourObject( BLUE ) then
    setTrackSpeeds( 0, 0 )
  else
    turretTargDirn ← getTargetDirection( )
    rotation ← getTurretRotation( )
    chassisTargDirn ← getRelativeDirection( turretTargDirn, rotation )
    newRotation ← calculateRotation( targDirn )
    rotateTurret( newRotation )
    setTracksMoveDirection( chassisTargDirn )
  end if
end loop
```

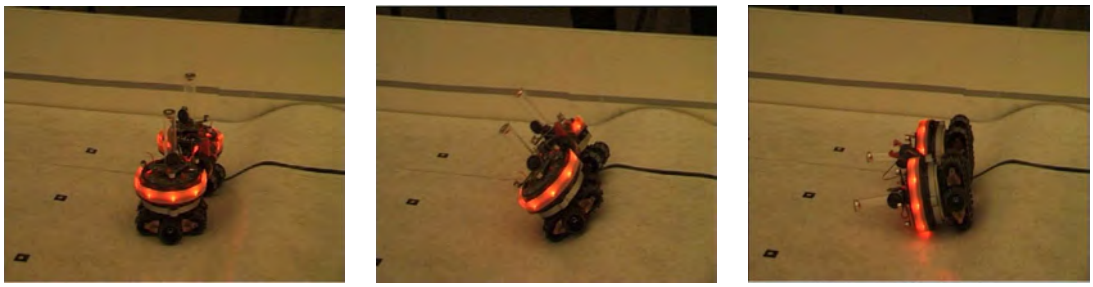
4.3 Results

We conducted a series of experiments in two different environments (see Figure 4.1) with groups of 1, 2 and 3 s-bots.

4.3.1 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

[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.



(a) (b) (c)

Figure 4.7: 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.

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.[2]

4.3.2 Trials with a single s-bot in Environment B

We modified the controller so that the s-bot was prevented from switching out of `Independent_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.[3]

4.3.3 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 the swarm-bot succeeded in overcoming the hill. In the other 7 trials 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 4.7 shows a sequence of photos from one of these unsuccessful trials.

4.3.4 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 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 the s-bots still managed in each case to self-assemble into a swarm-bot of 2 s-bots. In 2 of these 4 trials the 2 s-bot swarm-bot went on to successfully reach the target area. In the 2 other trials the 2 s-bot swarm-bot was obstructed by the third s-bot

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

[3]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.

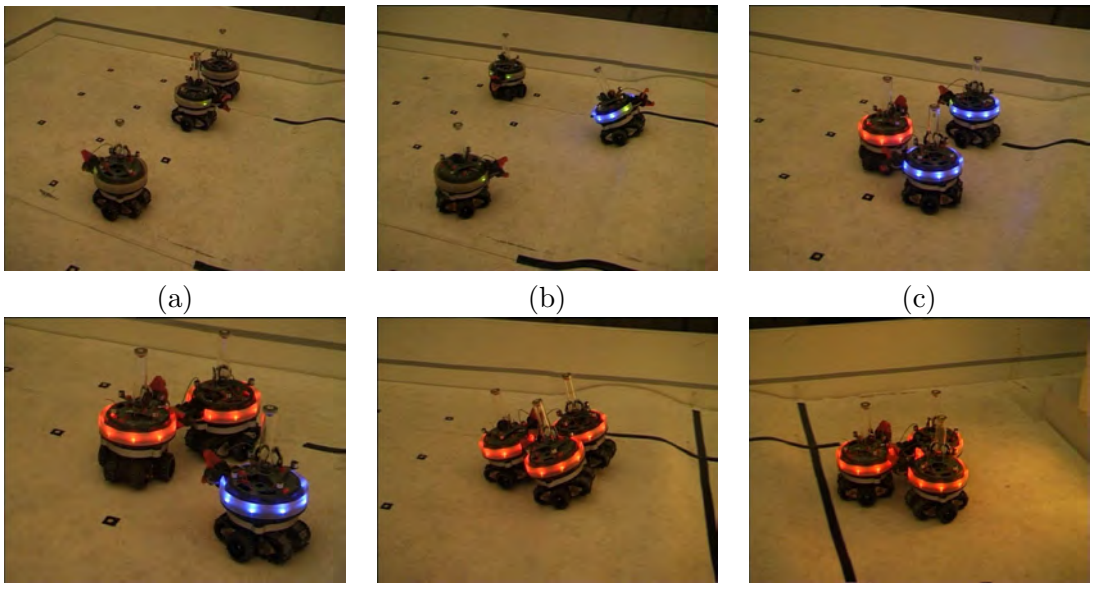


Figure 4.8: (a) The s-bots start in a random configuration. (b) A single s-bot detects a slope it cannot overcome alone and turns blue (c) Other s-bots detect blue colour, turn blue themselves and aggregate. One s-bot then seeds the assembly process by turning red. (d) One s-bot has assembled to the 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.

which failed to self-assemble. Figure 4.8 shows a sequence of images taken from a typical successful trial.

4.4 Analysis

4.4.1 Success Rate

Table 4.3 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.[4]

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 (26 s-bots out of 40) complete the task. The 3 s-bot trials show a further clear improvement — 87% (52 s-bots out of 60) complete the task.

The fourth row (% successful in a 3 s-bot swarm-bot) shows that in the 3-s-bot trials 80% of s-bots (48 s-bots out of 60) successfully self-assembled into a 3 s-bot swarm-bot. The same row shows us that the same 80% of s-bots also completed the task in a 3 s-bot

[4]For example in the 3 s-bot trials 7% of s-bots (4 s-bots out of 60) completed the task as part of a 2 s-bot swarm-bot.

Table 4.3: 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 #trials: 20 #s-bots: 20		2 s-bot trials #trials: 20 #s-bots: 40		3 s-bot trials #trials: 20 #s-bots: 60	
	A	**C**	**A**	**C**	**A**	**C**
% Successful (total)	N/A	0	100	65	93	86
% Successful alone	N/A	0	N/A	0	N/A	0
% Successful in 2 s-bot swarm-bot	N/A	N/A	100	65	13	7
% Successful in 3 s-bot swarm-bot	N/A	N/A	N/A	N/A	80	80
% Failed	N/A	100	0	35	7	13

swarm-bot. Thus in 3 s-bot trials, whenever all the 3 s-bots successfully self-assembled into a 3 s-bot swarm-bot they always went on to successfully overcome the rough terrain. By contrast, in the 2 s-bot trials 100% of the s-bots (40 s-bots out of 40) self-assembled into a 2 s-bot swarm-bot. Despite this only 65% of the s-bots (26 s-bots out of 40) successfully overcame 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 4.7). 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.

4.4.2 Timing Analysis of 2-s-bot Trials in Environment B

We have identified five phases of task execution for the 2 s-bot trials in Environment B. Figure 4.9 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.[5] The five phases of task execution are discussed individually below.

- **Independent Phototaxis**. This initial phase is represented in Figure 4.9 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 `Independent_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

[5]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.

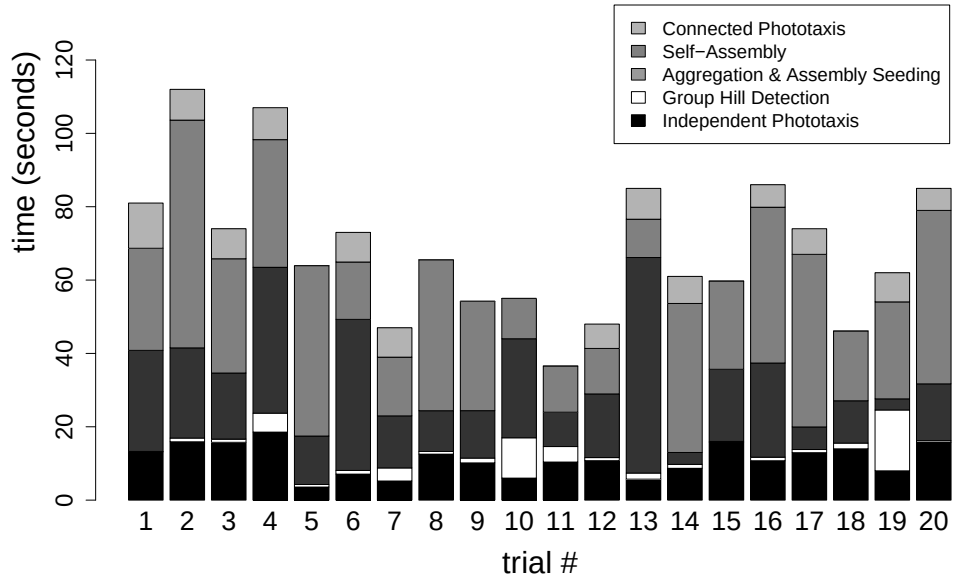


Figure 4.9: Timing analysis of 2 s-bot trials in Environment B (for further explanation see text).

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 4.9 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.

- **Aggregation & Assembly Seeding**. This phase is represented in Figure 4.9 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 4.9 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 switched to `Connected_Phototaxis` behaviour. This phase takes between 10.4 s and 62.1 s.

- **Connected Phototaxis**. This phase is represented in Figure 4.9 by the light grey segment. In this phase, 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) the last phase (Connected Phototaxis) is not shown in Figure 4.9. 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.

4.4.3 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 4.10 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 4.10 we instead mark these moments with a ‘C’ and an ‘S’ respectively.

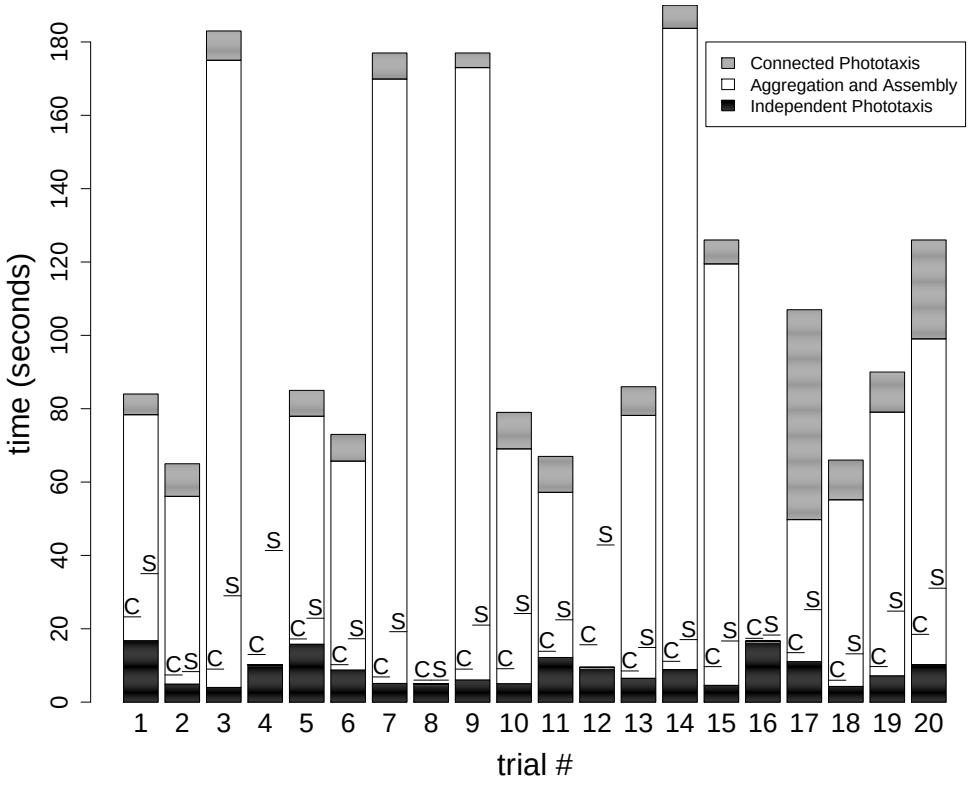


Figure 4.10: Timing analysis of 3 s-bot trials in Environment B (for further explanation see text).

- **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.

- **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 `Connected_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.

- **Connected 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 4.10 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 4.8. In other trials (e.g., trials 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 `Independent_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.[6] 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.

The 'S' in Figure 4.10 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.

4.4.4 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

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

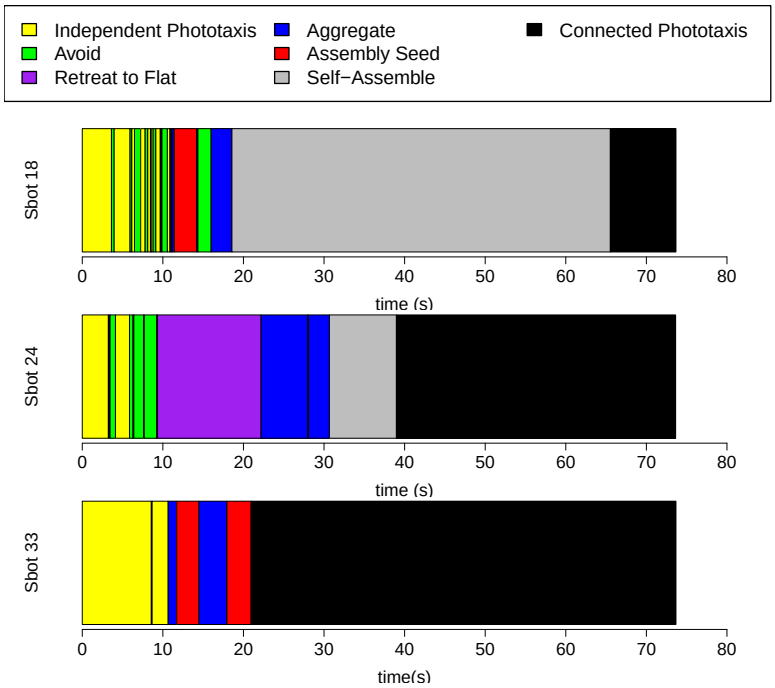


Figure 4.11: Behavioural Analysis of 3 s-bot trial 16.

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 4.11 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 with Sections 4.4.2 and 4.4.3 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 `Independent_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 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 `Connected_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 remain stationary 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 phototaxis to the target. The task is completed at 73.6 s.

4.5 Discussion and Conclusions

The work presented in this chapter contributed to self-assembly research by providing the first demonstration of decisional autonomy over the triggering of the self-assembly process in a real-world robotic system. More precisely, in this chapter, we presented distributed behavioural control to enable a robotic system to display a simple autonomous self-assembly response mechanism. We conducted experiments to show that this response mechanism could successfully distinguish between different environments, and complete the given task with a high success rate, particularly when 3 s-bots were used. However our approach as presented so far has several limitations:

1. The distributed behavioural control is specific to this experimental set-up.
2. Once self-assembled, the composite robotic entity does not modify its behaviour based on the new composite morphology — each constituent s-bot pursues the goal of phototaxis 'greedily', co-operation being limited to the minimum necessary to allow the composite entity to move.
3. We established that autonomous self-assembly response mechanism is possible, but not whether it actually enhances the efficiency of the system (as opposed, for example to just always pre-emptively self-assembling independently of the environment).
4. There is no active control over the morphology that is generated through self-assembly in response to environmental contingencies.

In the next chapter, we address limitations 1-3, above. We conduct experiments in an environment in which the orientation of the hill can change, and show how the composite robotic entity can modify its group-level behaviour in response to different hill orientations. We compare results of autonomous self-assembly response against other 'strategies' such as pre-emptive self-assembly. This allows us to isolate and analyse performance benefits of the self-assembly response mechanism. We show how the self-assembly response mechanism can be used in other problem scenarios. Limitation 4 above is addressed in Chapters 7, 8 and 9.

Chapter 5

The Value of Self-Assembly

In this chapter, we present research conducted to analyse the contribution of self-assembly to system performance. Existing self-assembly research has largely ignored the issue of when and if self-assembly capabilities are beneficial. To facilitate our analysis, we introduce the concept of a *self-assembly strategy*. A self-assembly strategy is a generic group level behaviour that corresponds to a particular degree of autonomy over the self-assembly process and the deployment of the resulting collective robotic entity. In this chapter, we present a more rigorous experimental set-up for the hill climbing task, and show how generic self-assembly strategies map to specific distributed behavioural control for this task. We conduct real-world experiments with the hill-crossing task to isolate and analyse the performance benefits of self-assembly to the system by comparing the performance of different strategies. We analyse both the costs (overheads) and benefits (improvements in efficiency) of the different strategies, and discuss the conditions under which autonomous self-assembly is an appropriate solution to a task.

This chapter is organised as follows. In Section 5.1, we present the experimental set-up. In Section 5.2, we present the *basic self-assembly response* strategy. This strategy allows the system to trigger the self-assembly process when individual robots prove incapable of solving a task independently. This strategy is a refined version of the behaviour presented in Chapter 4. In Section 5.3, we use the *basic self-assembly response* strategy to analyse the benefits of acting as a physically larger connected entity. In Section 5.4, we use the *basic self-assembly response* strategy to analyse the benefits of decisional autonomy. In Section 5.5, we present the *connected coordination* strategy. This strategy coordinates the sensing and actuation of the assembled robots to allow them to respond to the environment as a single collective entity. In Section 5.6, we analyse the costs and benefits of the *connected coordination* strategy. In Section 5.7, we consider the scalability of our behavioural control. In Section 5.8, we demonstrate that our generic strategies can indeed be applied in other problem domains by applying the *basic self-assembly response* strategy to a hole crossing task. In Section 5.9, we discuss the contribution of this chapter and present our conclusions.

5.1 Experimental Setup

The hill crossing task requires a group of between one and three s-bots to navigate over a priori unknown terrain to a *target area* containing a light source. The s-bots are considered to have completed the task if they reach the target area without toppling over.

We conduct experiments in seven different environments (see Figure 5.1). Six of the

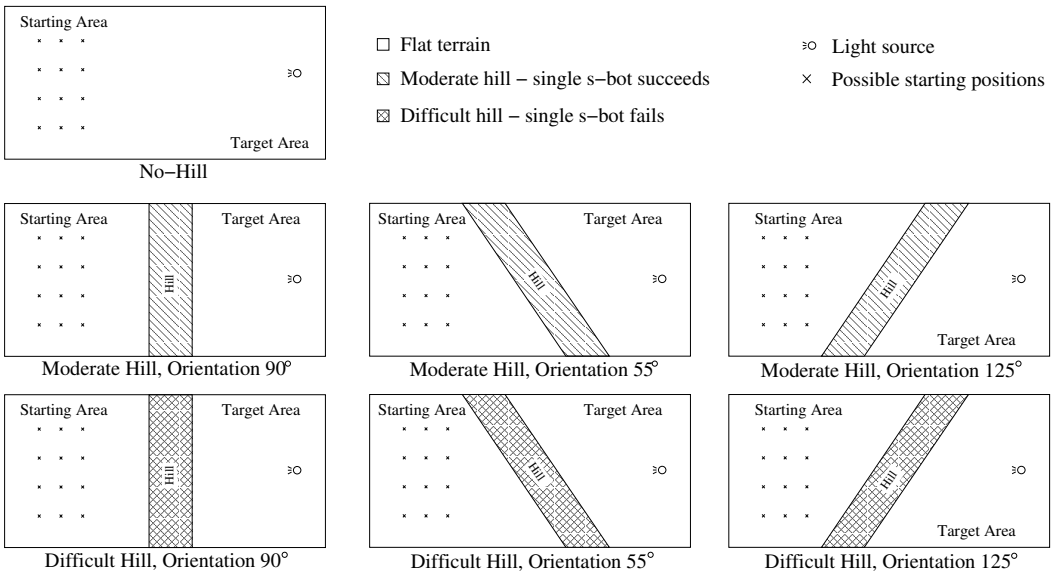


Figure 5.1: Scale diagram of the seven environments used in this study. Each environment measures 210 cm x 105 cm. Three environments contain the 'moderate' hill (2.8 cm high, navigable by a single s-bot). Three environments contain the 'difficult' hill (6.5 cm high, not navigable by a single s-bot). One environment has no hill. Starting positions are marked by crosses.

seven environments contain a hill. We use two types of hill—a 'moderate' hill and a 'difficult' hill. The moderate hill can be overcome by a single s-bot. S-bots can only navigate the difficult hill as part of a larger self-assembled entity (the steepness of the hill would cause a single s-bot to topple). Both the moderate hill and the difficult hill can be oriented in three different ways.

The hill and the arena boundaries together demarcate the *starting area* and the target area. In the no-hill environment, the starting area and target area are considered to be the same as for environments with hill orientation 90° (see Figure 5.1). In each experiment, the starting positions of the participating s-bots are assigned randomly by uniformly sampling without replacement from a set of 12 possible starting points in the starting area. The initial orientations are chosen randomly from a set of 4 possible directions.

5.2 The *Basic Self-Assembly Response* Strategy

In this section, we describe the *basic self-assembly response* strategy and present the distributed behavioural control that enables the corresponding group level behaviour for our hill crossing task. The *basic self-assembly response* strategy is illustrated in Figure 5.2. The robots start by trying to execute a task independently. If they fail to complete the task independently, they self-assemble and attempt to solve the task as a larger composite robotic entity.

5.2.1 Strategy Implementation for the Hill Crossing Task

In our implementation, the s-bots start to navigate independently towards the target light source. In the absence of any steep hill, the robots complete the task independently. If, however, the robots detect a hill that is too steep to be navigated individually, they first

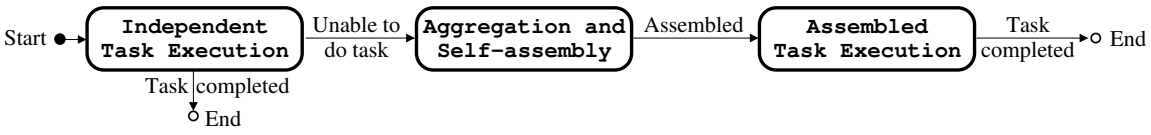


Figure 5.2: *Basic self-assembly response* strategy: group-level behaviour.

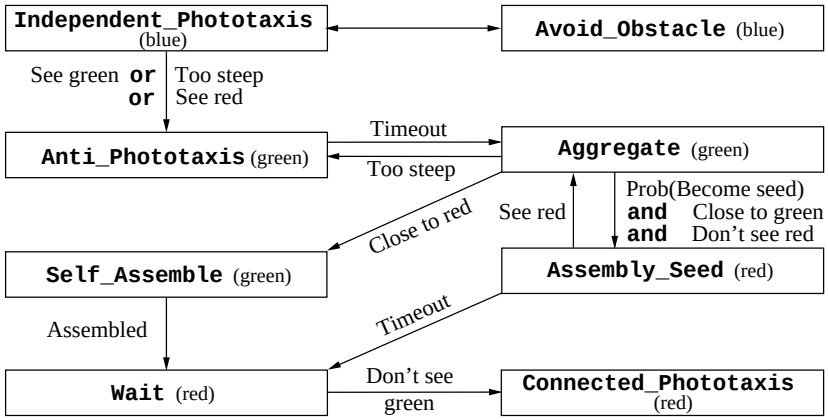


Figure 5.3: Distributed behavioural control to implement the *basic self-assembly response* strategy for the hill crossing task. This finite state machine controller is executed independently on individual robots. The starting state is `Independent_Phototaxis`. Colours in parentheses refer to the LEDs that are illuminated in the corresponding state.

aggregate, then self-assemble into a larger connected entity. The robots then navigate collectively to the target light source.

The distributed behavioural control we use to implement the strategy is shown in Figure 5.3. This behavioural control is a refined version of the behaviour presented in Chapter 4. Figure 5.4 shows a three s-bot system using the strategy. An s-bot starts by illuminating its blue LEDs and navigating independently towards the light source in the target area (state `Independent_Phototaxis`). The light source is detected with the camera. While navigating towards the light source, the s-bot uses its proximity sensors to move away from arena walls and other robots that are too close (state `Avoid_Obstacle`). If the s-bot finds itself on a hill too difficult for it to pass alone (detected using its inclinometers), or if it sees (i.e., detects with its camera) a green s-bot or a red s-bot, it illuminates its green LEDs and retreats away from the hill for a given length of time (state `Anti_Phototaxis`). It then switches into state `Aggregate`, and tries to get close to a red s-bot (the seed or an s-bot already assembled to the seed), or if no red s-bot is perceived, to search for and get close to another green (aggregating) s-bot. In the latter case, if the s-bot is sufficiently close to another green s-bot and can still see no other red s-bots, it can become, with a given probability, the seed that triggers the process of self-assembly (state `Assembly_Seed`). A seed s-bot lights up its red LEDs, and waits until a timeout has expired. If it sees another red s-bot within the timeout period, it reverts to state `Aggregate`. Otherwise, after the timeout it switches to state `Wait`. If an aggregating s-bot gets sufficiently close to a red (assembled) s-bot, then it starts self-assembling (state `Self_Assemble`). Assembled s-bots switch to state `Wait`. S-bots in state `Wait` illuminate their red LEDs. When they no longer see any green (aggregating or assembling) s-bots, they switch to state `Connected_Phototaxis` and navigate collectively to the light source.

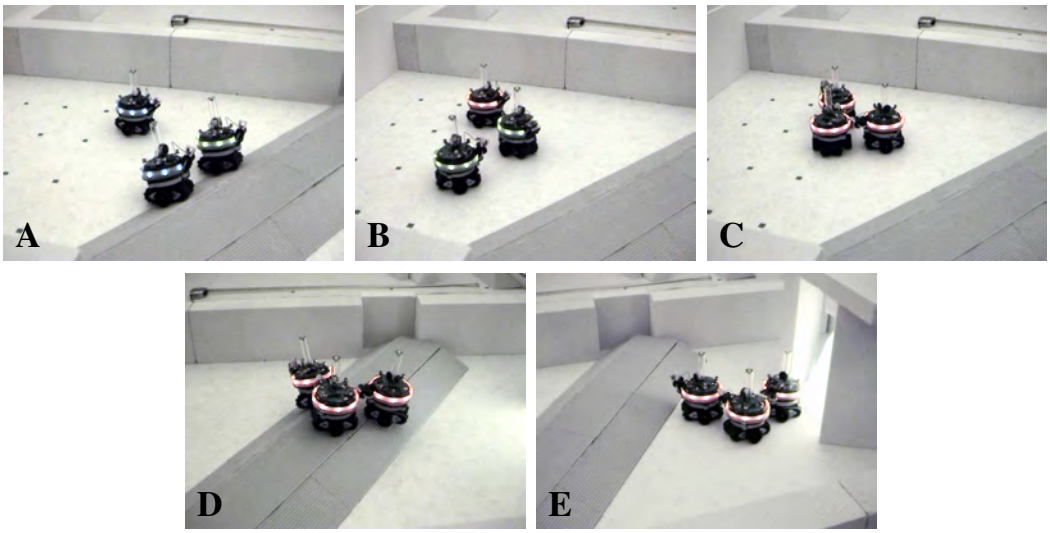


Figure 5.4: A group of s-bots using the *basic self-assembly response* strategy in an environment containing a hill that is not navigable individually. (A): Three s-bots start from random positions and orientations. Initially, they perform independent phototaxis, and have their blue LEDs illuminated. One s-bot detects a slope it cannot overcome alone and illuminates its green LEDs. (B): The other s-bots detect colour green (local communication). All of the s-bots retreat away from the light source for a fixed time period. One of the robots probabilistically becomes the seed for the self-assembly process. (C): The s-bots self-assemble. (D,E): The s-bots perform connected phototaxis towards the light source and successfully overcome the obstacle to arrive in the target area. A video of the experiment from which these snapshots were taken can be found in the online supplementary material [110].

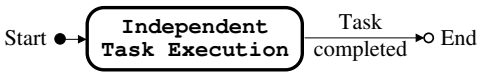


Figure 5.5: *Independent execution only* strategy: group-level behaviour.

During collective navigation, each connected s-bot performs greedy phototaxis (i.e., each s-bot heads in a direct line towards the light source). After self-assembly, however, the orientations of the constituent s-bots' turrets are fixed. Each s-bot, therefore, continually rotates its turret to compensate for the track movements that move the s-bot towards the light source [68]. For a detailed description of the individual states in this finite state machine see [110, 111].

5.3 Benefits of Scale

In this section, we describe experiments we conducted with the *basic self-assembly response* strategy to analyse the simple scale benefits of acting as a physically larger self-assembled entity. To provide a basis for comparison, we also conducted control experiments with a simple strategy that causes the s-bots to carry out the task independently, irrespective of environmental conditions.

5.3.1 The *Independent Execution Only* strategy

This strategy is illustrated in Figure 5.5. The distributed behavioural control to implement this strategy for the hill crossing task is a modified version of the distributed behavioural control for the *basic self-assembly response* strategy in which the transition to state `Anti_Phototaxis` is disabled (see Figure 5.3). Thus, only the states `Independent_Phototaxis` and `Avoid_Obstacle` are executed. The result is that each s-bot moves independently towards the light at a constant speed irrespective of the type of terrain it encounters.

5.3.2 Results

We conducted 60 trials with a single s-bot using the *independent execution only* strategy in the difficult hill environments (20 trials per hill orientation). The s-bot failed to overcome the hill in all 60 trials. In each trial, the s-bot reached the hill and then toppled backwards due to the steepness of the slope. To confirm that the s-bot was failing due to the intrinsic properties of the slope, we repeated this experiment at a number of different constant speeds and observed the same result.

We conducted 60 trials with 2 s-bots using the *basic self-assembly response* strategy in the difficult hill environments (20 trials per hill orientation). Both s-bots successfully self-assembled into a two s-bot swarm-bot in every trial. In 21 trials (35%), the swarm-bot succeeded in overcoming the hill, and thus completed the task. In the other 39 trials (65%), the assembled swarm-bot failed to overcome the hill. These failures happened when the assembled swarm-bot moved towards the light source with an orientation overly parallel to the orientation of the hill and thus toppled over.

We conducted 60 trials with 3 s-bots using the *basic self-assembly response* strategy in the difficult hill environments (20 trials per hill orientation). In 47 trials (78.3%), all 3 s-bots succeeded in self-assembling and overcoming the hill. A number of different

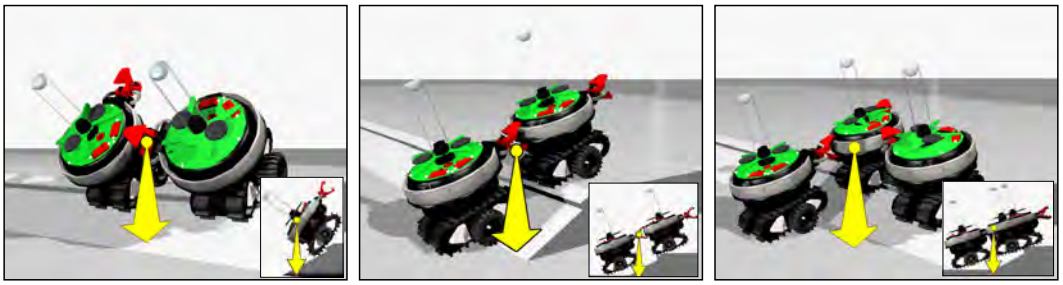


Figure 5.6: Different swarm-bot configurations and orientations on the difficult hill (to scale). Left: a linear two s-bot swarm-bot topples on the difficult hill as its centre of gravity (see yellow arrow) escapes its vertical footprint. Centre: a linear two s-bot swarm-bot with an appropriate orientation does not topple. Right: a non-linear three s-bot swarm-bot can approach with any orientation and will not topple.

types of failure occurred. In 9 trials, all three s-bots toppled over on the hill because they self-assembled into a roughly linear swarm-bot and also approached the hill with an orientation nearly parallel to that of the hill (the same type of failure that we witnessed in the 2 s-bot trials). In some trials, the self-assembly process itself failed (the addition of an extra s-bot makes the self-assembly process more complex): in 2 trials, a single s-bot failed to complete the task when only 2 out of the 3 s-bots succeeded in self-assembling into a connected entity (the 2 s-bot swarm-bot went on to complete the task). In 1 further trial, all three s-bots failed when again only 2 out of the 3 s-bots successfully self-assembled, but the 2 s-bot swarm-bot toppled on the hill. In a single trial, an s-bot that was part of the three s-bot swarm-bot detached on the hill due to a faulty grip, and thus toppled over. In total, 147 of the 180 participating robots successfully completed the task (81.7%).

The results show that navigating as a two s-bot swarm-bot instead of individually caused the task completion rate to increase from 0% to 35% (a significant increase according to the two-tailed Fisher's exact test, $p < 0.001$).[1] The addition of a third s-bot to the system produced even more stable connected swarm-bot entities, further increasing the task completion rate to 81.7% (a significant increase according to the two-tailed Fisher's exact test, $p < 0.001$).

The overall task completion rate was still far from optimal—35% and 81.7% for two s-bot and three s-bot experiments respectively (percentages of individual s-bots that succeed in overcoming the hill). Inappropriate orientation of the swarm-bot caused 100% and 81.8% of failures with two s-bots and three s-bots respectively. Inappropriate orientation occurs when the stochastic self-assembly process produces a swarm-bot with an approximately linear morphology, and when the resulting random orientation of such a linear swarm-bot is overly parallel to the orientation of the hill (see Figure 5.6). Using the *basic self-assembly response* strategy, a swarm-bot attempts to overcome the hill with its initial random orientation. For a strategy which resolves this problem see Section 5.5.

[1] To minimise damage to the robots caused by toppling over, we used just a single s-bot for the trials of the *independent execution only* strategy. We make the assumption that the task completion rate of an s-bot in a one robot trial is identical to the task completion rate of an s-bot in a two robot trial if the two s-bots are navigating independently. Note that this assumption does not take into account accidental interference between independently navigating s-bots—it is for example possible (although unlikely) that one s-bot could prevent another from toppling over by coincidentally giving it a push at the right moment.

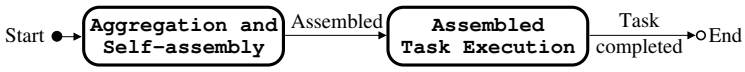


Figure 5.7: *Preemptive self-assembly* strategy: group-level behaviour.

5.4 Benefits of Decisional Autonomy

In this section, we describe experiments we conducted with the *basic self-assembly response* strategy to analyse the performance benefits of giving a system decisional autonomy over when and if to self-assemble. To provide a basis for comparison, we also conducted control experiments with a simple strategy that causes the s-bots to aggregate and self-assemble irrespective of their environment.

5.4.1 The *Preemptive Self-Assembly* Strategy

The strategy is illustrated in Figure 5.7. The distributed behavioural control to implement this strategy for the hill crossing task is a modified version of the distributed behavioural control for the *basic self-assembly response* strategy in which the starting state is `Aggregate` instead of `Independent_Phototaxis` (see Figure 5.3). The result is that the s-bots aggregate and self-assemble irrespective of the environment. The connected swarm-bot entity then performs connected phototaxis to the light source in the target area.

5.4.2 Results: Validation of the Response Mechanism

In this subsection, we describe a number of trials that we conducted using the *basic self-assembly response* strategy. In the difficult hill environments, we conducted 60 trials with two s-bots and 60 trials with three s-bots (these experiments have already been discussed in Section 5.3.2). In every trial, all of the s-bots successfully detected the presence of the slope in every trial, thus triggering the self-assembly process.

In the moderate hill environments, we again conducted 60 trials with two s-bots (20 trials per hill orientation) and 60 trials with three s-bots (20 trials per hill orientation). Finally, in the no-hill environment, we conducted 20 trials with two s-bots. In the $60 + 60 + 20 = 140$ trials in the moderate hill and no-hill environments, all of the s-bots always correctly 'chose' not to self-assemble and to navigate to the target area individually. Thus, in all 260 trials, the group-level response mechanism correctly classified the environment and provoked the appropriate system response (self-assembly or no self-assembly).

5.4.3 Results: Benefits of Decisional Autonomy

We conducted 20 trials with two s-bots using the *preemptive self-assembly* strategy in the no-hill environment. In all 20 trials, both s-bots successfully completed the task. This provides a baseline against which we compare the *basic self-assembly response* strategy. Throughout this section, we assume that the mean completion time of the *preemptive self-assembly* strategy in the no-hill environment is a lower bound for the mean completion time of the same strategy in moderate hill or difficult hill environments.

Figure 5.8 (left) shows a box-and-whisker plot of completion times for the *preemptive self-assembly* strategy in the no-hill environment and the *basic self-assembly response* strategy in the no-hill, moderate hill and difficult hill environments (successful trials only).

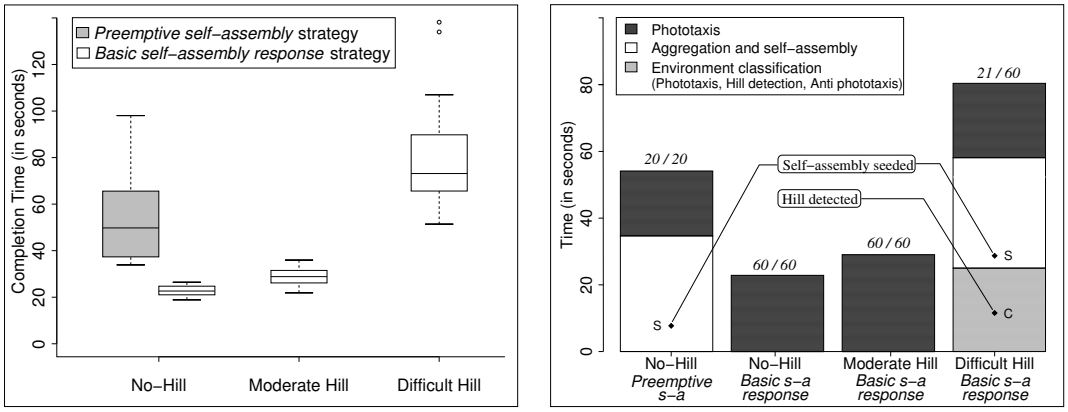


Figure 5.8: Left: Box-and-whisker plot of completion times for two s-bots using the *preemptive self-assembly* strategy (grey box) and the *basic self-assembly response* strategy (white boxes). Each box comprises observations ranging from the first to the third quartile. The median is indicated by a horizontal bar, dividing the box into the upper and lower part. The whiskers extend to the farthest data points that are within 1.5 times the interquartile range. Outlier data points are represented by circles. Right: Break-down of mean completion times for two s-bots using the *preemptive self-assembly* strategy (left bar) and the *basic self-assembly response* strategy (right three bars) in no-hill, moderate hill and difficult hill environments. Only data from completed trials are presented (number of completed trials and number of trials in total are indicated above each bar).

Figure 5.8 (right) shows the mean completion times from the same experiments broken down into the different phases of task completion.

In the no-hill environment, the *basic self-assembly response* strategy performed significantly better than the *preemptive self-assembly* strategy (two-tailed Mann-Whitney, $p < 0.001$). The mean task completion times for the two s-bots were respectively 22.9 s and 54.1 s. The *basic self-assembly response* strategy took on average 57.7% less time to complete the task than the *preemptive self-assembly* strategy. Looking at the break-down of the mean completion times in Figure 5.8 (right), we can see that s-bots using the *preemptive self-assembly* strategy spent over half of their time on actions that were not necessary to complete the task (i.e., aggregation and self-assembly).

In the moderate hill environments, the mean completion time for the two s-bots using the *basic self-assembly response* strategy was 29.0 s, which is 27.1% more than the mean completion time for the same strategy in the no-hill environment. This increase is due to the extra overhead of environment classification, which takes place during phototaxis. S-bots using the *basic self-assembly response* strategy slow down on the slope to test its navigability. Nevertheless, even using the lower bound mean completion time for the *preemptive self-assembly* strategy, the *basic self-assembly response* strategy still significantly outperforms the *preemptive self-assembly* strategy (two-tailed Mann-Whitney, $p < 0.001$). In the moderate hill environments, the mean completion time for the *basic self-assembly response* strategy was 46.4% less than the lower bound mean completion time for the *preemptive self-assembly* strategy.

In the difficult hill environments, the mean completion time for the two s-bots using the *basic self-assembly response* strategy was 80.4 s. In environments where self-assembly is necessary (difficult hill environments), it is intuitively clear that the *preemptive self-assembly* strategy is more efficient than the *basic self-assembly response* strategy. S-bots

using the *basic self-assembly response* strategy have an extra overhead of environment classification consisting of initial independent phototaxis, hill detection and anti-phototaxis—see Figure 5.8 (right).

The relative frequency with which the s-bots encounter the different environments determines which of the two strategies is more efficient. If we consider a distribution of environments containing only no-hill and difficult hill environments, then we can use the mean completion times to calculate the upper bound ratio of no-hill environments encountered, α, for which the efficiency of the two strategies is identical[†]:

$$\begin{aligned} 22.9\alpha + 80.4(1-\alpha) &= 54.1 \\ \Rightarrow \alpha &\approx 0.457 \end{aligned}$$

We conclude that for two s-bots executing the hill crossing task, the *basic self-assembly response* strategy is more efficient than the *preemptive self-assembly* strategy if more than 45.7%[†] of encountered environments are no-hill environments. If, instead, we consider a distribution of environments containing only moderate hill environments and difficult hill environments, a similar analysis reveals that the *basic self-assembly response* strategy will be more efficient if at least 51.3% of the environments are moderate hill environments.

In the future, it will be essential for the designers of distributed robotic systems to consider the costs as well as the benefits of autonomous self-assembly. Calculating α for the hill crossing task tells us when the autonomous self-assembly response mechanism actually reduces efficiency. Of course, future tasks will be more complex, and future versions of the α metric will need to be correspondingly more sophisticated. However, the calculation of such metrics might allow future system designers to determine whether an autonomously self-assembling system is an appropriate choice for the task they need to solve.

5.5 The *Connected Coordination* Strategy

In this section, we present the more sophisticated *connected coordination* strategy that allows the assembled robots to coordinate their sensing and actuation so that they respond to their environment as a single collective robotic entity[2]. The *connected coordination* strategy is shown in Figure 5.9. As before, the robots self-assemble (if necessary) in response to environmental contingencies. However, the assembled robotic entity is sensitive to its collective success or failure. Depending on the task, if a potential group failure is detected, the assembled entity can either modify its group level behaviour or its constituent robots can revert to independent task execution.

5.5.1 Strategy Implementation for the Hill Crossing Task

Our implementation allows the self-assembled swarm-bot to detect when it is inappropriately oriented, and then to retreat and rotate itself to try and achieve a more appropriate

[†]Upper bound ratio, as it is calculated using a lower bound for the mean completion time of the *preemptive self-assembly* strategy.

[2]The *basic self-assembly response* strategy did require some dedicated behavioural control to allow the individual s-bots to move while connected (see Section 5.2.1). However, this minimal co-ordination consisted of each connected s-bot rotating its tracks with respect to its turret to enable individual 'selfish' phototaxis (the orientation of the turret being fixed by the connections between s-bots). The group level behaviour was simply what emerged from the combination of the 'selfish' behaviours of the constituent s-bots.

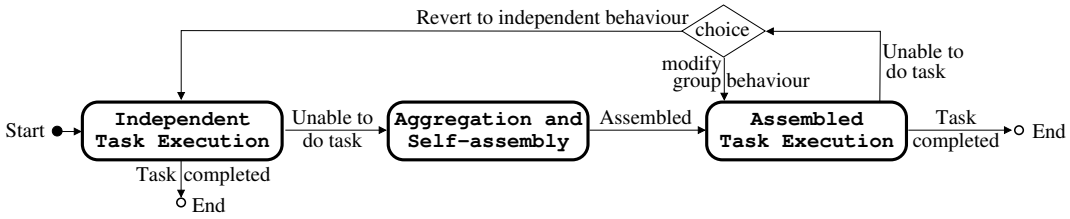


Figure 5.9: *Connected coordination* strategy: group-level behaviour. In this study, the choice between modifying group behaviour and reverting to independent execution is task specific and is therefore decided in advance when implementing the strategy for a particular task.

orientation. The s-bots self-assemble, as before, if they encounter a difficult hill. When the self-assembled swarm-bot encounters the hill, it analyses the orientation of the hill and compares it to its own orientation. If its own orientation is insufficiently perpendicular to the orientation of the hill, the swarm-bot retreats to flat ground by performing anti-phototaxis. It then rotates until its orientation is appropriate with respect to the remembered orientation of the hill. The swarm-bot then performs phototaxis again, and upon encountering the hill again compares its orientation to that of the hill. If its orientation is appropriate, it continues to perform phototaxis and thus navigates over the hill. Otherwise, the cycle of retreating and rotating repeats until the swarm-bot has an appropriate orientation. Note that the 'choice' in this implementation of the *connected coordination* strategy is to modify the group behaviour rather than reverting to individual behaviour. (For an example of an implementation of this strategy for another task, where the choice to revert to individual behaviour is more appropriate, see Chapter 6.)

The distributed behavioural control uses a leader-follower architecture (see Figure 5.10). The s-bot that seeds the self-assembly process becomes the *lead s-bot* and is responsible for determining whether or not the swarm-bot is appropriately rotated. Using its LEDs, the lead s-bot issues instructions to advance, retreat or rotate to all other s-bots (*follower s-bots*) in the swarm-bot. Follower s-bots illuminate their own LEDs to mimic the LEDs of the s-bot they are gripping (which is guaranteed to be closer to the lead s-bot in a linear morphology). In this way, instructions propagate along the swarm-bot from the lead s-bot to all the follower s-bots. The only exception is if a follower s-bot detects the hill before the lead s-bot, in which case the instruction to retreat propagates in the opposite direction.

The distributed behavioural control for the *connected coordination* strategy is an extension of the distributed behavioural control for the *basic self-assembly response* strategy. Control execution is branched into one of two possible finite state machine extensions (one for the lead s-bot, one for the follower s-bots) after the system has self-assembled. The two finite state machine extensions are shown in Figure 5.10. All of the s-bots start by executing the controller for the *basic self-assembly response* strategy, as illustrated in Figure 5.3. However, once they are in state `Wait`, instead of switching to state `Connected_Phototaxis`, they trigger the relevant *connected coordination* strategy finite state machine extension. In other words, the behavioural controller for the *connected coordination* strategy is constructed by substituting the state `Connected_Phototaxis` in Figure 5.3 with the state `Responsive_Phototaxis` in Figure 5.10 from either the lead s-bot finite state machine, if the s-bot seeded self-assembly, or, otherwise, the follower s-bot finite state machine.

Figure 5.11 illustrates a two s-bot system executing the behavioural controller for the

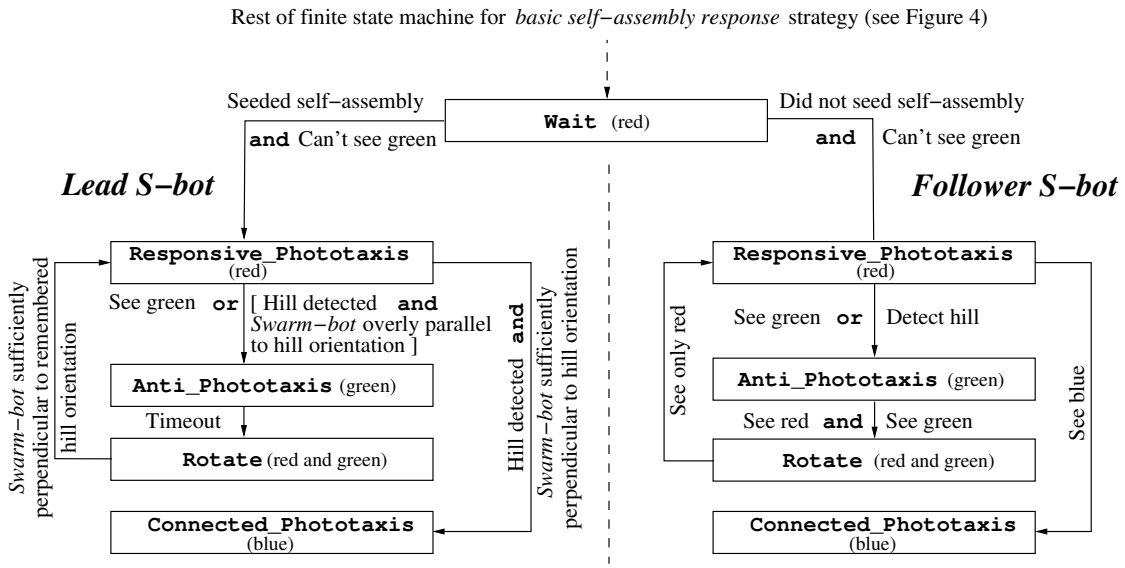


Figure 5.10: Distributed behavioural control to implement the *connected coordination* strategy for the hill crossing task. This distributed behavioural control consists of finite state machine extensions to the distributed behavioural control for the *basic self-assembly response* strategy. The s-bot first executes the distributed behavioural control for the *basic self-assembly response* strategy (see Figure 5.3). However, instead of executing the `Connected_Phototaxis` state the s-bot switches into state `Responsive_Phototaxis`, either as a lead s-bot, if it seeded self-assembly, or, otherwise, as a follower s-bot.

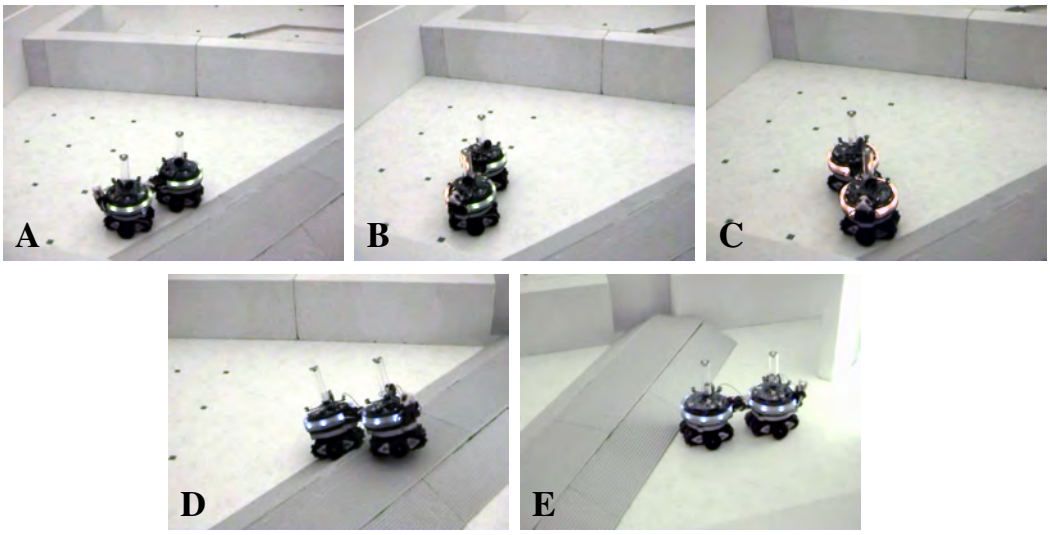


Figure 5.11: Execution of the controller for the *connected coordination* strategy. (A): The s-bots have already aggregated and self-assembled as a response to having encountered the hill. The connected swarm-bot performs phototaxis and approaches the hill with an inappropriate (random) orientation. Having detected the hill, the swarm-bot performs antiphototaxis, with the result that it moves away from the hill. (B): The swarm-bot rotates until it has a more appropriate orientation (based on memory of the hill orientation). (C): The swarm-bot aproaches the hill with its new orientation. (D): The swarm-bot recognises that it has an appropriate orientation and attempts to overcome the hill. (E): The swarm-bot arrives in the target area. For a video of this experiment see [110].

connected coordination strategy. In state `Responsive_Phototaxis`, the s-bots perform collective phototaxis while constantly checking the orientation of any hill they encounter with respect to the orientation of the swarm-bot (each s-bot uses its inclinometers to determine the orientation of the hill and its camera to determine the orientation of the swarm-bot). If a hill is encountered and the orientation of the swarm-bot is appropriate with respect to the orientation of the hill (perpendicular with a tolerance of 20°), the s-bots continue performing phototaxis to the light source, but no longer check the orientation of any encountered hills (state `Connected_Phototaxis`). If the orientation of the swarm-bot is not appropriate, the s-bots remember the orientation of the hill and retreat away from the hill for a given length of time (state `Anti_Phototaxis`). They rotate until the orientation of the swarm-bot is appropriate with respect to the remembered hill orientation (state `Rotate`), and then start performing collective phototaxis again (state `Responsive_Phototaxis`). For a detailed description of the individual states in this finite state machine see [110, 111].

5.6 Benefits of Connected Coordination

In this section we present experiments that we conducted with the *connected coordination* strategy to analyse the benefits of coordinating the sensing and actuation of the assembled robots to allow them to act as a single collective entity. We restrict our experimentation to the two s-bot case, as a two s-bot swarm-bot is linear by definition. We thus avoid the additional challenge of either checking whether the swarm-bot that has formed is linear or of enhancing the self-assembly process to explicitly generate a linear morphology[3].

5.6.1 Results

Table 5.1 shows how the *connected coordination* strategy compares against the other strategies in the difficult hill environments. With the *connected coordination* strategy, over 60 trials, two s-bots achieved the optimal task completion rate of 100% (see last row of the table). This increase in task completion rate can be attributed to connected coordination, that ensures the swarm-bot is appropriately rotated with respect to the hill. The effectiveness of the rotation mechanism can been seen in Figure 5.12 (left). Swarm-bots with connected coordination orient themselves against each of the three hill orientations significantly better than swarm-bots without connected coordination (two-tailed Mann-Whitney, $p < 0.001$).

Table 5.1: Experimental results for environments with the difficult hill. All strategies were evaluated in 60 independent trials. The last two columns show the percentage of s-bots that correctly assembled and the percentage of s-bots that completed the task.

# S-bots	Strategy	Assembled	Completed
1	*Independent execution only*	-	0.0%
2	*Basic self-assembly response*	100.0%	35.0%
3	*Basic self-assembly response*	98.3%	81.7%
2	***Connected coordination***	**100.0%**	**100.0%**

[3]To let more than two s-bots self-assemble into a linear formation, more elaborate control over the self-assembly process would be required. This type of morphological control is the subject of Chapters 7 to 9.

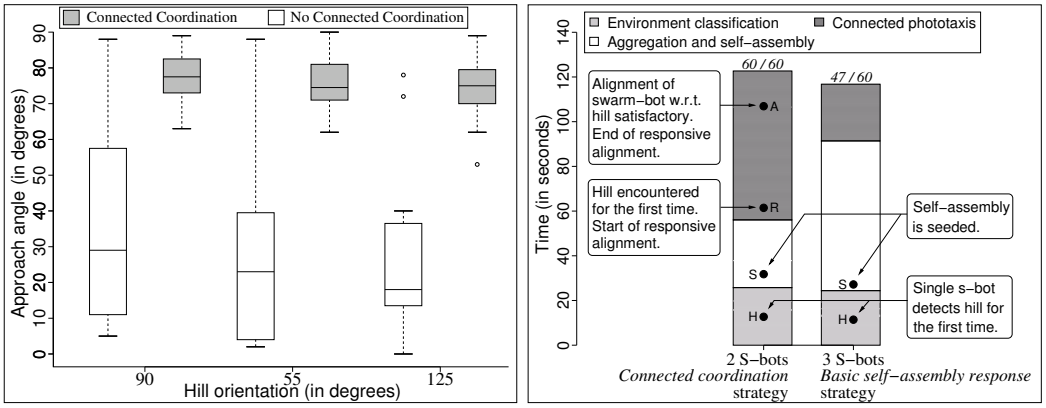


Figure 5.12: Left: Box-and-whisker plot showing acute approach angles of two s-bot swarm-bots (orientation of the swarm-bot with respect to the orientation of the hill). The orientation of the swarm-bot is measured at the moment that the swarm-bot first makes contact with the hill on its final approach (if the swarm-bot approached the hill more than once, the earlier approaches are ignored). Each box represents 20 trials. White boxes: swarm-bots without connected coordination. Grey boxes: swarm-bots with connected coordination. Right: Break-down of mean completion times for 2 s-bots using the *connected coordination* strategy (left bar) and for 3 s-bots using the *basic self-assembly response* strategy (right bar) in the difficult hill environments. Only data from trials that were completed by all s-bots are presented (number of completed trials and number of trials in total are indicated above each bar).

We have seen in Section 5.3.2 that a two s-bot swarm-bot relying only on benefits of scale has a success rate of 35%. To improve the success rate, we investigated two options.

- The first option was to further rely on the simple benefits of scale by increasing the number of s-bots to three whilst still using the *basic self-assembly response* strategy. This resulted in a task completion rate of 81.7%. However, the self-assembly and aggregation process becomes more complex with three s-bots and therefore takes longer and is more prone to failure (see also Table 5.1)

- The second option was to use a different strategy altogether. Using the *connected coordination* strategy, the assembled s-bots coordinated their sensing and actuation to leverage their collective morphology. This strategy achieved an optimal task completion rate. However, the responsive rotation mechanism is an extra time overhead and on average took up 37.1% of the total task completion time.

A breakdown of the mean completion times for these two options in the difficult hill environments is given in Figure 5.12 (right). Overall, there is no significant difference in the mean completion times of the *basic self-assembly response* strategy with three robots and the *connected coordination* strategy with two robots (two-tailed Mann-Whitney, $p < 0.001$).

5.7 Scalability

The behavioural control that we used to implement our strategies was strictly distributed and relied only on local communication. Thus, although the hill climbing task we presented

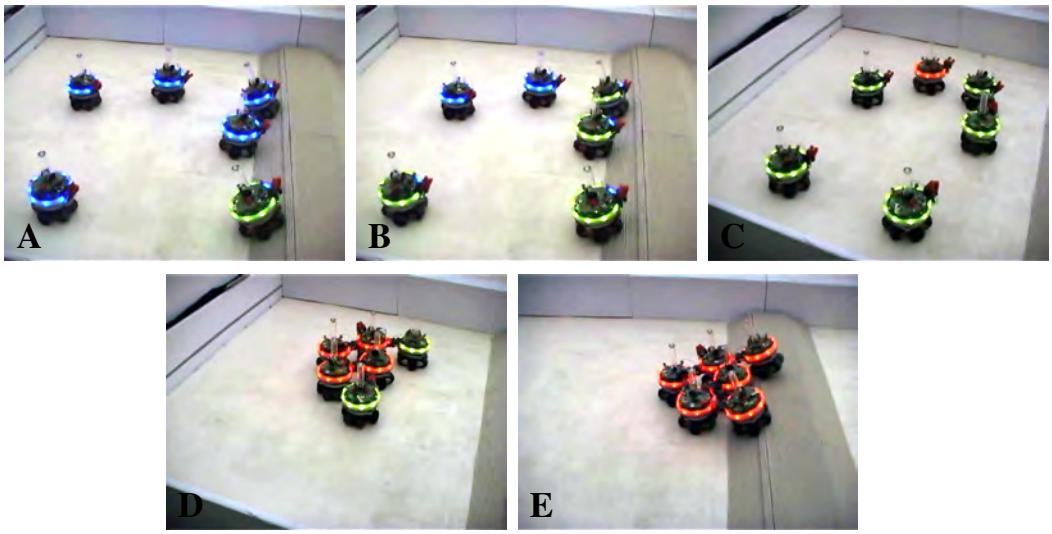


Figure 5.13: Scalability experiment in the difficult hill environment: 6 s-bots using the *basic self-assembly response* strategy self-assemble to cross the hill.

was solved optimally with two robots executing the *connected coordination* strategy, we would nonetheless expect our distributed behavioural control to scale to larger numbers of robots. To test this hypothesis we conducted several proof-of-concept experiments repeating our detailed experimentation but with larger numbers of robots. Figures 5.13 and 5.14 show experiments using the *basic self-assembly response* strategy in the difficult hill environment with six real robots. Figure 5.15 shows the responsive rotation mechanism of the distributed behavioural control for the *connected coordination* controller working with three (manually pre-assembled) s-bots.

Figure 5.14 is interesting, as it shows a rudimentary form of group size regulation that we had not explicitly encoded into the system, but that emerged as a result of the distributed, stochastic nature of the behavioural control. In this trial, the robots aggregated into two separately seeded groups of four and two s-bots respectively. This double seeding was possible because in this particular trial the two s-bots that became seeds were out of visual range of each other. Both groups successfully self-assembled and navigated over the hill. This type of group size regulation is very basic, and is self-organised probabilistically based on the density of robots and the visual range of the camera. It is neither practical nor efficient for the size of the compound robotic entity to grow indefinitely. In our experiments with 6 real robots, we already noticed that interference effects were common—the larger numbers of constituent robots meant that it was more probable for the tracks of one robot to impede the tracks of another robot during collective motion. In addition, the characteristics of the hardware platform place physical limits on the optimal (and maximum) size of compound entities [95].

5.8 *Basic Self-Assembly Response* strategy in a Hole Crossing Task

In this section, we show how the generic *basic self-assembly response* strategy can be applied to another problem domain — a hole crossing task. In this task, the robots are required to cross a hole of a priori unknown width as they navigate to a light source. We

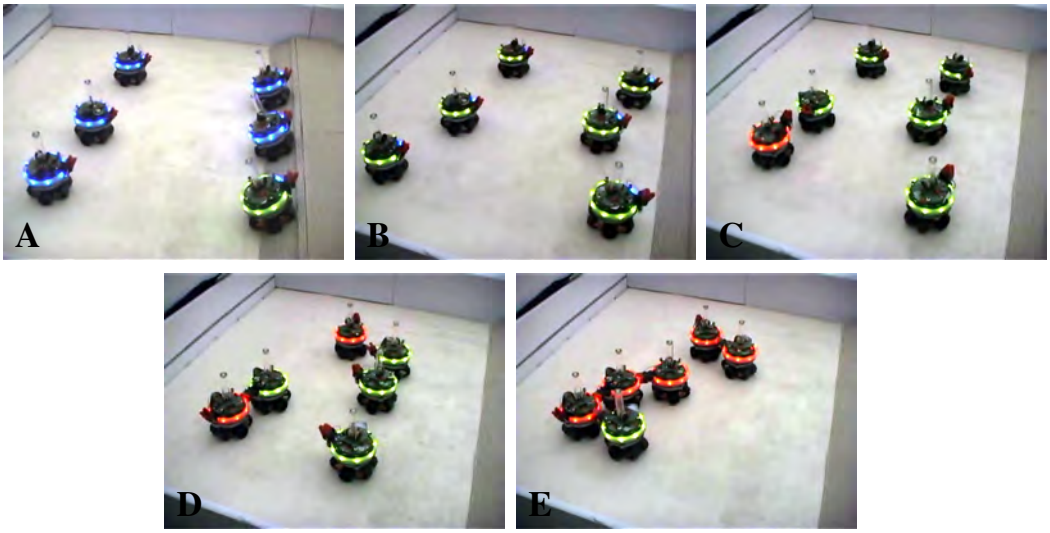


Figure 5.14: Scalability experiment in the difficult hill environment: 6 s-bots using the *basic self-assembly response* strategy self-assemble into two groups, both of which successfully cross the hill.

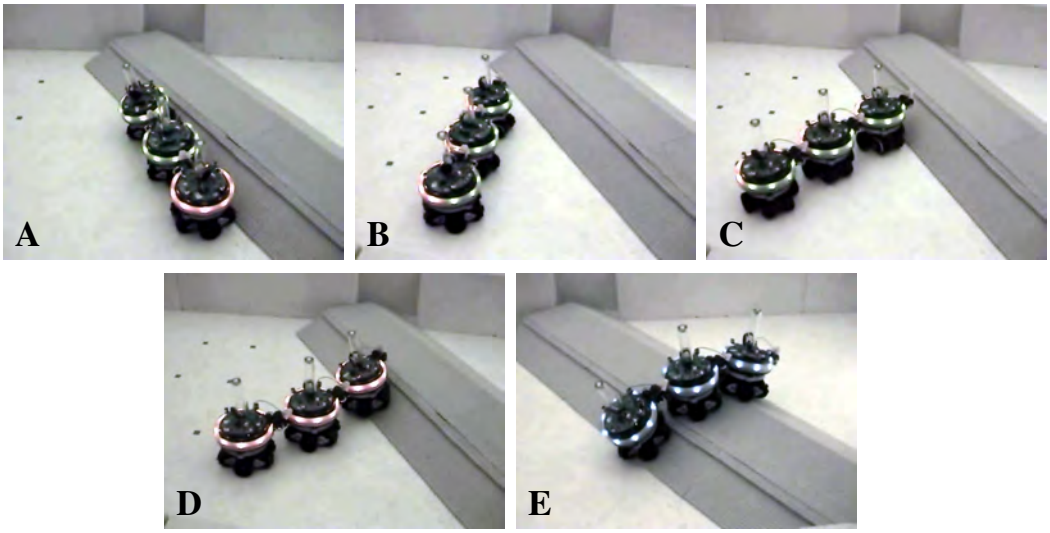


Figure 5.15: Scalability experiment in the difficult hill environment: 3 manually pre-assembled s-bots using the *connected coordination* strategy rotate to cross the hill. For videos of these experiments see [110].

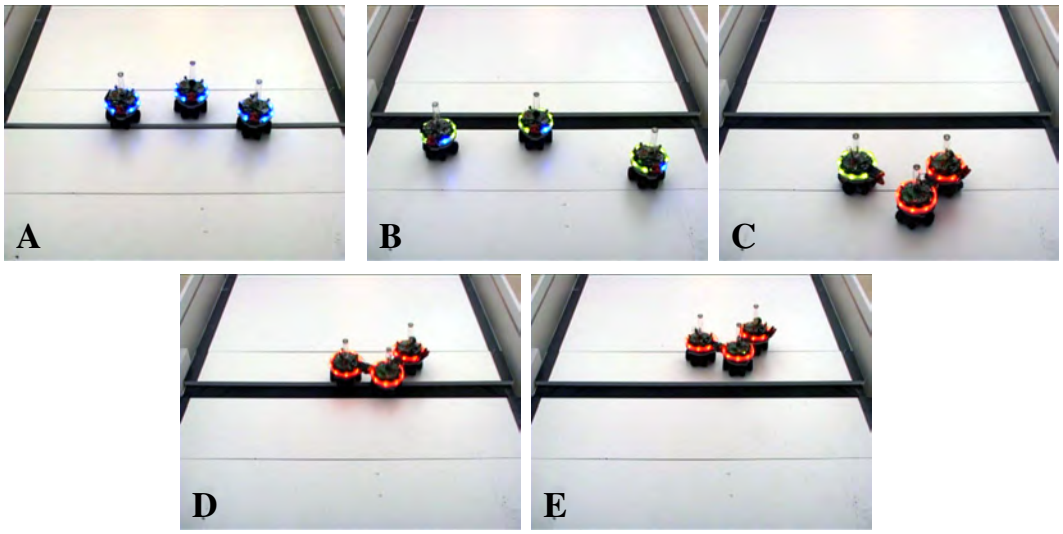


Figure 5.16: The *basic self-assembly response* strategy in the hole crossing task. (A): s-bots cross a 3 cm hole individually. (B-E): s-bots respond to a 10 cm hole by self-assembling and crossing collectively. For videos of these experiments see [110].

parametrise the task with two possible hole widths—a 3 cm hole that a single s-bot can cross, and a 10 cm hole that a single s-bot would topple into [95]. The robots detect holes with their infrared ground sensors. To implement the *basic self-assembly response* strategy for hole crossing, we only needed to make a single minor modification to the distributed behavioural control for the hill crossing task. We replaced the 'too steep' condition with a 'wide hole' condition that is triggered when more than one ground sensor fires at the same time (see Figure 5.3). Results of a single trial for each width of hole are shown in Figure 5.16. The success of the hole crossing experiment shows that the strategies we developed are reusable in other problem domains.

5.9 Discussion and Conclusions

The work presented in this chapter contributed to self-assembly research by making explicit the value that autonomous self-assembly adds to a robotic system. We also explicitly considered the costs of self-assembly. Research to date has largely assumed that self-assembly must be valuable, without analysing exactly why and has also ignored the overheads associated with the self-assembly process. In this chapter, we isolated different types of performance benefit that self-assembly can bring to a robotic system, namely benefits of scale, benefits of decisional autonomy and benefits of connected co-ordination. By comparing different generic self-assembly strategies, we were able to measure the costs and benefits of self-assembly in terms of task completion performance. Although our analysis focused on the hill crossing task, we also demonstrated that our generic strategies were applicable to another task — hole crossing.

In this chapter, we refined the distributed behavioural control for the hill crossing task and added a connected coordination mechanism to allow connected robots to benefit from collective sensing and actuation. We analysed the performance of the distributed behavioural control, and showed under what circumstances self-assembly adds value as an autonomous response mechanism. However, the approach presented in this chapter still

has the following limitations:

1. The system can exercise decisional autonomy over when and if to self-assemble. However, there is no concept of appropriate resource allocation — the system is not required to determine the size and number of composite robotic entities required to solve the task.

2. There is no active control over the morphology that is generated through self-assembly in response to environmental contingencies. Morphology based issues of both shape and size need to be addressed.

In the next chapter, we address limitation 1, above. We apply the self-assembly strategies from this chapter to a robot rescue task, and show how the system can allocate resources appropriately based on the size of the robot that needs to be rescued. In Chapters 7 onwards, we address limitation 2 above by considering a distributed approach to the generation of specific morphologies using self-assembly.

Chapter 6

Resource Allocation

Self-assembling systems are well suited to parallel task execution. Some tasks might be solvable by individual robots working in parallel. More demanding tasks might require the greater physical abilities of composite entities created through self-assembly. In some cases, depending on the task and the available resources (i.e., number of robots), acting as a single large composite entity might be appropriate. In other cases, many composite entities could work in parallel and possibly cooperate with each other. Such potential flexibility is one of the reasons that self-assembling systems are so alluring.

To achieve this kind of flexibility, a self-assembling system must be able to analyse its task and environment, and be able to distribute resources accordingly. This problem of resource allocation has been largely ignored in the self-assembly literature. There is a large body of literature on task allocation in non-self-assembling multi-robot systems. However, little of this work is directly applicable to self-assembling systems, where the parameters of the task must determine the nature and extent of physical cooperation and/or parallel execution.

In this chapter, we explore the problem of resource allocation using a real world group transport scenario. In our scenario, dedicated 'rescue' robots must find broken, immobile robots and transport them to a designated repair zone. The broken down robots can either be single agents or pre-assembled composite robotic entities. A single broken down agent can be transported by a single rescuing agent, whereas a broken down composite entity is sufficiently heavy that multiple rescue robots must self-assemble in order to effect the rescue. The system has no a priori knowledge either of the number of rescue robots or of the number and size of the broken entities.

We present a distributed controller that solves the above task while efficiently allocating resources. Each rescue robot tries to move any broken robots that it finds, and independently determines whether or not the object is successfully being moved. Based on this determination, the rescuing robot uses local communication to either attract or repel other rescue robots. This chapter is organised as follows. In Section 6.1, we describe the experimental setup for the robot rescue task. In Section 6.2, we describe the distributed behavioural control required to carry out the task. In Section 6.3, we present results of real-world experiments. Finally, in Section 6.4, we discuss the contribution of this chapter and present our conclusions.

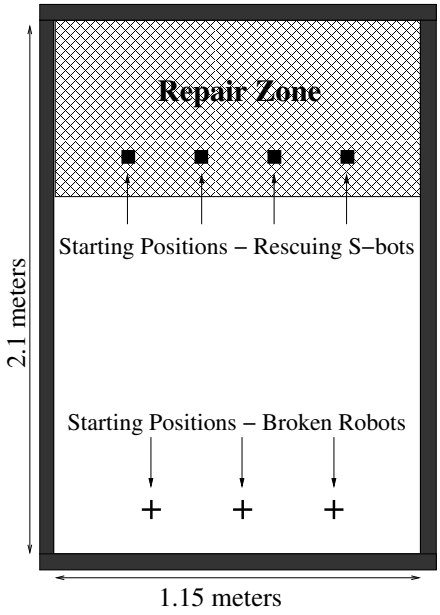


Figure 6.1: The arena for the robot rescue task.

6.1 Experimental Set-up

Our experimental setup requires 'rescue' s-bots to search the arena shown in Figure 6.1, to find broken robots and then to transport them to the designated repair zone (darker floor). A light source is located two meters outside of the arena beyond the repair zone (not shown in Figure 6.1). Transporting s-bots perform phototaxis to ensure that they transport the broken robots in the correct direction towards the repair zone.

All of our experiments involve either one or two rescuing robots and either one or two broken robots. We use two types of broken robot. We refer to a single broken s-bot as a *1-s-bot broken robot* (a single rescuing s-bot is able to transport a 1-s-bot broken robot). We refer to a broken robot that is a composite entity made up of two physically connected s-bots as a *2-s-bot broken robot* (the rescuing s-bots must team up in order to collectively transport a 2-s-bot broken robot). We consider broken robots to be immobile. However, we make the simplifying assumption that broken robots are still able to use their LEDs to signal that they require assistance.

6.2 Distributed Behavioural Control

The desired behaviour of our multi-agent system is for the rescue s-bots to locate and transport the broken robots to the repair zone. Ideally, based on the properties of the broken robots (size, weight), only the minimum required number of rescue s-bots should be allocated to transport each broken robot, thus leaving as many rescue s-bots as possible available for other tasks.

We use an instantiation of the *connected coordination* strategy (see Figure 5.9) for the robot rescue task. The resulting behavioural control logic executed independently by each rescue s-bot is shown in Figure 6.2. A rescue s-bot searches for any broken robots by performing a random walk with its blue LEDs illuminated (`Random Walk`). Broken robots signal that they need help by illuminating their red LEDs. When a rescue s-bot detects

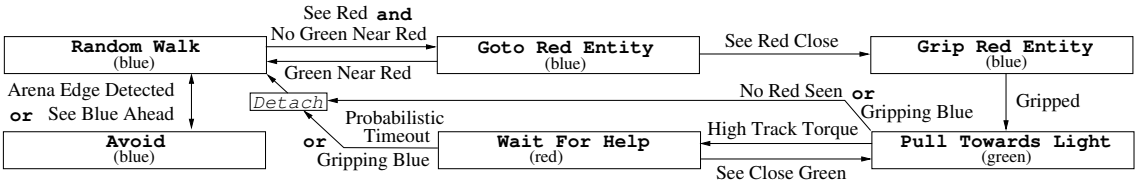


Figure 6.2: Distributed behavioural control to implement the *connected coordination* strategy for the robot rescue task. Each rescue s-bot executes this behavioural control logic independently.

red LEDs without any nearby green LEDs, it assumes that it has seen a broken robot that is not currently being rescued and heads towards the red LEDs (`Goto Red Entity`). If more than one red entity is seen, the rescuing s-bot picks one of the red entities at random. It then grips the broken robot (`Grip Red Entity`) and tries to pull the broken robot to the repair zone by heading towards the light source (`Pull Towards Light`) with its green LEDs illuminated.

As long as a rescue s-bot is successfully pulling a broken robot (low track torque), it stays green. This tells other rescue s-bots within camera range not to go towards the broken robot it is pulling. However, if the rescue s-bot fails to move the broken robot (high track torque measured consistently for 5 seconds), the rescue s-bot stops pulling and illuminates its red LEDs (`Wait For Help`). At this point, the rescue s-bot is indistinguishable from the broken robot that it is still gripping. Thus, other rescue s-bots will approach and grip either the broken robot or any rescue s-bots that are attached to the broken robot and are in state `Wait For Help`. When a new rescue s-bot attaches, it will turn green as it attempts to pull the broken robot. Any attached s-bots in state `Wait For Help` are prompted by the sight of green LEDs to try to pull again. If the new larger number of attached rescue s-bots is now sufficient to move the broken robots, all of the rescue s-bots will stay green and thus prevent any further rescue s-bots from approaching. Otherwise, after 5 seconds of high torque, they will realise that the broken robot is still not movable, and will switch to state `Wait For Help`.

This process repeats itself, with progressively more rescue s-bots attaching to the broken robot until there are enough rescue s-bots to move the broken robot. In this way, the system automatically finds the minimum group size capable of moving a broken robot. Note that in this implementation of the *connected coordination* strategy, the success / failure detection mechanism is based on the s-bot track motor torque sensors, while the choice required by the strategy is to revert to independent action (individual attached s-bots probabilistically detach and revert to individual action) when the system is not able to recruit enough rescue s-bots to move a particular object (group failure).

If there are several large composite broken robots present, and a small number of rescue s-bots, it is possible that the individual rescue s-bots might randomly distribute themselves among the broken robots in such a way that none of the large broken robots can be moved. To prevent this type of deadlock situation, a rescue s-bot in state `Wait For Help` has a low probability of detaching. A detached robot returns to state `Random Walk` and thus turns blue, which signals to any s-bot that might be gripping the detached robot that it should in turn detach. Rescuing robots also detach if they no longer detect any red LEDs—in our experiments, the broken robots detect arrival in the repair zone using ground sensors, and switch off their red LEDs. Together, these two detachment mechanisms ensure that as long as there are enough rescue s-bots present in the environment to move any given

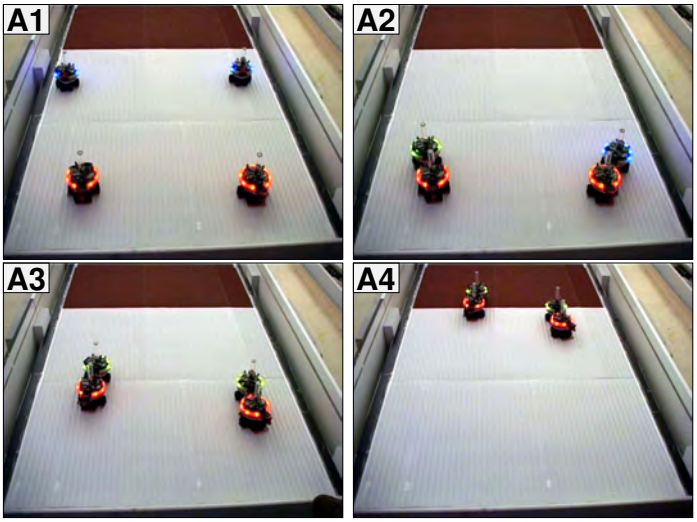


Figure 6.3: Robot Rescue. Physical Cooperation. A robot rescue experiment with two rescue s-bots and two 1-s-bot broken robots.

broken robot, the rescue s-bots will eventually combine forces to move the broken robot.

6.3 Results

We carried out four sets of real-world experiments. Each set of experiments is defined by the number of rescuing s-bots, the number of broken robots and the size of the broken robotic entities. The rescuing robots have no a priori knowledge of the experiment configuration. Videos of the experiments can be found on the web at: `http://iridia.ulb.ac.be/supp/IridiaSupp2009-011`.

6.3.1 Rescuing Broken Robots in Parallel

We conducted 5 trials of an experiment with two rescue s-bots and two 1-s-bot broken robots (photos A1-A4 are of a single trial). In each experiment, the system successfully allocated a single rescue robot to each of the broken robots and the broken robots were transported in parallel to the repair zone.

This experiment shows that when possible the system correctly 'chooses' parallel execution (the incorrect choice would be for two rescue s-bots to assemble to the same broken robot).

6.3.2 Physical Cooperation to Rescue a Broken Robot

We conducted 5 trials of an experiment with two rescue s-bots and a single 2-s-bot broken robot (photos B1-B4 are of a single trial). In each trial, one rescue s-bot found the broken robot, then tried and failed to move the broken robot alone. The attached rescue s-bot waited for help, and after the other rescue s-bot attached, together the two rescue s-bots succeeded in transporting the broken robot to the repair zone.

In one of the trials, the initial attachment configuration of the rescue s-bots failed to move the broken robot. One of the rescue s-bots detached based on the probabilistic time out twice, but reattached both times. In the final configuration the rescue s-bots succeeded in transporting the broken robot to the repair zone.

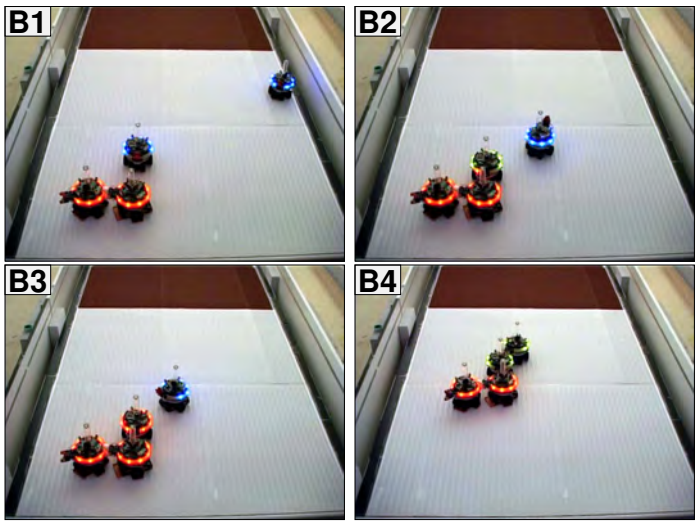


Figure 6.4: Robot Rescue. Physical Cooperation. A robot rescue experiment with two rescue s-bots and one 2-s-bot broken robot.

This experiment shows that the system correctly allows the rescue s-bots to physically coordinate in order to solve a task that is beyond the physical capacities of a single s-bot.

6.3.3 Efficiency Gains Through Group Size Regulation

We conducted 5 trials of an experiment with two rescue s-bots and a single 1-s-bot broken robot (photos C1-C4 are of a single trial). In each trial, a single rescue s-bot connected to the broken robot, and successfully transported it to the repair zone. The other s-bot continued to explore the arena without attempting to attach to the moving broken robot.

This experiment shows that the system correctly allocates the minimum number of rescue s-bots (in this case 1 s-bot) to a task, and leaves other rescue s-bots free to potentially carry out other tasks.

6.3.4 Reallocation of Resources in a Deadlock Situation

We conducted 5 trials of an experiment with a single rescue s-bot and two broken robots: one 1-s-bot broken robot and one 2-s-bot broken robot (photos D1-D4 are of a single trial). In this experiment, we initially 'break' the 2-s-bot broken robot (i.e., illuminate its red LEDs). We allow the rescuing s-bot to find and attach to the heavy broken robot before we 'break' the 1-s-bot broken robot.

In four out of the five trials, the rescue s-bot probabilistically timed out, explored the arena, found and attached to the 1-s-bot broken robot and successfully transported it to the repair zone (in one of these successful trials the rescue s-bot first re-attached to the 2-s-bot broken robot and timed out again). In a single trial, the rescue s-bot failed to detach correctly due to a hardware failure.

This experiment shows that the system correctly resolves a deadlock situation where a rescue s-bot is attempting an impossible task, and manages to reallocate resources (the rescue s-bot) to another task that is feasible.

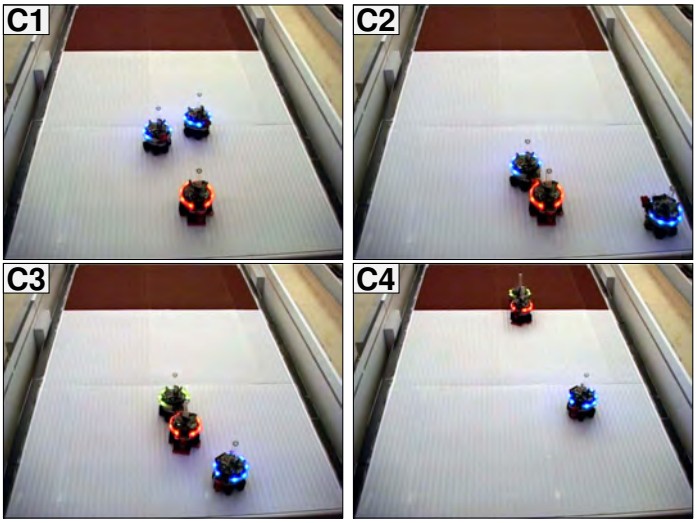


Figure 6.5: Robot Rescue. Group Size Regulation. A robot rescue experiment with two rescue s-bots and one 1-s-bot broken robot.

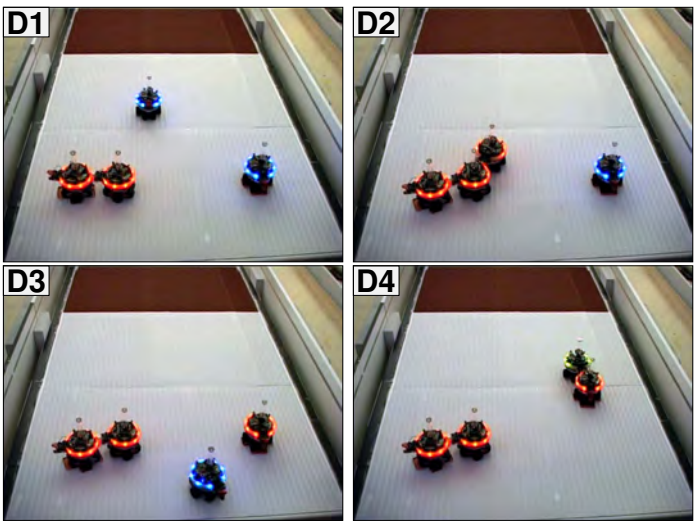


Figure 6.6: Robot Rescue. Reallocation of Resources. A robot rescue experiment with two rescue s-bots and one 1-s-bot broken robot.

6.4 Discussion and Conclusions

The work presented in this chapter contributed to self-assembly research by demonstrating distributed resource allocation in a real-world self-assembling robotic system. Our resource allocation mechanism avoided central control by using local rules of attraction and repulsion in combination with a motor torque based mechanism that allowed the individual robots to determine the success or failure of the composite entity of which they were part. The behavioural control principles could therefore be re-used to create similar resource allocation dynamics in any system where the individual robots have a means firstly of detecting group success or failure, and secondly of attracting and repulsing other robots.

In this chapter, we conducted real world experiments in a group transport rescue scenario which showed that our distributed control logic displays several important and novel properties. In particular, our system was able to maximise parallel execution (and hence efficiency) by allocating the minimum number of agents required to solve tasks of varying magnitudes. In addition, the system included a reset mechanism which allowed it to resolve potential deadlock situations (when all agents are distributed among tasks in a sufficiently sparse way that none of the tasks are solvable). We discovered that this mechanism also allowed for potentially beneficial random reconfiguration of spatial arrangements within a single transport rescue group.

The work in this chapter also showed that the self-assembly strategies developed in chapter 5 are applicable to a very different problem domain. In subsequent chapters, we address the problem of using self-assembly to generate different morphologies appropriate to particular tasks.

Chapter 7

Directional Self-Assembly

In this chapter, we present a behavioural control mechanism that provides control over the orientation of individual connections between robots on the swarm-bot platform. We call this mechanism *directional self-assembly*. Prior to this work, all self-assembly done with the swarm-bot platform had been completely stochastic, i.e., the robots would connect to each other with random orientations, thus generating random morphologies. The directional self-assembly mechanism is local, as it deals with a single connection between two robots. However, when used by distributed system level behaviour, directional self-assembly enables the generation of specific global morphologies. The local mechanism we present in this chapter is specific to the swarm-bot platform, as it relies on s-bot specific sensors and actuators. However, the system level behavioural control that we present in subsequent chapters abstracts away from platform specific details, and is therefore potentially applicable to other robotic platforms.

This chapter is organised as follows. In Section 7.1 we present our overall methodology. In Section 7.2, we detail the connection slot mechanism based on camera and LEDs that allows a robot to invite a physical connection from another robot while specifying a particular direction and orientation of connection. In Section 7.3 we present the low level distributed behavioural control that allows the robots to use the connection slot mechanism. In Section 7.4, we present results of real-world experiments we conducted to test the robustness and accuracy of directional self-assembly. Finally, in Section 7.5, we discuss the contribution of this chapter and show how the work from this chapter leads into the research presented in subsequent chapters.

7.1 Methodology

A robot invites a connection from other robots using its LEDs. By lighting up a particular configuration of LEDs, an s-bot can invite a connection at different points on its body. By choosing a particular approach pattern for the connecting robot, we can ensure that the orientation of a connecting robot relative to the robot to which is connecting is consistent for any given connection point.

We also present the navigation logic for robots that are not connected. This stochastic navigation logic allows non-connected robots to search for possible connection points, then navigate to the connection point. Distributed logic prevents interference when more than one robot is attracted to the same connection point.

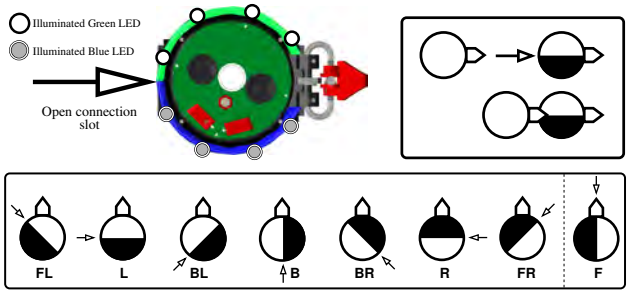


Figure 7.1: Above left: An s-bot with an open connection slot to its rear. To open the connection slot the s-bot illuminates its left green LEDs and its right blue LEDs. Above right: Diagrammatic representation of another s-bot connecting to the same connection slot. This representation is used for the rest of this thesis. Below: The 8 possible connection slots that an s-bot can open. Only seven connection slots can be used for extending the morphology. The eighth connection slot (F) indicates a grip point that is already occupied by the gripper of the s-bot displaying the connection slot. This connection slot is used for signalling instead (see Chapter 8).

7.2 The Connection Slot

An s-bot illuminates a particular configuration of LEDs to indicate a point on its body where another non-attached s-bot should grip. We term such a configuration of LEDs a *connection slot*.

The left-hand side and right-hand side of a connection slot are indicated by four illuminated green LEDs and four illuminated blue LEDs, respectively (see Figure 7.1). Each s-bot has 8 LED locations. A green LED, a blue LED and a red LED are located at each LED location. A connection slot can be opened between any two neighbouring LED locations, except between the two front LED locations, where the gripper is mounted. Thus an s-bot that is connected to a morphology can extend the local structure using one of seven different connection slots. As a connection slot requires the use of all eight LED locations, an s-bot can only open one connection slot at a time.

7.3 Behavioural Control

In this section, we detail the behavioural control logic that enables an s-bot to find, approach and attach to a connection slot. We refer to an s-bot that is displaying an open connection slot as an *extending s-bot*. We refer to an s-bot that is not yet part of the connected structure as a *free s-bot* when it is searching for a connection slot or as an *attaching s-bot* when it is attempting to grip an *extending s-bot*. We refer to an s-bot that has formed a successful connection and thus become part of the morphology as a *connected s-bot*. Figure 7.2 illustrates these terms by displaying a morphology in the process of being constructed using directional self-assembly. A connection slot is considered 'open' until a free s-bot attaches to it, at which point it is considered 'filled'.

We first describe the close range approach and gripping logic. We then describe the longer range navigation logic which the s-bots use to find and navigate to the connection slot.

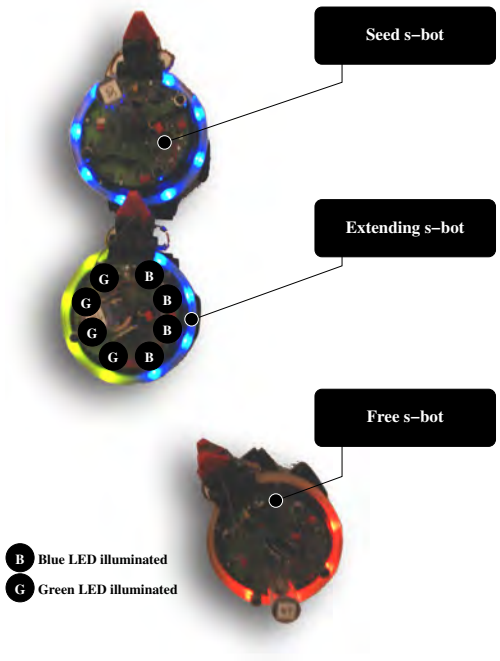


Figure 7.2: Roles in morphology growth: the *seed* is the s-bot starts a new morphology, an s-bot displaying a connection slot is an *extending s-bot*, and a *free s-bot* is one that is neither connected to a morphology nor is displaying a connection slot.

7.3.1 Approaching and Gripping a Connection Slot

The behavioural control logic for approaching and gripping a connection slot is designed to ensure that attaching s-bots always connect to any given connection slot with the same orientation (i.e., that for each connection slot position, the angle of the newly connected s-bot and the extending s-bot remains consistent). Our approach logic tries to achieve a 90 ° angle between connected and extending s-bots. In other words, the orientation of the connecting s-bot is normal to the body of the extending s-bot at the point of attachment.

An attaching s-bot uses the illuminated LEDs of an extending s-bot's open connection slot to calculate an *approach vector* (see Figure 7.3). The head of the approach vector is called the *grip point* and indicates the point on the extending s-bot's body at which the attaching s-bot should grip. The tail of the approach vector is called the *approach point*, and is 13 cm away from the body of the extending s-bot. The heading of the approach vector matches the orientation that the connection slot specifies for the attaching s-bot. This means that by simply following the approach vector to the grip point and then gripping, an attaching s-bot will grip the extending s-bot at the correct location and will assume the correct corresponding orientation.

Based on the approach vector calculation, we use a five phase strategy to let free s-bots find and grip a connection slot. The trajectory of a free s-bot approaching and gripping a connection slot is shown in Figure 7.3 (inset).

1. The free s-bot directly approaches the extending s-bot until it is within 35 cm of the grip point.

2. The s-bot circles around the extending s-bot until it is within +/- 20 ° of the approach

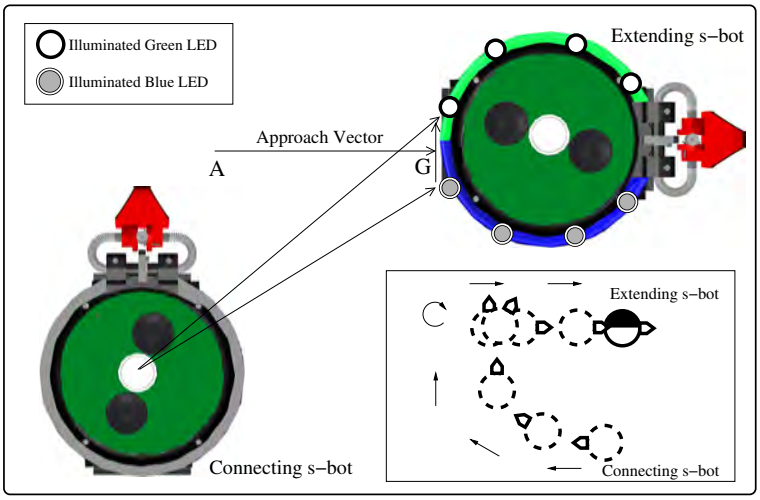


Figure 7.3: Based on an extending s-bots illuminated connection slot LEDs, an attaching s-bot calculates the approach point (A), grip point (G) and approach vector $\overrightarrow{AG}$. Inset: The motion of a connecting s-bot as it navigates to and grips a connection slot.

vector.

3. The s-bot navigates to the *approach point* at the start of the approach vector
4. The s-bot rotates to face the connection slot.
5. The s-bot approaches the connection slot and attempts to grip the extending s-bot.

Thus, an attaching s-bot first navigates to the approach point at the tail of the approach vector. It then rotates to face the grip point at the head of the approach vector, and only then starts to navigate along the approach vector to the grip point. The closer an s-bot gets to the grip point, the more accurately it perceives the LEDs of the connection slot, and therefore the more precisely it can calculate the approach vector. Corresponding corrections to the attaching s-bots trajectory are made continuously as the s-bot approaches the grip point. As the s-bot moves closer, the speeds of the left and right s-bot tracks are respectively set to:

$$s_l = 5.6\,\text{mm/s} + 21.8\,\text{mm/s} \cdot \frac{d}{130\,\text{mm}} \cdot f(\theta), \tag{7.1}$$

$$s_r = 5.6\,\text{mm/s} + 21.8\,\text{mm/s} \cdot \frac{d}{130\,\text{mm}} \cdot f(-\theta), \tag{7.2}$$

where d is the distance to the grip point, and $f : \theta \rightarrow [0, 1]$ is a function that maps the angular difference between the current heading and the ideal heading, θ, to a speed modifier in the range $[0, 1]$. The result of applying this speed modifier term to Eqn. (7.1) and Eqn. (7.2) is that the s-bot continually corrects its alignment as it approaches the connection slot. During the approach, the speed of the s-bot is reduced as a linear function of the distance to the grip point (the magnitude of the alignment corrections becomes correspondingly smaller). When the s-bot determines that it is close enough to connect, it attempts to grip by closing its gripper. If the grip fails, the s-bot moves back and starts navigating to the approach point again.

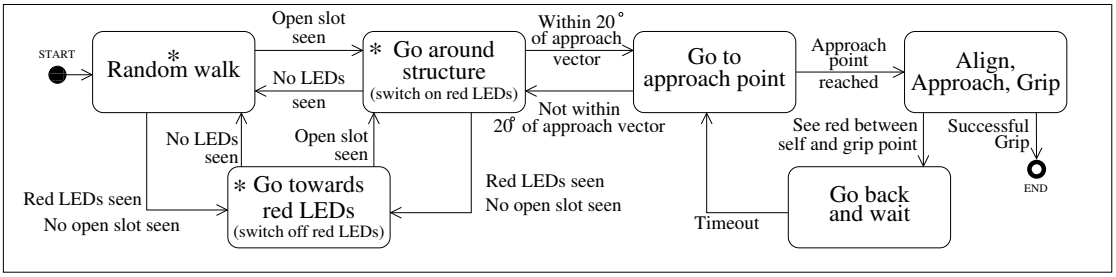


Figure 7.4: Free s-bot navigation logic for finding, approaching and gripping connection slot. A free robot performs a random walk if it sees no connection slots or red LEDs. If the robot sees a connection slot, it moves around the connected structure until it can approach the slot from the correct angle. If the approach angle is correct, but another free robot is currently connecting to the open slot, the robot moves back and waits a random amount of time before proceeding. If a free robot does not see an open connection slot, it navigates towards red LEDs (the assumption being that these LEDs must belong to a free robot within visual range of a connection slot). When a connection has been made, code specific to a particular morphology is executed. Obstacle avoidance based on vision and proximity sensor readings is performed in the three control states marked by an asterisk (*).

7.3.2 Finding and Navigating to a Connection Slot

In our directional self-assembly paradigm, any free s-bot can attach to any open connection slot. The navigation logic for free s-bots must therefore firstly allow free s-bots to search for open connection slots, and secondly ensure that the free s-bots do not interfere with each other while either finding or approaching a connection slot (as multiple free s-bots may see the same connection slot at the same time).

The navigation logic for the free s-bots is shown in Figure 7.4. If a free s-bot cannot see any coloured LEDs, it performs a random walk until it can see some LEDs. If a free s-bot can see an open connection slot, it starts the five phase navigation strategy. While it is performing this navigation, it turns on its red LEDs.

If a free s-bot can only see red LEDs (and can't therefore see a connection slot), it switches off its own LEDs and navigates towards the closest red LED. The assumption is that this red LED must belong to an s-bot within visual range of a connection slot. This mechanism helps to compensate for the relatively short perception range of the s-bots (approximately 50 cm). During initial experiments before this logic was implemented, free s-bots would often spend a significant amount of time at the edges of the arena searching for the connected structure.

If an attaching s-bot sees red LEDs between itself and the connection slot during the approach phase (described in Section 7.3.1), the attaching s-bot assumes that there is another s-bot already trying to attach to the same connection slot. It therefore abandons its approach, moves back, and waits a random amount of time before proceeding. This prevents the free s-bots from obstructing each other by trying to connect to the same connection slot at the same time.

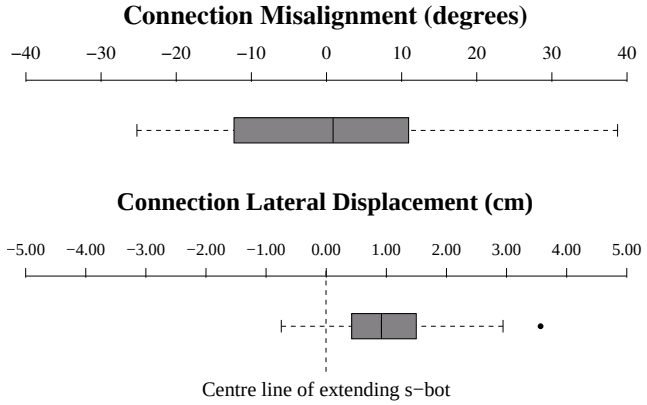


Figure 7.5: Precision of the connection slot mechanism. Above: Angular precision, measured by the mismatch between the alignment of the attaching s-bot and the alignment of the extending s-bot. Below: Positional precision, measured by the lateral displacement of attaching s-bot's grip from extending s-bot's center line.

7.4 Experimental Results: Precision and Timing

We analyzed the angular precision, the positional precision and the timing of the directional self-assembly mechanism. We conducted 96 experiments in which a single free s-bot attached to a stationary extending s-bot. In every experiment, the extending s-bot had the same position and orientation. At the start of each experiment, the extending s-bot opened connection slot B (directly behind it, see Figure 7.1). For the free s-bot, we used 12 possible starting positions and eight possible starting orientations. Over the 96 experiments we used each possible combination of these starting positions and orientations once. The starting positions for the free s-bot were evenly distributed around a circle of radius 35 cm centered on the extending s-bot.

The angular precision of the connection slot mechanism is measured by the angular misalignment of the attaching s-bot from the orientation specified by the connection slot. In our experiments, as the extending s-bot opens connection slot B, we measured the mismatch between the alignment of the attaching s-bot and the alignment of the extending s-bot. The results are shown in Figure 7.5 (top). Note that the mean angular misalignment is very close to zero.

The positional precision of the connection slot mechanism is measured by the lateral displacement between the grip point specified by the connection slot and the point at which the attaching s-bot actually grips. In our experiments, as the extending s-bot opens connection slot B, we measured the distance from the center line of the extending s-bot to the point which the attaching s-bot gripped. The results are presented in Figure 7.5 (bottom). We can see that there is a clear bias towards attaching to the right of the center line. This is due to a hardware asymmetry in the distribution of the s-bot LEDs. When an s-bot illuminates its 4 left green LEDs and its 4 right blue LEDs, the point equidistant between the green and blue LEDs at the back of the s-bot is nearly 1 cm to the right of its center line. However, the alignment of the connection slot mechanism is not affected by this asymmetry. We did not, therefore, deem it necessary to compensate for the positional inaccuracy in this study.

The mean times spent on the different navigation activities are shown in Figure 7.6. In these results, we do not represent the activity of navigating directly to the connection slot

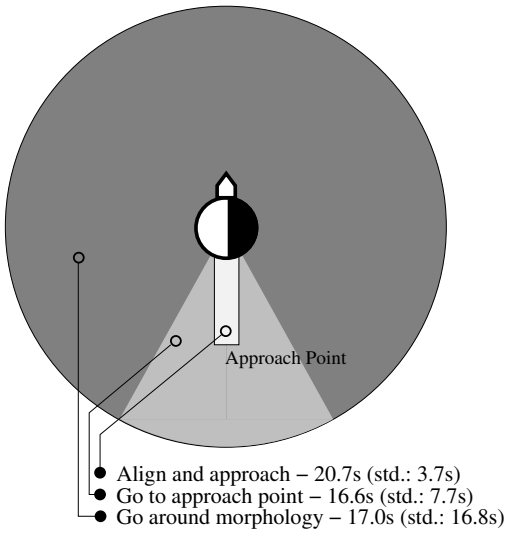


Figure 7.6: Timing of the connection slot mechanism. Mean time and standard deviation spent on different activities while forming a connection.

(phase 1 in 7.3.1), since in our experiments the initial placement of the s-bots is sufficiently close to the connection slot that they always start directly in phase 2 or phase 3 — either navigating around the morphology or navigating to the approach point. The largest share of the time was spent on the final alignment and approach phases, during which the s-bot rotates on the approach point to face the grip point and then follows the approach vector to the grip point. Although the distances covered during the different activities vary significantly, the mean times spent on the different activities are comparable. This is a consequence of the increasing precision required as the s-bot gets closer to the connection slot — the more precision required, the slower the s-bot moves. The mean time from the start of an experiment until the s-bot attached was 54.3 s.

In all 96 experiments, the free s-bot successfully connected to the extending s-bot. In 2 of the 96 experiments the grip failed on the first attempt, and the free s-bot retreated to try another angle. In a further four experiments, the free s-bot retreated to try another angle before even attempting to grip, as it determined that it was approaching from an incorrect angle. In a single experiment, the free s-bot lost sight of the connection slot and was manually replaced on its starting position.

7.5 Discussion and Conclusions

In this chapter, we showed how a self-propelled self-assembling robotic system can control the direction of new connections. We developed low level platform-specific behavioural control to enable one s-bot to invite a connection from another s-bot with a particular angle of connection. We established the reliability and precision of this so called directional self-assembly mechanism in a repeated set of real-world experiments.

However, in order to use the work of this chapter to form morphologies made up of many individual robots, additional higher level logic is needed that uses local directional self-assembly to generate specific global morphologies. In the next chapter, we demonstrate a higher level morphology control logic that is fully distributed.

Chapter 8

Distributed Morphology Growth

In this chapter, we present research that resulted in the first real-world demonstration of distributed morphology growth using a self-propelled self-assembling system. We present *swarmorph* — a distributed morphology control system. As discussed in Chapter 1, self-assembling multi-robot systems potentially have the flexibility to solve diverse tasks. However, in order to leverage this potential, the system must be able to form appropriate morphologies. Figure 8.1 shows three examples of different self-assembled robotic entities with morphologies suited for different tasks.

The approach we elaborate in this chapter is completely decentralised. Decentralised control is considered beneficial in multi-robot systems, as it tends to enhance system scalability and robustness. In our approach, robots that join a growing morphology use local morphology extension rules to determine how to extend the morphology appropriately. As these rules are based only on local sensing, the class of morphologies we can form using this technique is limited to periodically repeating morphologies. Nonetheless, we show that we can form a wide range of morphologies using a limited set of morphology extension rules. By using decision making based on local sensing, we can also increase the period of repetition to generate more complex morphologies.

Our approach uses the directional self-assembly mechanism from the previous chapter as a building block. This directional self-assembly mechanism is specific to the swarm-bot platform. Swarmorph, however, is encoded at a level of abstraction that does not contain any platform specific details. This abstraction is achieved by treating the directional self-

Figure 8.1: Three examples of robotic entities self-assembled into morphologies appropriate for the task. Left: A connected robotic entity crosses a trough. A line formation is well-suited to this task, since it allows the entity to stretch further and minimises the number of robots suspended over the trough. Centre: A more dense structure provides greater stability for rough terrain navigation. Right: A pushing morphology appropriate for moving heavy objects.

assembly mechanism as a 'black-box' infrastructure component. Thus swarmorph rules and control logic could rapidly be ported to another system. The only requirement would be to implement the directional self-assembly mechanism for the target system.

We first discuss our methodology in more detail (Section 8.1). We then present the morphology extension rules that we developed, and show how they can be combined to make different morphologies (Section 8.2). We present a series of experiments with real robots in which we grow four example morphologies (Section 8.3). Finally, we present some simulation based experiments to test the scalability of the approach (Section 8.4), before concluding (Section 8.5).

8.1 Methodology

In contrast with many of the algorithmic approaches discussed in Chapter 2, our approach does not involve any explicit reference to a blueprint of the global morphology being formed. Instead, we rely on self-organised growth to create global morphologies through the repeated application of strictly local morphology extension rules.

We maximise decentralization by executing independent robotic behavioural controllers on each of the robots and by restricting inter-robot communication to strictly local colour-based communication. Swarmorph relies on a limited set of local morphology extension rules. These rules instruct robots that are already part of the morphology how (in what direction) and when to extend the morphology. To carry out the actual extension dictated by a particular morphology extension rule, the s-bots use the directional self-assembly mechanism described in the previous chapter. A rule can either trigger morphology extension in a fixed direction or the direction can depend on what the robot see in its immediate surroundings. Using these decentralised rules, we can generate a large number of different morphologies. At the beginning of each experiment the rules specific to the particular morphology being formed are copied onto each of the robots.

The robots have no fixed IDs from the onset of an experiment. Instead, the robots specialise when they connect to a morphology. Morphology growth starts when an initial *seed* s-bot initiates morphology growth by opening the first connection slot. The seed s-bot has its own particular morphology extension rules. All other extending s-bots (the seed can also be considered an extending s-bot) execute a homogeneous set of morphology extension rules. Thus, for each morphology we use two separate rule sets — one for the seed and one for all other extending s-bots.[1]

During morphology growth, all non-attached robots search for connection slots. When a robot finds a connection slot, it attaches at the specified location with the corresponding specified orientation. Once connected, the newly attached robot uses its LEDs to signal its successful attachment to the robot displaying the connection slot. Both robots are then free to open new connection slots based on the local morphology extension rules specific to the morphology being formed.

Figure 8.2 shows some of the morphologies we have generated using swarmorph.

[1] In this chapter, we pre-specify the seed robot to simplify experimentation. However, this is not a strict requirement for our approach. In Chapter 9 we show how homogeneous control can be achieved by letting the first s-bot to meet an obstacle become the seed.

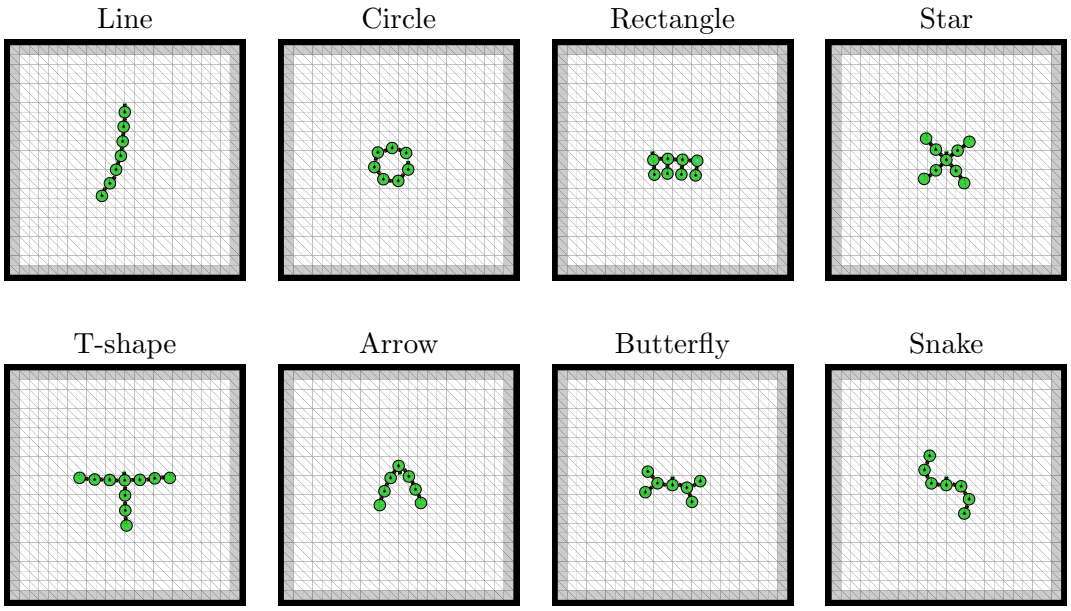


Figure 8.2: Examples of different morphologies that can be made using swarmorph. These morphologies have been generated in simulation.

8.2 Morphology Extension Rules

In this section, we discuss the set of morphology extension rules we have implemented and show how they can be used to generate specific morphologies.

Linear Morphologies — Rule: `Extend`

The `Extend` rule uses directional self-assembly to extend the structure by opening a specified connection slot. The rule takes as a parameter the connection slot that should be opened. Using just this morphology extension rule we can already form simple morphologies like the line and the circle in Figure 8.2. For the line morphology, each connected s-bot just executes the rule `Extend( B )`, while for the circle morphology each connected s-bot executes the rule `Extend( BL )`.

Branching Morphologies — Rules: `Send HS Sig, Wait HS Sig`

The rules `Send HS Sig` and `Wait HS Sig` together make up the handshake signal. The handshake signal allows an attaching s-bot to communicate to an extending s-bot that it has successfully attached to the extending s-bot's open connection slot. This communication is required by the more complex branching morphologies that require a single s-bot to extend the local structure in more than one direction. To do this, the extending s-bot must know when a connection slot has been filled, so that it can open the next connection slot at the right time (an s-bot can only open one connection slot at a time).

The s-bot hardware does not include any dedicated sensors to detect when it has been gripped by another s-bot. We designed the handshake signalling mechanism to compensate for this absence. The handshake mechanism is composed of two morphology extension rules, one executed by the sender of the handshake signal, and one executed by

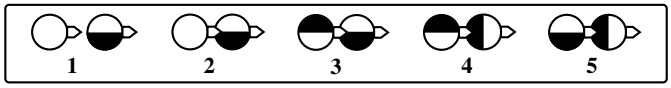


Figure 8.3: Handshake rules: `Send HS Sig, Wait HS Sig`. (1): The extending s-bot is executing the `Wait HS Sig` rule. (2): The attaching s-bot successfully grips the extending s-bot. (3): The newly connected s-bot executes the `Send HS Sig` rule (4): The extending s-bot recognises the signal and executes its next rule, to open connection slot L. (5): The `Send HS Sig` rule times out and the connected s-bot executes its next rule, to open connection slot B.

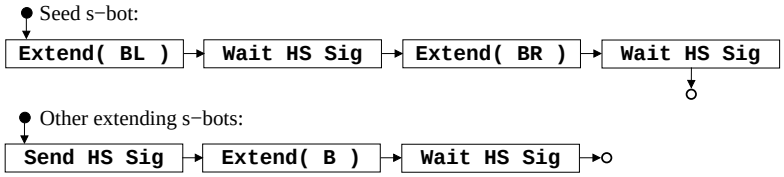


Figure 8.4: Extension rules: Arrow morphology

the receiver of the handshake (see Figure 8.3). The `Send HS Sig` rule tells the attaching s-bot to signal to the extending s-bot after it grips successfully. The signal takes the form of opening connection slot F. The `Wait HS Sig` rule tells the extending s-bot to wait until it detects the handshake signal before executing subsequent morphology specific morphology extension rules.

The `Send HS Sig` and `Wait HS Sig` rules provide the flexibility to create many more morphologies, such as the star, arrow and T-shape morphologies in Figure 8.2. Figure 8.4 shows the rule sequences required for a simple arrow morphology (more sophisticated behavioural control logic to generate the same morphology while enforcing balanced growth is presented in Section 8.3.1.3).

Symmetrically Growing Morphologies — Rule: `Balance`

The `Balance` rule enforces balanced morphology growth for as long as the extremities of the morphology are within visual range of each other. When we use the `Send HS Sig` and `Wait HS Sig` rules to generate morphologies such as the star, arrow or T-shape, we run the risk that the morphology will be unbalanced. In the arrow morphology, for example, it is stochastically possible that one arm of the arrow will continue to develop while the other arm does not develop at all. Such imbalances are particularly problematic when the number of s-bots is limited, as is the case in our real robot experimentation.

To solve this problem, we introduce the `Balance` rule. When executing this rule, a connected robot waits with its LEDs unilluminated until it cannot see any connection slots around it. In practice this means that the s-bot waits until it cannot see any green or blue LEDs. As soon as this is the case, the connected s-bot continues executing subsequent morphology extension rules.

The `Balance` rule allows parts of the morphology that are in visual range of each other to grow at the same rate. In the arrow morphology, for example, as long as the s-bots at the ends of the two arms of the arrow can still see each other, the two arms will differ in length by at most one s-bot.

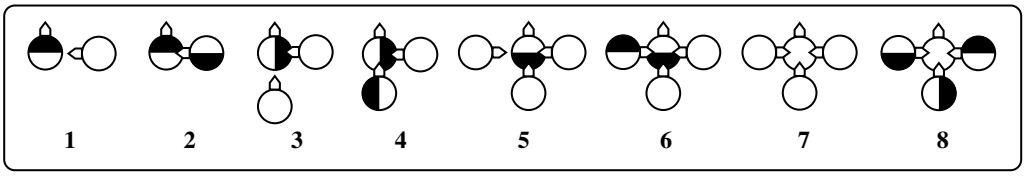


Figure 8.5: T-shape morphology growth. Use of `Balance` rule. The seed s-bot opens connection slots R,B,L in turn. The connected s-bots each execute the `Balance` rule, and therefore wait until they can see no green or blue LEDs before executing subsequent extension rules. The result is that once the seed s-bot is no longer displaying an open connection slot, all three connected s-bots open connection slot B at the same time.

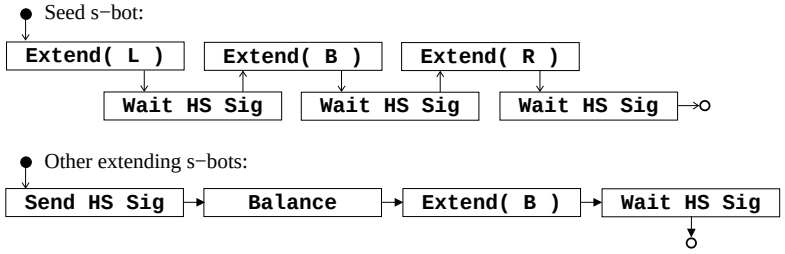


Figure 8.6: Morphology extension rules: Balanced T-shape morphology.

An example use of the `Balance` rule can be seen in Figure 8.5, that shows the growth of a balanced T-shape morphology. The rule sequences used to create this balanced growth are shown in Figure 8.6.

Morphologies with higher repetition period — Rule: `Decide`

More complex morphologies can be achieved if we allow the robots to make conditional decisions about how to extend the local structure based on local sensing. The `Decide` rule allows s-bots to make conditional decisions about how to extend the local structure based on what it can see in its surroundings. In particular, this allows an attaching s-bot to modify its behaviour based on the post handshake actions of the extending s-bot to which it is attaching.

In Figure 8.7, the extending s-bot in scenario A (left) is executing a different rule sequence from the extending s-bot in scenario B (right). In both scenarios, the attaching s-bot is executing the same rule sequence. The attaching s-bot uses the `Decide` rule to determine the appropriate post-handshake behaviour based on the post-handshake actions of the extending s-bot. The use of the `Decide` rule in the growth of the rectangle morphology is detailed in Section 8.3.1.4.

8.3 Experiments: Generating Morphologies With Real Robots

We selected four morphologies to construct in experiments with seven real s-bots. Examples of the four morphologies are shown in Figure 8.8. We first describe the distributed behavioural control logic used to grow each morphology. We then describe our experimentation strategy and present the results of the experiments.

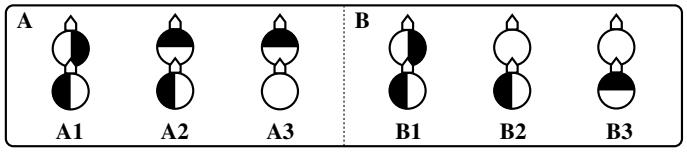
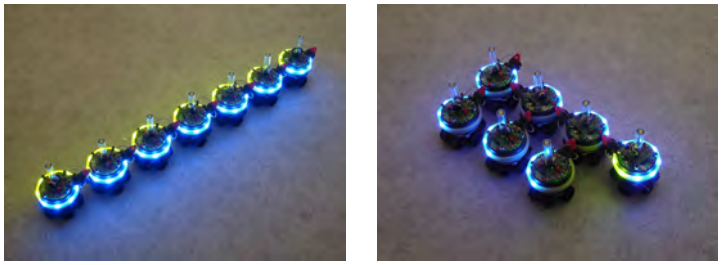
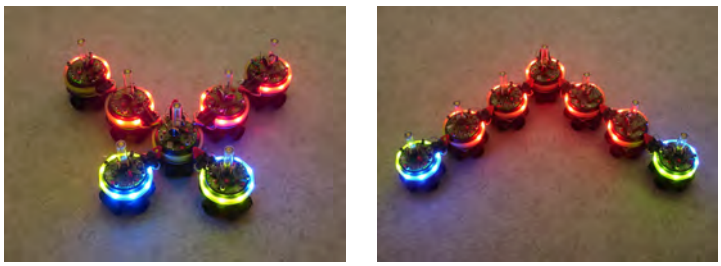


Figure 8.7: `Decide` rule. (A1,B1): Attaching s-bot successfully grips extending s-bot. (A2): Extending s-bot recognises handshake signal and executes subsequent rule which is to open connection slot R. (B2): Extending s-bot recognises handshake signal and has no subsequent rules. (A3): Attaching s-bot executes decision rule — sees green ahead (i.e., sees an open connection slot ahead) and therefore 'decides' not to execute any subsequent rules. (B3): Attaching s-bot executes decision rule — sees no green ahead (i.e., does not see an open connection slot ahead) and therefore 'decides' to open connection slot R.



Line Rectangle



Star Arrow

Figure 8.8: Four different morphologies constructed with 7 real robots: line, rectangle, star and arrow. (The rectangle and star morphologies would become symmetric with the addition of more robots.)

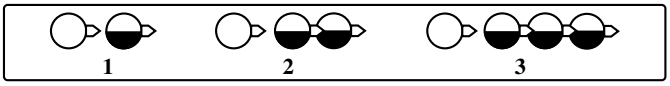


Figure 8.9: Line morphology growth. (1): An attaching s-bot approaches the seed's open connection slot. (2): The newly connected robot opens connection slot B, and another attaching s-bot approaches. (3): The newly connected s-bot again opens connection slot B and the morphology growth repeats itself.

Figure 8.10: Morphology extension rules: Line morphology.

8.3.1 The Four Morphologies

8.3.1.1 Line Morphology

Line morphology growth is shown in Figure 8.9. The extension rules for the line morphology are shown in Figure 8.10. In this morphology, as there is no handshaking, the connection slot LEDs remain illuminated. This has the ancillary benefit that free s-bots are naturally guided along the morphology towards the extending s-bot at the end of the line. This is because a free s-bot perceives the LEDs of all of the connected s-bots as a single composite connection slot. The standard directional self-assembly navigation logic thus directs the free s-bots around the whole connected morphology.

8.3.1.2 Balanced Star Morphology

The extension rules for the balanced star morphology are shown in Figure 8.11. Figure 8.12 shows the growth of a balanced star morphology.

The s-bot camera can perceive LEDs farther away (up to 50 cm) than the proximity sensors can detect obstacles (up to 15 cm). To facilitate star morphology growth, we therefore let s-bots that are part of the morphology, but that are not displaying an open connection slot illuminate some of their red LEDs. This makes them visible to free robots, and thus helps to guide the free robots around the morphology to an open connection slot. This results in faster morphology growth, especially when the number of free robots is limited.

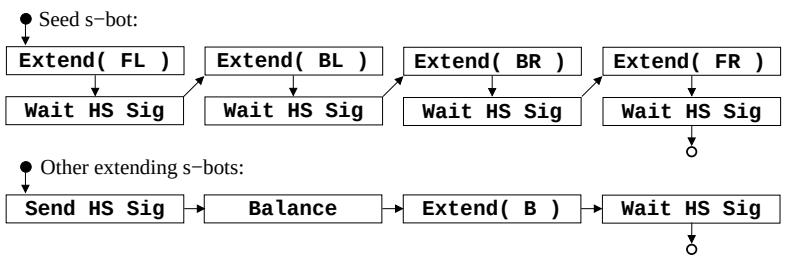


Figure 8.11: Morphology extension rules: Balanced star morphology.

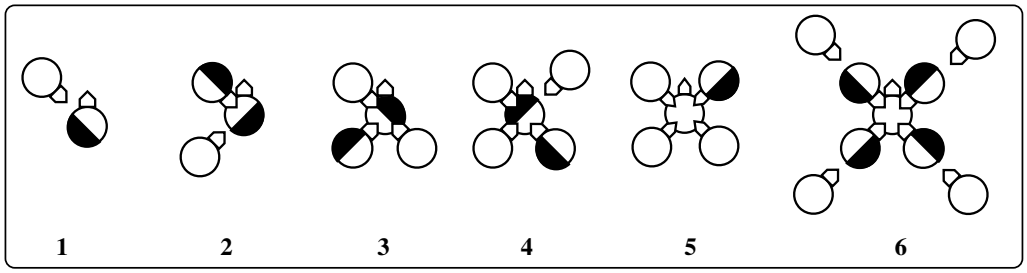


Figure 8.12: Balanced star morphology growth. (1-4): The seed s-bot opens connection slots FL,BL,BR,FR in turn. The connected s-bots each execute the `Balance` rule, and therefore wait until they can see no green or blue LEDs before executing subsequent extension rules. (5-6): The result is that once the seed s-bot is no longer displaying an open connection slot, all four connected s-bots open connection slot B at the same time.

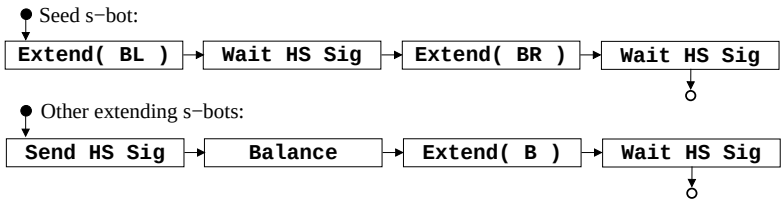


Figure 8.13: Morphology extension rules: Balanced arrow morphology.

8.3.1.3 Balanced Arrow Morphology

Balanced arrow morphology growth follows a similar pattern to the balanced star morphology shown in Figure 8.12, differing only in that the seed does not open connection slots FR and FL. The extension rules for the arrow morphology are shown in Figure 8.13. As in the balanced star morphology, we let connected robots without an open connection slot illuminate some of their red LEDs to guide free robots around the morphology.

8.3.1.4 Rectangle Morphology

Rectangle morphology growth is shown in Figure 8.14. The extension rules to grow the rectangle morphology are shown in Figure 8.15.

In the rectangle morphology, each newly connected s-bot decides how to extend the morphology based on what it can see in its immediate vicinity. If the newly connected robot can see a connection slot open after the handshake, it does not extend the morphology (since there is another s-bot already extending the morphology). If, on the other hand, the newly connected s-bot cannot see a connection slot open after the handshake, it extends the morphology first by opening connection slot L, then by opening connection slot B.

In the rectangle morphology, connected robots without an open connection slot illuminate some of their blue LEDs to guide free robots around the morphology. The use of blue LEDs is more efficient than red LEDs for this purpose, as it allows free s-bots to distinguish between the s-bots that are part of the morphology and other free s-bots with illuminated red LEDs. We used red LEDs rather than blue LEDs in the balanced star and balanced arrow morphologes above, since in these morphologies more than one connection slot can be open, and the illumination of extra blue LEDs would cause the free robots to

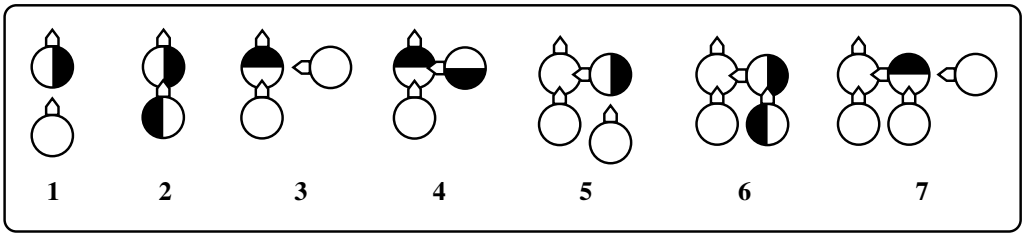


Figure 8.14: Rectangle morphology growth. (1): Seed opens connection slot B (2): Connected s-bot sends handshake signal. (3): Seed sees handshake signal, then opens connection slot R. First connected s-bot executes `Decide` rule — sees connection slot and therefore executes no subsequent extension rules. (4): Another s-bot connects and handshakes. (5): Seed sees handshake signal but does not open another connection slot. Connected s-bot executes `Decide` rule — does not see a connection slot, and therefore opens connection slot L itself. (6): Another s-bot connects and handshakes. (7): Morphology growth repeats.

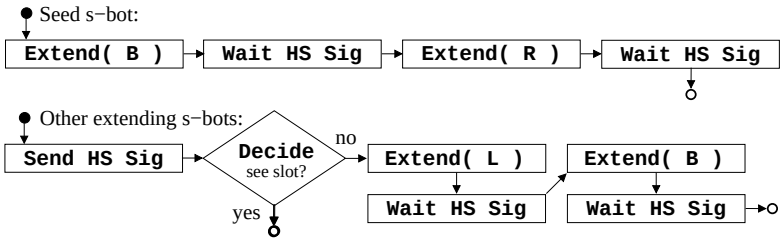


Figure 8.15: Rectangle morphology extension rules.

perceive non-existent connection slots.

8.3.2 Results

We grew each of the four example morphologies (line, balanced arrow, balanced star, rectangle) 10 times with 7 real s-bots. Completed examples of the four morphologies are shown in Figure 8.8. Photographs and videos of experiments described in this chapter can be found in [107].

We conducted our experiments in a walled arena of 220 cm x 220 cm. In each experiment, we grew a single morphology. The free s-bots started each experiment at one of 12 points (randomly sampled without replacement) evenly distributed around a circle of radius 50 cm centered on the seed s-bot. The free s-bots were placed in one of four possible starting orientations (randomly sampled with replacement). An experiment was considered finished once all 6 free s-bots had successfully attached or after 15 minutes had expired.

All 6 free robots successfully connected to the morphology in 38 out of 40 experiments. In a single rectangle morphology experiment, one robot failed to connect, and in another of the rectangle morphology experiments, two robots failed to connect.

8.3.2.1 Timing

Figure 8.16 shows morphology growth over time. Each line in the figure represents the mean growth timing of a single morphology over ten experiments. All four morphologies

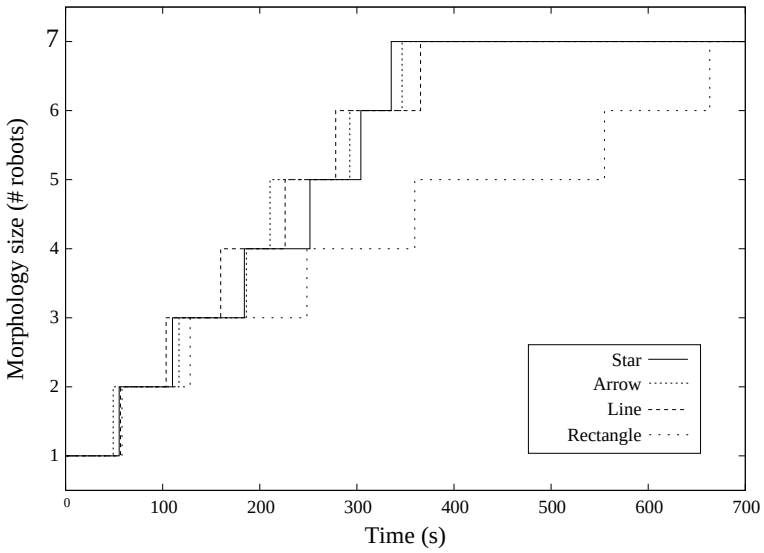


Figure 8.16: Mean morphology growth times of four specific morphologies over ten experiments. In each experiment a single morphology was constructed with seven real s-bots. Each horizontal line segments represents a time interval during which the morphology size remains constant. Vertical line segments correspond to moments at which a free s-bot connects to a morphology.

grow at a similar rate until they reach a size of three s-bots. Subsequent growth rates for the star, arrow and line morphologies remain relatively close, and the three morphologies have comparable mean completion times of 335 s, 347 s and 366 s, respectively. The rectangle morphology had a slower growth rate, with a mean completion time of 634 s.

There are two reasons for the slower growth of the rectangle morphology. Firstly, the connected robots are more densely distributed in the rectangle morphology than in the other morphologies. This means that inaccurate connections (misalignment from specified orientation or lateral displacement from specified grip point) can render subsequent connections difficult or in some cases even impossible. The rectangle morphology is also the only morphology to rely on conditional morphology extension (with the use of the `Decide` rule). In two separate rectangle morphology experiments the decision procedure failed — the attaching robot incorrectly determined its location in the structure. In both of these experiments, however, the system proved sufficiently robust to recover from the failure. Morphology growth continued and the morphology subsequently reassumed a rectangular shape.

A different breakdown of the timing results can be seen in Figure 8.17. Every successful connection from every experiment is individually represented. In the line morphology, the time intervals between connections is largely consistent. This is because the number of open connection slots remains constant in the line morphology (there is always one open connection slot). Later connections become slightly slower, as the number of free robots diminishes.

In general, the incremental speed of morphology growth is dependent on the ratio of open connection slots to free robots. During arrow morphology growth, for example, there are usually two connection slots open (after the first two robots have connected). This allows two free s-bots to approach the connection slots at the same time (parallel

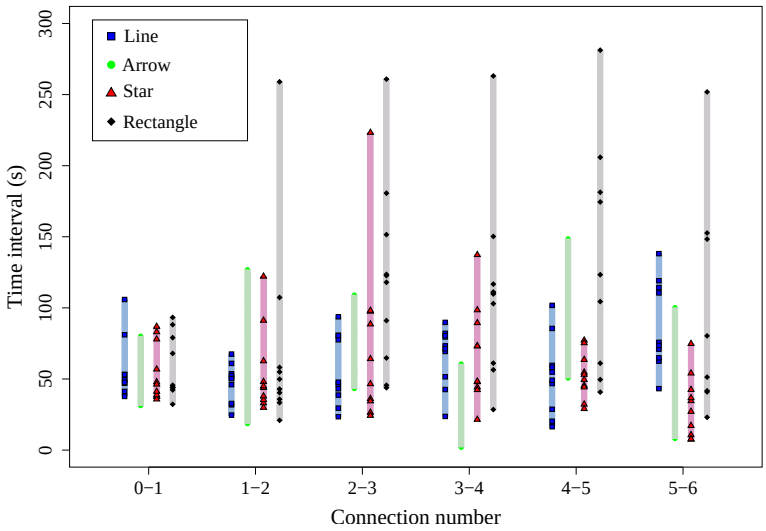


Figure 8.17: Time intervals between connections. Each point represents a time interval between two successive connections in a single trial. There is one point for every successful connection in every trial. The time interval between connections 0 and 1 (0-1) is the time from the start of the trial to the time of the first connection. The coloured bars indicate the range of values for a given morphology and time interval.

execution is not, of course, guaranteed as it depends on stochastic s-bot movement). This dynamic can be seen in Figure 8.17 by the short time intervals between arrow morphology connections 3-4. A similar mechanism explains the short time intervals between star morphology connections 4-5 and 5-6. In the star morphology, once the fourth robot has connected, all four connected robots open connection slots. This allows the fifth and sixth robots to find an open connection slot quickly and in parallel.

8.4 Experiments: Scalability

To investigate the scalability properties of our morphology growth scheme, we conducted experiments with larger numbers of s-bots in simulation. Our simulation environment consists of a specialised software simulator [18]. The simulator allows behavioural control code developed for the real s-bots to be run without any modification directly on the simulated s-bots.

8.4.1 Simulation Verisimilitude

To verify the verisimilitude of our simulation environment, we repeated in simulation the experiments that we had already done with the real robots. We set up an experiment with 7 simulated s-bots for each of the four morphologies tested on real s-bots in the previous section. Table 8.1 lists the mean completion times for 10 real robot experiments for each morphology and the results obtained in 100 replications in simulation for the same morphologies. For the star, arrow and line morphologies the mean completion time in simulation is slightly lower than the mean completion time observed in the experiments with real robots. The mean completion time for the rectangle morphology is 31% lower in simulation than observed in our experiments with real robots. We believe that this

Table 8.1: Mean completion times for different morphologies on real s-bots and in simulation.

	Real s-bots	Simulated s-bots	Difference
Star	335.32 s	325.31 s	−3.0%
Arrow	346.56 s	331.37 s	−4.2%
Line	365.53 s	339.76 s	−7.0%
Rectangle	663.46 s	456.71 s	−31.2%

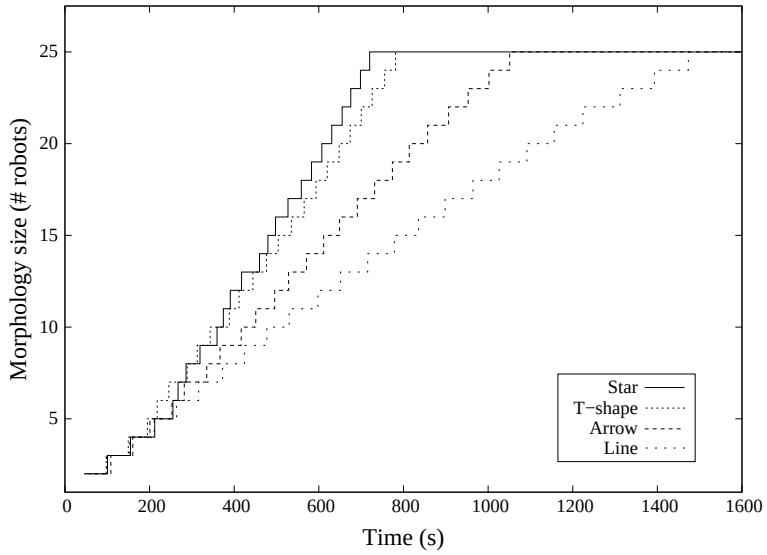


Figure 8.18: Morphology growth over time for four morphologies formed with simulated robots. We fix the number of free s-bots in the system at 10, by feeding a new s-bot into the simulation after every connection. Each line represents the mean growth timings of a single morphology over 100 experimental trials.

difference is primarily due to the fact that when the robots are densely packed, the camera is more accurate in simulation than in reality. In the rectangle morphology, the robots in the rectangle morphology are located close to each other giving rise to effects such as occlusions and reflections of the light emitted from the LEDs. These effects are not simulated by our simulation environment. In our scalability testing in the next section we do not, therefore, use the rectangle morphology or any other densely packed morphologies.

8.4.2 Scalability Performance

In order to test how fast various morphologies grow when more robots are available, we conducted experiments with four different morphologies. We chose the morphologies line, arrow, T-shape and star, because they respectively can have one, two, three and four connection slots open simultaneously (see Figure 8.2). We kept the number of free robots at a constant count of 10. Each time a free robot connected to the morphology a new free robot was added to the simulation. For each morphology we conducted 100 trials. In each trial, we varied the initial placement and orientation of the free s-bots. A trial was considered finished when the morphology reached a size of 25 connected robots.

Figure 8.18 shows the mean connection times for the different morphologies. The growth profiles of the morphologies are similar until the fifth robot attaches to the morphology, at which point they start to differ. The star morphology can have up to four connection slots open simultaneously and on average reaches a size of 25 robots in 720 s. The T-shape morphology can have up to three connection slots open simultaneously and reaches a size of 25 robots in 781 s. The relatively small difference (61 s) between the average completion times is probably due to the fact that the number of free robots is fixed to 10. In order for the extra open connection slots to speed up morphology growth, there have to be robots in the vicinity each time a new connection slot is opened. Since the 10 free robots are not always evenly distributed, the star morphology completes only marginally faster than the T-shape morphology.

Comparing the completion times for the star and the T-shape morphologies with the arrow and line morphology, the benefit of multiple open connection slots becomes more apparent: The arrow morphology (which has up to two connection slots open simultaneously) and the line morphology (which has only one connection slot open at a time) reach the size of 25 robots in 1,051 s and 1,473 s, respectively. For the line morphology, the rate of growth slows down as the morphology gets larger. The line morphology only has one connection slot open at a time, and because of its one dimensional growth, it has a larger span than any other morphology. Thus, even once a free s-bot has approached the connected structure, it can take a long time for the s-bot to navigate to the open connection slot. This is not the case for the other morphologies, partly because more connection slots are open and partly because they grow in two dimensions.

8.5 Discussion and Conclusions

The work presented in this chapter contributed to self-assembly research by providing the first real-world demonstration of specific morphology growth on a self-propelled self-assembling robotic platform. This is an important milestone, as self-assembling systems must be able to generate task-specific morphologies if they are ever to (selectively) replace more traditional monolithic robotic systems.

In our proposed approach — swarmorph — a specific morphology is determined by a set of local morphology extension rules built on top of the directional self-assembly mechanism presented in Chapter 7. Swarmorph is potentially platform agnostic, as the component rules are abstracted away from platform specific implementation. We conducted a series of real-world experiments with up to seven robots, and achieved a high success rate in building four different morphologies. In further simulation based experiments, we showed that the approach scales well.

Although the approach presented in this chapter allows for the self-organised growth of morphologies, it has a number of limitations with respect to our goal of giving the robots the capacity to respond to their environment by self-assembling into task-specific morphologies:

- The seed robot is predetermined: the seed robot was determined before the start of each experiment, and it was given the subset of rules specific to the seed of a given morphology. Ideally, a robot should become a seed as a response to encountered obstacles.

- Only one morphology can be built in each experiment: the seed robot and the free robots are given a single set of rules for a specific morphology at the onset of each

experiment. There is no way for the robots to decide which morphology to grow during the experiment as, for instance, a response to different types of obstacles.

- The morphologies that can be generated are limited to simple repeating structures. The sophistication of the morphologies that can be grown is dependent on the available morphology extension rule set.
- There is no way to stop morphology growth: there is no way for robots forming a morphology to stop after a certain number of robots have attached to the morphology. The balance primitive could be used to limit the morphology growth, but only for small and dense morphologies in which all robots are within visual range. For larger morphologies, growth continues until there are no more free robots.

In the next chapter, we show how symbolic communication between connected s-bots can help to overcome the above limitations.

Chapter 9

Scripted Generation of Arbitrary Morphologies

A practically useful morphology generation system should be able to form arbitrary morphologies. In other words, the system should be capable of forming any morphology that the hardware of the system allows. We would also expect such a system to be able to control the size of the morphologies generated — at a certain point morphology growth must stop to allow the new self-assembled robotic entity to carry out its task. Finally, we would expect the generation of multiple morphologies to be possible — we would like separate self-assembled robotic entities to be able to carry out tasks in parallel. To cross an obstacle, for example, a specific morphology of three robots may be sufficient. If we had many hundreds of robots trying to cross the obstacle, it would normally make more sense to form many distinct morphologies of three robots rather than a single giant morphology.

In the rule based system presented in the previous chapter, connected robots followed local morphology extension rules to extend the local structure by inviting new connections in a particular direction. However, the absence of symbolic communication in combination with the homogeneity of the robotic behavioural controllers, meant that each newly connected robot had to follow the same morphology extension rules, with the result that only periodically repeating structures were possible.

In this chapter, we retain the paradigm of morphology growth through local pattern extension. However, to enable the desirable system properties listed above, we make two key changes to the system. Firstly, we abstract our system into a set of behavioural control instructions that form the basis of a new morphology creation language (swarmorph-script) that can be executed on real robots. Secondly, we augment the system's communication capabilities to enable symbolic communication between physically connected robots.

The abstraction of morphology generation logic into a scripting language has several advantages. Most importantly, it massively reduces the development cycle for the design of new morphologies. Morphology control logic can be prototyped in simulation, then rapidly changed and tested again either in simulation or on real robots. Modifying morphology generation logic on the real robots can be done by changing a text file on the robot itself, without going through the off-line recode, compile and upload cycle. Morphology generation logic also becomes much more accessible to non-expert users. The scripts presented in this chapter are almost direct transcriptions of the scripts actually run on the real robots. To understand the control logic, one doesn't need an expert user to diagrammatically represent the logic (as was the case in the previous chapter), or to examine C++ source code. Instead, a well commented script is almost self-explanatory.

Finally, the scripting paradigm forces a strictly modular approach. Errors in one script instruction can be tracked down and fixed independently, and as the language expands, one can be confident that the existing instructions work reliably, as they have been used and tested in many previous scripts.

This chapter is organised as follows. In Section 9.1, we present an overview of the swarmorph-script morphology generation methodology. In Section 9.2, we present an initial set of script instructions that enable branching and communication, and show how these instructions can be combined to generate arbitrary morphologies. In Section 9.3, we present a further set of instructions that enable the creation of multiple morphologies through morphology splitting. Finally, in Section 9.4, we discuss the contribution of the research in this chapter, and point the way forward to the work presented in subsequent chapters.

9.1 Methodology

We retain the paradigm of global morphology growth through directed local extension presented in Chapter 8. Using swarmorph-script, each robot still operates completely autonomously and independently. Robots that are not attached to a morphology do not know the state of the global morphology. They cannot, therefore, deduce where they should attach. Instead, robots that are already part of the morphology use their coloured LEDs to indicate where and with which orientation new robots should connect in order to extend the morphology correctly.

When a new robot connects to the morphology, it initiates symbolic communication with the robot to which it connected. Through this communication, the newly connected robot receives instructions about how to extend the local structure. Following these instructions, the newly connected robot in turn attracts other robots by lighting up its own coloured LEDs. When a subsequent new robot attaches, it once again initiates communication, and is told in turn how to extend the structure. As this process repeats itself, the morphology grows accordingly.

Arbitrary morphologies can thus be created by communicating information about the local state of the morphology (communication of a single number suffices) to each new robot that connects to the structure. The newly connected robot can extend the local structure appropriately based on this information, then pass on updated state information to the next robot to attach. A similar mechanism allows for morphology size regulation — a counter can be incremented and passed on by each attaching robot. Once the counter has reached a certain threshold, the next robot can stop extending the morphology.

Generation of multiple morphologies is enabled by adding ‘disconnect’ and ‘retreat’ instructions to our morphology creation language. A robot that is part of a finished pattern can spark the creation of a new morphology by inviting a new connection to the already finished morphology. Once a robot has attached to this new connection, instructions to ‘disconnect, retreat and start a new morphology’ are communicated to the newly attached robot.

9.2 Behavioural Control with Swarmorph-script

In this section, we show how swarmorph-script can be used to implement distributed, homogeneous behavioural control that results in the creation of arbitrary morphologies of specific sizes. We first present the instructions that enable communication and control flow

branching, and present an example script to illustrate the use of these instructions. We go on to show how homogeneous behavioural control can be implemented in swarmorph-script using obstacle based seeding. Finally, we present results of experiments in which we generate such morphologies with real robots.

9.2.1 Directional Self-Assembly

We continue to use the directional self-assembly mechanism (see Chapter 7) as the basis for the morphologies we generate using swarmorph-script. Below, we describe the two swarmorph-script instructions that we have implemented to incorporate the distributed behavioural control of directional self-assembly into our morphology control language.

9.2.1.1 Instructions:

`InviteConnection(` *`[connection slot]`* `)`, `FindSlotThenConnect()`

The command `InviteConnection` opens a connection slot. The slot that should be opened is given as a parameter to the command. This parameter can take the following values: `front-left`, `left`, `back-left`, `back`, `back-right`, `right`, `front-right`. Once an extending s-bot has opened a connection slot, it waits until another s-bot has connected to the connection slot before executing subsequent instructions in its control script.

When an s-bot executes `FindSlotThenConnect`, it tries to navigate to the nearest open connection slot and attempts to connect. To do so, it follows the navigation logic for free s-bots specified in Chapter 7. After a successful grip, it continues to execute subsequent instructions in its control script.

9.2.2 Communication and Control Flow

When an s-bot connects to a connection slot, the s-bot displaying the connection slot typically communicates to the newly attached s-bot how to extend the local structure. The most efficient way of doing this (that requires the least communication) is for the sending s-bot to transmit a single number — the sequence identifier. The receiving s-bot receives the number, then it follows the behavioural control logic (a sequence of script instructions) associated with that identifier. On the receiving s-bot this association between the identifier received and the corresponding sequences of swarmorph-script instructions is achieved using dedicated control flow branching instructions (explained below).

9.2.2.1 Communication Instructions:

`SendInstrSeqId(seq-id)`, `ReceiveInstrSeqId()`

Our morphology generation approach relies on local communication between physically connected robots. The instructions `SendInstrSeqId` and `ReceiveInstrSeqId` respectively send and receive a number identifying a predefined sequence of instructions.

The s-bot has no hardware dedicated to local point-to-point communication and we have therefore implemented a simple protocol based on visual communication: We use the three colours red, green and blue. Each time a bit is transmitted, the sending robot changes the illumination of its LEDs. The colour green represents a '0' bit, blue represents a '1' and red represents a repeat bit. We rely on acknowledgement to distinguish adjacent bits. The receiver acknowledges receipt of each bit by lighting up its LEDs to match the colour of the sender's LEDs. Once the receipt of a bit has been acknowledged, the

sender transmits the next bit. This acknowledgement mechanism necessitates our use of the dedicated colour for a repeat bit.

When transmitting a substring of two or more bits of the same value, every other bit will be represented by the colour red (starting from the second bit). As an example, assume that the sender transmits the binary string '00': It first lights up its green LEDs to send the first '0'. When the receiver has acknowledged receiving '0' by lighting up its green LEDs, the sender lights up its red LEDs to indicate that the new bit has the same value as the previous one. In this way, every new bit causes a change in colour that can be acknowledged by the receiver. If the sender transmits the binary string '111', it first lights up its blue LEDs to communicate the first '1', then its red LEDs to communicate the second '1', and finally blue again to transmit the third '1'.

To indicate the end-of-transmission, the sender turns off its LEDs. When the receiving robot detects end-of-transmission, the receiver performs a simple validity check by counting the number of bits received. We transmit each number as a sequence of 5 bits. This allows us to perform a simple error check — the receiver checks at the end of a transmission that the number of bits received is a multiple of 5. The receiver illuminates its green LEDs if the length check indicated that the string was successfully received. Otherwise it illuminates its blue LEDs to request retransmission.

We tested the communication mechanism on real robots (see [24] for a video showing the communication mechanism in action) and it proved reliable but relatively slow: We transmitted strings of 100 bits 10 times. In all 10 trials, the string was successfully communicated and on average the transmission took 121 seconds resulting in an average bandwidth of 0.8 bits/second. On other robot platforms more efficient means for local and situated communication [138] might be available, such as infra-red [124] or communication through the mechanism that connects different modules, see for instance [99]. For our purposes, however, the visual communication protocol suffices to implement the swarmorph-script approach on real robots.

9.2.2.2 Control Flow Instructions: `if, then, end, StopExecution()`

We have added a simple branching construct:

if *[condition]* **then**
  `[sub-script]`
end

The `[sub-script]` is executed if and only if *[condition]* is true. The condition is typically whether or not a feature, such as a hole, has been detected in the environment.

The instruction `StopExecution` causes the execution of the script to stop.

9.2.2.3 Generic Script Structure using Communication and Control Flow

We present two example scripts (one for a pre-designated seed s-bot and one for all other robots) that use communication and branching instructions to achieve arbitrary morphology generation.

The example script for a robot that starts a morphology is shown in Script 1. The extending s-bot first opens a connection slot to its rear. When another robot connects to the connection slot, the robot executing the script sends a '1' telling the newly connected robot to execute instruction sequence 1. An example of the script for connecting robots is shown in Script 2.

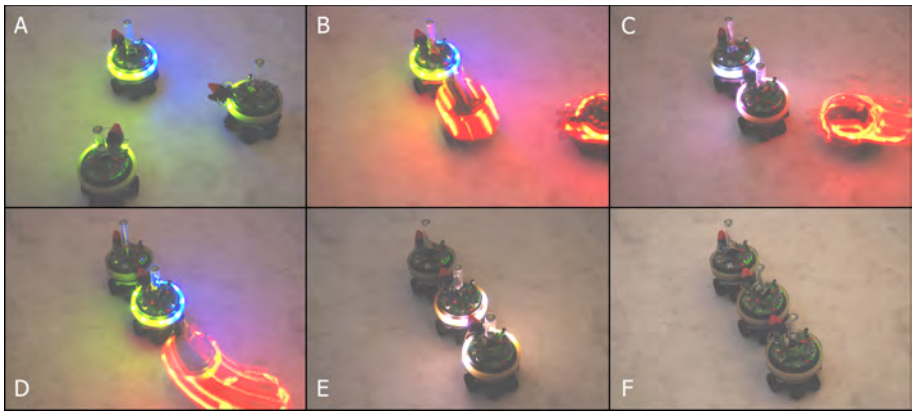


Figure 9.1: Result when one robot executes Script 1 and two robots execute Script 2. See text for details.

Script 1: Simple example of a script for a robot that starts a new morphology.

```
InviteConnection( back )      // Open a connection slot wait for an s-bot to connect
SendInstrSeqId( 1 )           // Tell the connected s-bot to execute instruction sequence '1'
StopExecution()
```

Script 2: Simple example of a script for a robot that connects to an existing morphology and extends it.

```
FindSlotThenConnect()         // Find and connect to a robot with an open connection slot.
ReceiveInstrSeqId()           // Initiate communication and receive an instruction sequence ID.
if received-seq-id = 1 then
    InviteConnection( back )      // Open a connection slot and wait for an s-bot to connect.
    SendInstrSeqId( 2 )           // Tell the connected s-bot to execute instruction sequence '2'
    StopExecution()
end
if received-seq-id = 2 then
    StopExecution()
end
```

A robot executing the script shown in Script 2 first searches for a connection slot and attempts to connect. Once a successful connection has been formed, the robot expects to receive the identifier of an instruction sequence from the robot to which it connected. If it receives a '1', it opens a connection slot to its rear and waits for another robot to connect. If it receives a '2', it calls the `StopExecution` instruction and takes no further action.

An example morphology grown with three robots is shown in Figure 9.1. One robot starts the morphology by executing Script 1 and two robots connect to the morphology by executing Script 2. The first robot opens a connection slot to its rear (Figure 9.1A) and sends sequence identifier 1 to the first robot that connects (Figure 9.1C). The first robot to connect in turn opens a connection slot to its rear, and when the connection slot is filled (Figure 9.1D), it sends sequence identifier 2 to the second (and last) robot that connects to the morphology (Figure 9.1E). The last robot does not open any connection slots and the morphology is complete (Figure 9.1F). The result is a line morphology.

9.2.3 Homogeneous Behavioural Control through Obstacle based Seeding

Giving a single robot a specific predefined role is undesirable in a distributed system, as it reduces the inherent robustness of the system. In the example scripts presented in the previous section (and in all of the morphologies generated in Chapter 8), a single robot was pre-designated as the seed. However, using a pre-designated seed means that if this single robot were to fail, the whole system would be incapable of of forming a morphology. With truly homogeneous behavioural control, the failure of a single robot is less likely to disrupt overall system performance.

To achieve homogeneous behavioural control in our experiments, we use the previously introduced branching instructions in conjunction with a new `RandomWalkUntil` instruction. A morphology is seeded when one of the s-bots detects an obstacle. This s-bot becomes the seed for the morphology and opens the first connection slot. The other s-bots see the open connection slot, and attempt to connect to the morphology. Once an s-bot has seen a connection slot, it can no longer become a seed. The 'obstacle' in our case is a dark patch on the floor of the arena representing a hole.[1] At the start of each experiment we place the s-bots pointing towards a 10 cm x 10 cm dark patch on the floor in the center of the arena (see Figure 9.2). All subsequent morphologies presented in this chapter use this type of obstacle based seeding using the `RandomWalkUntil` instruction.

9.2.3.1 Instruction: `RandomWalkUntil( [condition] )`

When a robot performs a random walk, it moves in straight lines and performs simple obstacle avoidance based on readings from its proximity sensors. The direction and length of the straight lines are chosen randomly. A robot continues to perform the random walk until the given condition is met. The exit conditions conditions we have implemented to date are *hole-detected* (the s-bot detects a hole in the ground using its infrared ground sensors) and *connection-slot-detected* (the s-bot detects an open connection slot using its camera). Once the give exit condition is met, the robot stops random walking, and executes the next instruction in its behavioural control script.

9.2.3.2 Generic Script Structure using Obstacle Based Seeding

The basic structure of any script that uses obstacle-based seeding is shown in Script 3. The s-bot performs a random walk until either a hole (the dark patch) or a connection slot is seen. If a hole is detected, the s-bot becomes the seed of the new morphology. Otherwise, if a connection slot is seen, the robot attempts to connect to the existing morphology and waits to receive a instruction sequence identifier. This number will map to a particular sequence of pre-defined instructions that will tell the robot how to extend the structure locally.

[1] In this study, our focus is on morphology control rather than navigation. We therefore use a dark patch in place of the hole to prevent robot casualties. To the s-bot's ground sensors, a dark patch on the floor is almost indistinguishable from a hole.

Figure 9.2: Starting configuration for experiments using obstacle-based seeding.

Script 3: Basic structure of a script using obstacle based seeding.

```
RandomWalkUntil( hole-detected or connection-slot-detected )
if hole-detected then
    // Become the seed of a new morphology
    InviteConnection( morphology-specific-connection-slot )
    SendInstrSeqId( morphology-specific-sequence-id )
    ...
    Continue the morphology.
    ...
    StopExecution()
end
if conn-slot-detected then
    // Connect to morphology and execute instruction sequence based on received id
    FindSlotThenConnect()
    ReceiveInstrSeqId()
    ...
    Execute morphology-specific instructions.
    ...
end
```

9.2.4 Results

In this section, and in Sections 9.3 and 9.3.3, we present results of experiments conducted with real s-bots. In each experiment, we execute a single specific script (written in the swarmorph-script language) between 6 and 11 robots. We execute the same script independently on each robot, in other words behavioural control is distributed and homogeneous. We present photographs of the resulting morphologies. Every photograph presented in these sections is the final outcome of an actual experimental trial. Videos of each trial can be seen in [24].

We demonstrate the mechanism of local communication, branching and predefined instruction sequences used to generate arbitrary morphologies. We demonstrate this mechanism with three morphologies — the *horseshoe*, the *shovel* and the *square*.

The *horseshoe, shovel and square* morphologies and the corresponding scripts we used to generate them are shown in Figure 9.3, Figure 9.4 and Figure 9.5. In one experiment we manually reset a robot that had suffered a low level software failure. Faulty proximity sensors on a few of the robots resulted in occasional robot collisions — in some cases this resulted in one of the robots toppling over. When this happened we manually righted the toppled robot. On average, a robot toppled over once for every 661 seconds of ex-

```
RandomWalkUntil( hole-detected or
                 connection-slot-detected )
if hole-detected then
    InviteConnection( left )
    SendInstrSeqId( 1 )
    InviteConnection( right )
    SendInstrSeqId( 2 )
    StopExecution()
end
if connection-slot-detected then
    FindSlotThenConnect()
    ReceiveInstrSeqId()
    if received-seq-id = 0 then
        StopExecution()
    end
    if received-seq-id = 1 then
        InviteConnection( right )
        SendInstrSeqId( 0 )
    end
    if received-seq-id = 2 then
        InviteConnection( left )
        SendInstrSeqId( 0 )
    end
end
```

Figure 9.3: *Horseshoe* script and result.

perimentation time (this figure applies to experiments presented in this Section and in Section 9.3).

In each of these experiments, whenever a new robot connects to the morphology, it receives the identifier of an instruction sequence that indicates how the local structure should be extended.

In the *horseshoe* (Figure 9.3) and the *shovel* (Figure 9.4), three different extension instruction sequences are used: one for the robot to the left of the seed (sequence identifier 1), one for the robot to the right of the seed (sequence identifier 2), and one further rule to stop the morphology growth locally. The *square* morphology is generated from three extension instruction sequences like the *horseshoe* and the *shovel* described above. However, in the *square* (Figure 9.5), we take advantage of the symmetry in the structure, and sequence identifier 2 is sent to both the third and fourth robot that attach to the seed. We could, in fact, have built a square using two extension instruction sequences only, by having the seed send sequence identifier 2 to all the robots that connect to it. The outline of the resulting morphology would be the same, but the topology of connections between the robots would be different.

9.3 Multiple Morphologies and Reconfiguration

In practical task-execution scenarios with large numbers of robots, the formation of multiple morphologies is likely to be a requirement. As we increase the number of self-assembling robots in a robotic swarm, the need for the self-assembled robots to have an appropriate morphology does not change. However, rather than generating a single giant morphology, it is more reasonable to assume that many smaller morphologies will be appropriate. This will allow parallel execution and teamwork at the level of self-assembled robotic entities. Over time, as some tasks are finished and others appear, it is also reasonable to expect that self-assembled robotic entities should be able to disassemble and subsequently re-assemble

```
RandomWalkUntil( hole-detected or
                 connection-slot-detected )
if hole-detected then
    InviteConnection( back )
    SendInstrSeqId( 0 )
    InviteConnection( left )
    SendInstrSeqId( 1 )
    InviteConnection( right )
    SendInstrSeqId( 2 )
    StopExecution()
end
if connection-slot-detected then
    FindSlotThenConnect()
    ReceiveInstrSeqId()
    if received-seq-id = 0 then
        StopExecution()
    end
    if received-seq-id = 1 then
        InviteConnection( right )
        SendInstrSeqId( 0 )
        InviteConnection( back-left )
        SendInstrSeqId( 0 )
    end
    if received-seq-id = 2 then
        InviteConnection( left )
        SendInstrSeqId( 0 )
        InviteConnection( back-right )
        SendInstrSeqId( 0 )
    end
end
```

Figure 9.4: *Shovel* script and result.

```
RandomWalkUntil( hole-detected or
                 connection-slot-detected )
if hole-detected then
    InviteConnection( front-left )
    SendInstrSeqId( 1 )
    InviteConnection( back-left )
    SendInstrSeqId( 0 )
    InviteConnection( back-right )
    SendInstrSeqId( 2 )
    InviteConnection( front-right )
    SendInstrSeqId( 2 )
    StopExecution()
end
if conn-slot-detected then
    FindSlotThenConnect()
    ReceiveInstrSeqId()
    if received-seq-id = 0 then
        StopExecution()
    end
    if received-seq-id = 1 then
        InviteConnection( left )
        SendInstrSeqId( 0 )
        InviteConnection( right )
        SendInstrSeqId( 0 )
    end
    if received-seq-id = 2 then
        InviteConnection( left )
        SendInstrSeqId( 0 )
    end
end
```

Figure 9.5: *Square* script and result.

into different morphologies.

In this section, we first present the morphology splitting instructions that allow for the creation of multiple morphologies. We present example experiments with real robots in which multiple copies of the same morphology are generated. We go on to describe the more sophisticated coordination mechanisms required to enable reconfiguration, the instructions that enable this coordination, and an experiments in which real and simulated robots autonomously reconfigure into sequences of different morphologies.

9.3.1 Morphology Splitting to Generate Multiple Morphologies

The generation of multiple morphologies in swarmorph-script is enabled through a morphology splitting mechanism. Once a morphology has finished forming, a robot that is part of a finished pattern can spark the creation of a new morphology by opening a new temporary connection slot. A free robot becomes part of the morphology, but only temporarily, because the extending robot sends a sequence identifier to the newly connected robot that tell it to split off from the morphology, retreat and start a new morphology from scratch.

9.3.1.1 Instructions: `Disconnect()`, `Retreat(time-out)`

The instructions `Disconnect` and `Retreat` respectively allow a robot connected to a morphology to disconnect (by opening its gripper) and to move backwards for a period of time. In this way, when a morphology is complete, a robot in the morphology can open a temporary connection slot and instruct the robot that connects to the temporary slot to disconnect and start a new morphology.

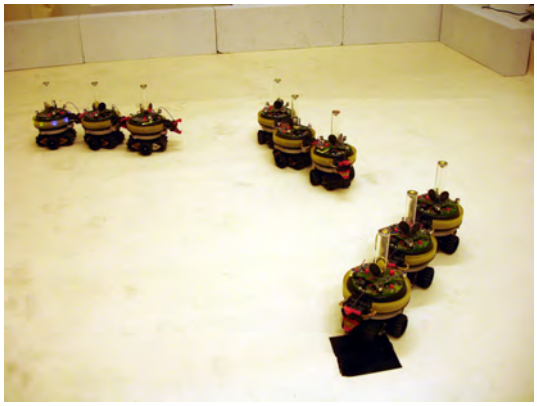
9.3.2 Results: Multiple Morphologies

In this section we demonstrate the generation of multiple morphologies by generating multiple identical copies of two morphologies — the *lines-of-three* morphology (Figure 9.6) and the *mini-squares* morphology (Figure 9.7). The experimental set-up is the same as in Section 9.2.4, including the use of obstacle-based seeding.

In the *lines-of-three* morphology, a linear morphology consisting of three s-bots is initially constructed. The third and final robot in the morphology opens a temporary connection slot. When the fourth robot connects to the morphology, it receives instruction sequence identifier '3' that causes it to disconnect, reverse for 5 s and then start a new *lines-of-three* morphology.

The *mini-squares* morphology utilizes the same principle. Each mini-square consists of four robots. The fifth robot to connect immediately disconnects, retreats and starts the morphology growth process again. For both the line morphology and the mini-square morphology, the formation process continues until no more robots are available.

We chose to generate several copies of simple morphologies in order to demonstrate the generation of multiple morphologies. In principle, there is nothing preventing the generation of multiple complex and/or different morphologies. If we had had sufficient robots available, we could have made multiple *horseshoes*, *shovels*, and *squares* in the same way.



```
RandomWalkUntil( hole-detected or
                 connection-slot-detected )
if hole-detected then
    InviteConnection( back )
    SendInstrSeqId( 1 )
    StopExecution()
end
if connection-slot-detected then
    FindSlotThenConnect()
    ReceiveInstrSeqId()
    if received-seq-id = 0 then
        StopExecution()
    end
    if received-seq-id = 1 then
        InviteConnection( back )
        SendInstrSeqId( 2 )
    end
    if received-seq-id = 2 then
        InviteConnection( right )
        SendInstrSeqId( 3 )
    end
    if received-seq-id = 3 then
        Disconnect()
        Retreat(5 s)
        InviteConnection( back )
        SendInstrSeqId( 1 )
    end
end
```

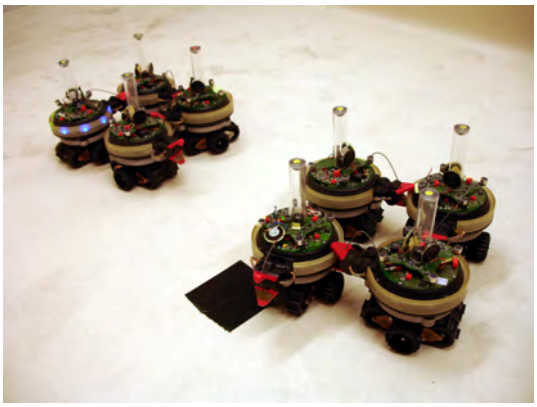
Figure 9.6: *Lines-of-three* script and result.

9.3.3 Reconfiguration

Reconfiguration in a self-assembling robotic system imposes a challenge of coordination. In the simple examples presented in this section, coordination consists of ensuring that each constituent robot waits until one morphology is complete before starting to assemble the next morphology. This type of coordination would be essential in most practical task-execution scenarios. Imagine a group of connected robots crossing a hole. If the first robots to cross the hole detect a new obstacle which triggers self-reconfiguration and these robots start trying to reconfigure while some robots are still suspended over the hole, the consequences could be disastrous.

In the hole crossing example, if an individual perceives an external stimulus that could trigger reconfiguration, it needs to be sure that the rest of the morphology is also ready to reconfigure before commencing its reconfiguration instruction sequence. Consider an individual robot that is part of a morphology and receives some 'trigger' to reconfigure. Although this individual robot is ready to reconfigure, before commencing its reconfiguration instruction sequence it needs to be sure that the rest of the morphology is also ready to reconfigure. The particular criteria for reconfiguration will depend on the morphology and the task. In the hole-crossing example, we would expect each different branch of the morphology to have successfully crossed the hole before any reconfiguration takes place.

On our hardware platform, each robot has a single gripper. This means that any morphology formed, whatever its spatial configuration (shape), has a tree-like connection topology, where a parent node can have many child nodes, but a child node can only have a single parent node. To enable coordination in a connected morphology, we allow for two types of communication. Status signals can be passed up the tree, and command signals can be passed down the tree.



```
RandomWalkUntil( hole-detected or
                 connection-slot-detected )
if hole-detected then
    InviteConnection( left )
    SendInstrSeqId( 1 )
    StopExecution()
end
if connection-slot-detected then
    FindSlotThenConnect()
    ReceiveInstrSeqId()
    if received-seq-id = 0 then
        StopExecution()
    end
    if received-seq-id = 1 then
        InviteConnection( right )
        SendInstrSeqId( 2 )
    end
    if received-seq-id = 2 then
        InviteConnection( right )
        SendInstrSeqId( 3 )
    end
    if received-seq-id = 3 then
        InviteConnection( back )
        SendInstrSeqId( 4 )
    end
    if received-seq-id = 4 then
        Disconnect()
        Retreat( 5 s )
        InviteConnection( left )
        SendInstrSeqId( 1 )
    end
end
```

Figure 9.7: *Mini-squares* script and result.

9.3.3.1 Instructions:

`SendSignal( [connection slot] ),`
`PauseUntilSignal( signal-from-[connection slot] )`

The implementation of sending and receiving signals relies on a mechanism very similar to that of the handshake rule presented in Chapter 8. The instructions `SendSignal` and `PauseUntilSignal` respectively send and wait for signals between connected robots in a morphology. Possible parameters for the `SendSignal` instruction are: `front-left`, `left`, `back-left`, `back`, `back-right`, `right`, `front-right`, `front`. Possible parameters for the `PauseUntilSignal` instruction are: `signal-from-front-left`, `signal-from-left`, `signal-from-back-left`, `signal-from-back`, `signal-from-back-right`, `signal-from-right`, `signal-from-front-right`, `signal-from-front`, `signal-from-any`.

For both instructions, the given parameter specifies the particular connected robot to which a signal should be sent or from which a signal should be received. The `front`/ `signal-from-front` parameters imply sending a signal to or receiving a signal from the parent s-bot (i.e., to/from the s-bot that has been gripped by the s-bot executing the `SendSignal` or `PauseUntilSignal` instruction). All other parameters imply sending a signal to or receiving a signal from a specified child s-bot (i.e., to/from an s-bot that has gripped the s-bot executing the instruction). The `signal-from-any` parameter for the `PauseUntilSignal` instruction allows execution to continue once a signal has been received from any s-bot to which the instruction executing s-bot is attached. This parameter is not applicable for the `SendSignal` command, as a signal must always be sent to a single specified s-bot.

9.3.3.2 Example Script: Reconfiguration

An example of reconfiguration using these two types of communication is shown in Figure 9.8. In this simple example, a three s-bot arrow morphology (one root node, two child nodes) reconfigures into a line morphology. The script to generate this behaviour is shown in Script 4 (the script includes instructions for the self-assembly of the initial arrow morphology which is not shown in the figure). The behavioural control is homogeneous for the three robots — the root node is the first robot to encounter a hole, the other two robots see an open connection slot, attach, and become the child nodes.

In Figure 9.8, a small grey circle in the center of an s-bot indicates that the s-bot is waiting for a signal. In Figure 9.8 step 1, the arrow morphology has already formed and the seed is waiting for both children to signal that they are ready to reconfigure. In steps 2-4, the child nodes independently wait for a given timeout before they are 'ready'. Once the child nodes are ready to reconfigure, they both independently signal their readiness to the root node.[2] Once they have signalled their readiness to the root node (status signals passing up the tree), the child nodes wait for a signal from the root node (command signals passing down the tree) before proceeding with any further reconfiguration steps. In step 5, the root node has received signals from both of its children, and thus knows that the whole morphology is ready to reconfigure. In steps 6-8, the root node signals to both of the child nodes in turn that they can proceed with the reconfiguration. In steps 9-10, having received the signal, each child node carries out its subsequent reconfiguration instructions, which results in the formation of a line.

[2] In this example, the child node signals do not overlap temporally. The algorithm would be unaffected, however, if both child nodes signalled their readiness at the same time—the root node would acknowledge receipt of each of the two signals in turn.

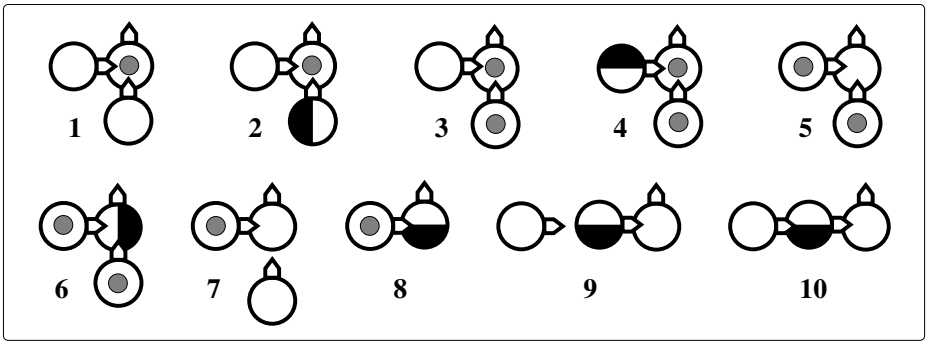


Figure 9.8: Coordinated reconfiguration example. A small grey circle in the center of an s-bot indicates that the s-bot is waiting for a signal. This behaviour is produced when each of the s-bots execute Script 4.

In the more general case, morphologies can be of arbitrary depth and have arbitrary numbers of branches. Status signals are passed up the tree from child nodes to parent nodes. Status information continues to ascend the tree in this manner until one node takes responsibility for collating the status information and issuing instructions to nodes beneath it in the tree. Starting from this collating node, command signals to start the reconfiguration process are then passed down the tree from parent nodes to child nodes.

In the experiments we perform in this study, the node that takes responsibility for collating status information is always the root node (that is, the node that has no parents). However, in other cases, a purely local reconfiguration might be more efficient — this could occur if a node further down the tree took responsibility for collating status information and issuing reconfiguration command signals. Imagine, for example, a self-assembled entity in which a single constituent robot develops a fault. The local structure could reconfigure to eject the faulty robot, or otherwise compensate for the fault. The rest of the assembled robots need not be involved in or even be aware of the local reconfiguration. Note that local configuration still requires coordination — it is just that the coordination is restricted to the subset of assembled robots that are reconfiguring.

9.3.4 Results: Reconfiguration

We performed an experiment with real robots in which six s-bots form the sequence of morphologies: star-line-square-star. We implemented a simple coordination strategy to ensure that each individual morphology is complete before reconfiguration into the next morphology begins: each sub-branch of the morphology reports local completion (signals passing up the tree). The root node collates the information, and once it is sure that all sub-branches have completed, it sends the command to reconfigure propagating through each branch (signals passing down the tree). As individual nodes receive the reconfiguration command signal, they execute their subsequent reconfiguration control logic that results in the formation of the next morphology.

Photographic snapshots of this experiment with the real robots are shown in Figure 9.10. For videos of the experiments described in this section and other explanatory material, including full swarmorph-script reconfiguration algorithms, see [106].

We conducted several more complex experiments in simulation. An example is shown in Figure 9.11. Here, by executing the relevant script, the s-bots first assemble into a 9-robot square formation and then reconfigure into three 3-robot arrow morphologies.

```
RandomWalkUntil( hole-detected or connection-slot-detected )
if hole-detected then
    // I am the root node, because I detected the hole before I saw a connection slot
    InviteConnection( back )          // Attract a child robot and...
    SendInstrSeqId( 1 )               // instruct it to follow rule 1
    InviteConnection( left )          // Attract another child robot and...
    SendInstrSeqId( 2 )               // instruct it to follow rule 2
    PauseUntilSignal( signal-from-any )          // One child is ready
    PauseUntilSignal( signal-from-any )          // Other child is ready
    SendSignal( back )                // Tell one child to start reconfiguration
    SendSignal( left )                // Tell other child to start reconfiguration
    StopExecution()
end
if connection-slot-detected then
    // I am a child node, because I saw a connection slot before I detected the hole
    FindSlotThenConnect()
    ReceiveInstrSeqId()
    if received-seq-id = 1 then
        Timeout()                     // Ready after timeout
        SendSignal( front )           // Inform parent I am ready
        PauseUntilSignal( signal-from-front )  // Wait for reconfigure command from parent
        Disconnect()
        FindSlotThenConnect()
    end
    if received-seq-id = 2 then
        Timeout()                     // Ready after timeout
        SendSignal( front )           // Inform parent I am ready
        PauseUntilSignal( signal-from-front )  // Wait for reconfigure command from parent
        InviteConnection( back )
    end
end
StopExecution()
```

Script 4: Reconfiguration script to generate the reconfiguration sequence in Figure 9.8. The script is independently executed on each of the robots.

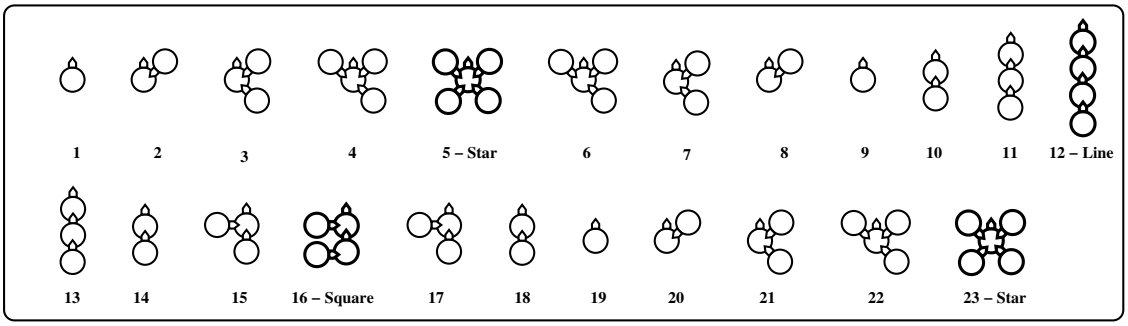
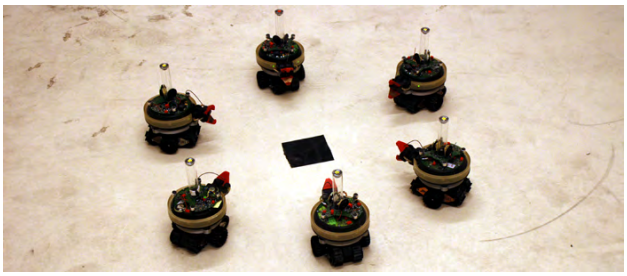


Figure 9.9: Reconfiguration steps from a star to a line to a square and back to star.

a) The start configuration:



b) The star morphology:

c) The line morphology:

d) The line morphology reconfiguring into the square:

e) The square morphology:

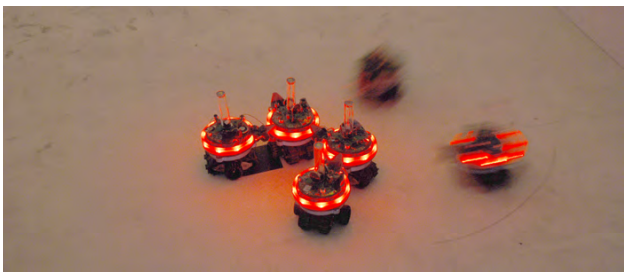


Figure 9.10: Photos of different stages in a self-reconfiguration experiment with six real s-bots.

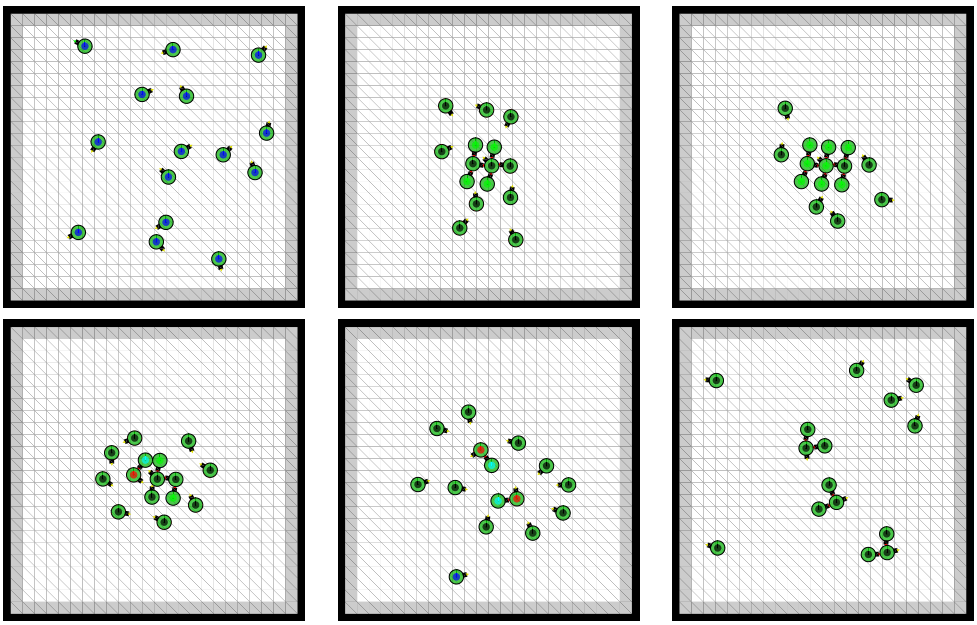


Figure 9.11: Snapshots of different stages in a self-reconfiguration experiment with fifteen simulated s-bots (reconfiguration from single 9 s-bot square morphology to three 3 s-bot arrow morphologies).

During the reconfiguration, two connected pairs of s-bots (four s-bots in total) remain connected. Using a swarmorph-script instruction for coordinated motion, these pairs travel away from the site of the original morphology in opposite directions to give themselves the space required to form subsequent morphologies. These pairs form the basis for two of the three new arrow morphologies. The s-bots that seeded the square morphology does not move, and becomes the seed for the third new arrow morphology. All other s-bots detach and try to join growing morphologies by attaching to displayed connection slots.

9.4 Discussion and Conclusions

The work presented in this chapter contributed to self-assembly research by demonstrating how arbitrary morphologies of specific sizes and shapes can be formed on a self-propelled self-assembling robotic system. We proposed a morphology generation language — swarmorph-script — which allows radically faster morphology prototyping, and enforces strict modularity of behavioural control. We also demonstrated coordinated reconfiguration through repeated self-assembly and disassembly.

We conducted a series of real-world and simulation based experiments, thereby demonstrating the versatility of our script based approach. We showed how multiple morphologies could be generated in parallel (separate morphologies formed by different groups of robots) or in series (the same group of robots reconfiguring into different morphologies by assembling and disassembling repeatedly). Both types of morphology generation are essential steps towards realising the vision outlined in the introduction of a system that can respond to environmental circumstances by creating teams of appropriately shaped composite robotic entities.

However, the system proposed so far still has the following limitations:

1. the system lacks the ability to form different morphologies in response to environmental contingencies;

2. the system lacks the ability to carry out tasks when assembled into a larger robotic entity;

3. robots can only participate in morphology growth when they already have been pre-programmed with the necessary instruction sets required to build a particular morphology. This limits the adaptive capabilities of the system, as robots cannot formulate appropriate morphologies on the fly and then request other robots to join a never before seen morphology.

Chapter 10 addresses points 1 and 2 above. The final point forms part of our ongoing research, and is discussed in Chapter 11.

Chapter 10

Morphologically Responsive Self-Assembling Robots

The research in this chapter contributed to the self-assembly research field by providing the first integrated demonstration of self-assembling robots responding to different environmental contingencies by forming appropriate morphologies. To the best of our knowledge this is the first research study in which self-assembling robots used morphogenesis to solve tasks. We present additional swarmorph-script instructions that allow us to integrate the decision making mechanism from Chapter 4 and enable the coordination required for task execution. The task consists of up to three subtasks that are encountered in an a priori unknown order. We show how commands can be combined into a script that allows the system to respond to the different tasks with appropriate morphologies. We conduct experiments on real-robots and in simulation to validate the approach. We discuss how this new use of self-assembling robots leads naturally to parallelism and corresponding new measures of scalability and interference for self-assembling systems, and apply these measures in the analysis of our experiments.

This chapter is organised as follows. In Section 10.1, we describe our methodology. In Section 10.2, we describe the task that we tackle in this chapter. In Section 10.3, we present the additional swarmorph-script instructions that integrate work from previous chapters. In Section 10.4, we present a series of experiments conducted in simulation with all three different tasks, each requiring a different morphology. Finally, in Section 10.5 we present some real-world experimentation conducted in environments with two different tasks, before concluding the chapter in Section 10.6.

10.1 Methodology

In our scenario, a series of subtasks must be completed. Each subtask is solvable by a dedicated self-assembled morphology, which is incapable of solving the other subtasks. The robots start at one end of the arena and perform phototaxis towards a light source at the other end of the arena. As they proceed, environmental cues[1] indicate the presence of particular subtasks to be solved. When they encounter a subtask, the robots must assemble into the appropriate morphology for the subtask at hand. Once that subtask is complete, the robots disassemble and continue phototaxis. They are thus ready to assemble into another morphology as soon as they encounter another subtask. The nature

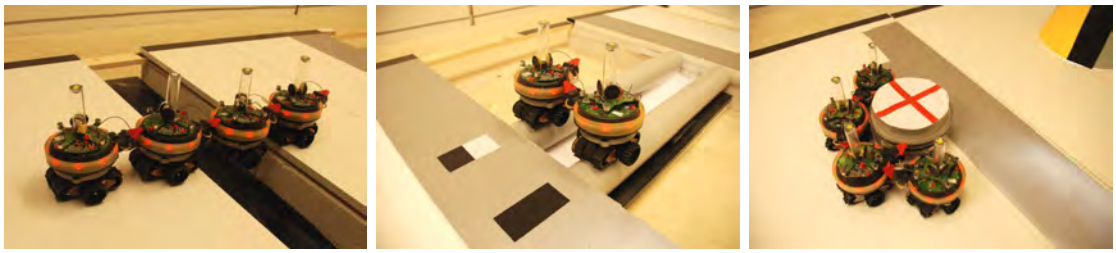


Figure 10.1: The three tasks and the appropriate morphology for each tasks. Left: The gap crossing task (line morphology). Middle: The bridge traversal task (support morphology). Right: The object pushing task (shovel morphology).

of the subtasks allows for a degree of parallel execution.

We extend the swarmorph-script language to enable the formation of particular self-assembled morphologies in response to the presence of specific tasks in the environment. Self-assembled morphologies are formed on demand in response to environmental cues. We demonstrate the feasibility of our enhanced system in a dedicated simulation environment. Using our scenario, we explore the behaviour of our system under different configurations. We investigate the negative influence of interference by increasing the number of robots while keeping the size of the arena and the number of tasks constant. We investigate how the system scales by concurrently increasing the size of the arena, the number of robots and the number of tasks.

10.2 Tasks and Morphologies

We have chosen three tasks: gap crossing, bridge traversal, and object pushing. None of these tasks can be solved by a single robot operating alone. Instead, the robots have to self-assemble and cooperate in order to accomplish each of the three tasks. Based on trial and error experimentation with real robots, we have designed the three tasks so that each task requires the robots to self-assemble into a dedicated morphology. Each morphology can solve one task and one task only, that is, the dedicated morphology that succeeds in solving one of the tasks will fail to solve if applied to either of the other two tasks. The tasks and their associated morphologies are shown in Figure 10.1 and described in detail below.

10.2.1 The Gap Crossing Task

In this task, the robots must cross a 22 cm rectangular hole that runs the width of the arena. An s-bot can detect the gap based on readings from its infrared ground sensors. Of the s-bot's four ground sensors, one points slightly forwards and one points slightly backwards. This allows an s-bot to detect a gap before falling into it. A gap of 22 cm was chosen because it is reliably passable by four real s-bots connected in linear morphology, while a three s-bot linear morphology will fail unless it is perfectly aligned (any smaller morphology always fails).

[1]The sensory equipment available on the s-bot platform is not sufficiently sophisticated to allow for a truly adaptive morphological response mechanism. Instead, as we discuss in Section 10.2, we place cues in the environment that are detectable by the s-bots. The cues uniquely identify the different tasks, and trigger the formation of the appropriate morphology.

Although the grippers on the s-bots are powerful enough for one s-bot to support two other s-bots suspended over the gap, it puts significant strain on the hardware. When we perform real-world replications of our experiment, we therefore replace the gap by an equal-sized stripe of black arena surface. To the s-bot ground sensors, the black surface is almost indistinguishable from a real hole. A video of the s-bots crossing a 22 cm real gap can be seen in [101].

10.2.2 The Bridge Traversal Task

In this task, the robots must use a bridge to cross a 50 cm rectangular hole that runs the width of the arena. The bridge is made of two pipes spaced 17.5 cm apart, each with a diameter of 8 cm. The curvature of the pipes is sufficient that a moving s-bot cannot balance on a single pipe. The two pipes are also sufficiently far apart that the wheels of a single s-bot cannot make contact with both pipes at the same time. Thus, a single s-bot cannot traverse a bridge alone. However, a composite robotic entity comprised of two physically connected s-bots (appropriately oriented) can traverse a bridge, since it can make contact with both pipes at the same time—each s-bot touches one of the pipes. The curvature of the pipes does not cause the constituent s-bots of such an entity to topple, as the s-bots mutually support each other, see Figure 10.1 (middle).

The on-board computer vision software does not enable the robots to estimate the width of a gap or to see the bridge. We have therefore placed a special reflective material before the bridged 50 cm gap to distinguish it from the 22 cm gap. The reflective material can be detected by an s-bot using its infrared ground sensors: readings are higher than for the normal arena floor. In order to determine the position of the bridge, we have put a distinct simple bar code in front of each pipe, see Figure 10.1 (middle). The bar code is made up of different materials that can be detected by an s-bot's ground sensors. Whenever a robot detects a bar code, it can use the bar code information to determine which pipe it is facing (left pipe or right pipe) and build the morphology to cross the bridge accordingly. We have also added reflective material on the far side of the bridge to allow the robots to detect when they have successfully crossed the bridge.

10.2.3 The Object Pushing Task

In this task, the robots have to perform cooperative transport by pushing two or more objects 30 cm towards the light source. The objects have a dimension and weight that prevents a single s-bot from pushing them. In fact, a shovel shape formed by four robots is necessary to reliably shift an object, see Figure 10.1 (right). We use objects with a diameter of 20 cm positioned in front of a 30 cm expanse of reflective material. The robots are programmed so that when they have reached the end of the reflective material, they disassemble and move back across the reflective material to search for more objects. The objects are wrapped in the same reflective material. An object that should be shifted can thus be detected by an s-bot based on proximity sensor readings—because of the reflective material, the readings for the object are higher than those for either other s-bots or for walls.

10.3 Behavioural Control

In this section, we describe the script that is used to solve our three task scenario.

10.3.1 Additional Swarmorph-script Commands

10.3.1.1 Navigation Instructions:

`IndividualPhototaxisUntil( [condition] ),`
`ConnectedPhototaxisUntil( [condition] )`

We add the instructions `IndividualPhototaxisUntil` and `ConnectedPhototaxisUntil` to the swarmorph-script language. For a free robot, the instruction `IndividualPhototaxisUntil` causes it to perform greedy phototaxis towards the target light source. This behaviour is equivalent to the 'Solo_Phototaxis' behaviour presented in Section 4.2. For a robot that is part of a morphology, the `ConnectedPhototaxisUntil( [condition] )` instruction causes it to perform collective phototaxis until the given condition is met. The possible conditions implemented to date are *signal-from-[connection slot]*, *gap-crossed* and *bridge-crossed* . The collective phototaxis behaviour is equivalent to the 'Connected_Phototaxis' behaviour presented in Chapter 4 (i.e., each s-bot coordinates the rotation of its turret with the movement of its tracks).

10.3.1.2 Control Flow Instructions:

`Label: label-string, Jump( label-string )`

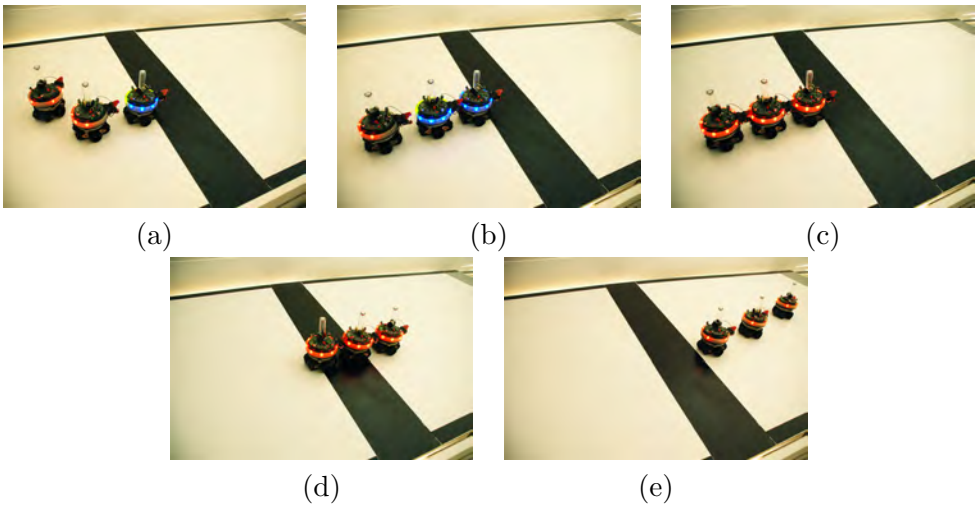
Together, the `Label` and `Jump` instruction provide a simple control flow mechanism that allows execution in a script to jump from one place to another. The `Label` instruction identifies a particular place in the script using a unique string identifier — the label-string. The `Jump` command can be used to transfer control to a location in the script identified by the particular `Label` instruction corresponding to the given label-string.

10.3.2 Example script: Respond to gap with a line morphology

Script 5 is an example of a complete script that uses task based morphogenesis. When executed on three or more s-bots, the s-bots perform phototaxis until a gap is detected. At this point, three s-bots will self-assemble into a linear morphology and attempt to cross the gap. Once the gap has been crossed, the s-bots disassemble and continue individually.

An s-bot executing Script 5 will first perform phototaxis until either a gap is detected or until another s-bot displaying a connection slot is seen. If an s-bot detects a gap using its infrared ground sensors, it becomes the seed of a new morphology: it first opens a new connection slot to its rear in order to invite a free s-bot to physically connect (see Figure 10.2a). When an s-bot has connected, the seed has to wait for a notification from the newly connected s-bot indicating that the morphology is complete before starting to cross the gap. When an s-bot connects to the seed, it receives sequence-id '0'. This tells the newly connected s-bot that it is the middle s-bot in the linear morphology and that it has to receive a connection from another s-bot in order to complete the morphology. The middle s-bot therefore opens a connection slot to its rear (see Figure 10.2b). When a free s-bot connects to the middle s-bot, the middle s-bot sends it sequence-id '1' to tell the newly connected s-bot that it is the last s-bot in the morphology. The middle s-bot then notifies the seed that the morphology is complete and starts to move across the gap (see Figure 10.2c-d). The s-bot that connected to the morphology last starts to move across the gap immediately after the middle s-bot has sent it sequence-id '1'.

The s-bot that connected to the morphology last is also the last s-bot to cross the gap. When it has crossed the gap (when its infrared ground sensors register solid group again), it sends a notification to the middle s-bot and disconnects from the morphology by opening



(a) (b) (c)

(d) (e)

Figure 10.2: An illustration of three s-bots forming a line to cross a gap. Each of the s-bots is independently executing the behavioural control logic shown in Script 5. For further explanation, see text.

its gripper. When the middle s-bot receives the notification, it forwards the notification to the seed and disconnects. When the seed receives the notification, it knows that the swarmbot has made it across the gap and that the two other s-bots have disconnected (see Figure 10.2e). Having crossed the gap, all of the s-bots restart their script and continue individually until a new gap is encountered.

```
Label: 'start'
IndependentPhototaxisUntil( gap detected OR connection-slot-detected )
if gap detected then
    // Instructions for seed s-bot:
    InviteConnection( back )
    SendInstrSeqId( 0 )
    PauseUntilSignal( signal-from-back )
    ConnectedPhototaxisUntil( signal-from-back )
    Jump( 'start' )
end
if connection-slot-detected then
    FindSlotThenConnect()
    ReceiveInstrSeqId()
    if received-seq-id = 0 then
        // Instructions for middle s-bot:
        InviteConnection( back )
        SendInstrSeqId( 1 )
        SendSignal( front )
        ConnectedPhototaxisUntil( signal-from-back )
        SendSignal( front )
        Disconnect()
        Jump( 'start' )
    end
    if received-seq-id = 1 then
        // Instructions for last s-bot:
        ConnectedPhototaxisUntil( gap-crossed )
        SendSignal( front )
        Disconnect()
        Jump( 'start' )
    end
end
```

Script 5: Formation of a 3 s-bot line morphology in response to a gap obstacle.

10.3.3 Behavioural Control

```
Label: 'PhototaxisAndLookForTasks'
IndependentPhototaxisUntil( gap detected or bridge-detected
                            or object-detected or connection-slot-detected )
if right-bridge-cue-detected then
    Retreat()                          // Retreat off the reflective material
    InviteConnection( left )           // Invite new connection from the left
    SendInstrSeqId(8)                  // Send instructions to the connected s-bot
    ConnectedPhototaxisUntil( bridge-crossed )        // Cross bridge
    Jump( 'PhototaxisAndLookForTasks' )               // Restart script
end
else if left-bridge-cue-detected then
    Retreat()                          // Retreat off the reflective material
    InviteConnection( right )          // Invite new connection from the right
    // Rest of this conditional code is identical to right-bridge conditional code above
    ...
end
else if gap detected then
    // Start a line morphology
    ...
end
else if object-detected then
    // Start a shovel morphology
    ...
end
else if connection-slot-detected then
    FindSlotThenConnect()
    ReceiveInstrSeqId()
    if received-seq-id = 1 then
        // Part of a bridge crossing morphology
        ConnectedPhototaxisUntil( bridge-crossed )
        Disconnect()
        Jump( 'PhototaxisAndLookForTasks' )           // Restart script
    end
    // Logic for the other morphologies
    ...
end
```

Script 6: Solve three subtasks in an unknown order.

A section of the script used to solve the three task morphogenesis problem is presented in Script 6. We show the global structure of the script, and focus on the part of the script that builds the two-s-bot morphology necessary to cross the bridge. Due to space considerations we do not show the full script. The full script can be found on the web at [29].

All s-bots execute this script. Initially, the s-bots perform phototaxis individually. When one of the s-bots encounters an obstacle, the s-bot illuminates its LEDs in order to invite another s-bot to connect (it opens a connection slot). When an s-bot performing phototaxis sees that another s-bot is inviting a connection, it ceases to perform phototaxis and instead tries to physically connect to the inviting s-bot. When a successful connection has been formed, communication is initiated.

Figure 10.3 shows an example of an arena used in simulation, the real arena, and an example run in which the script presented above is executed on four s-bots. A light source

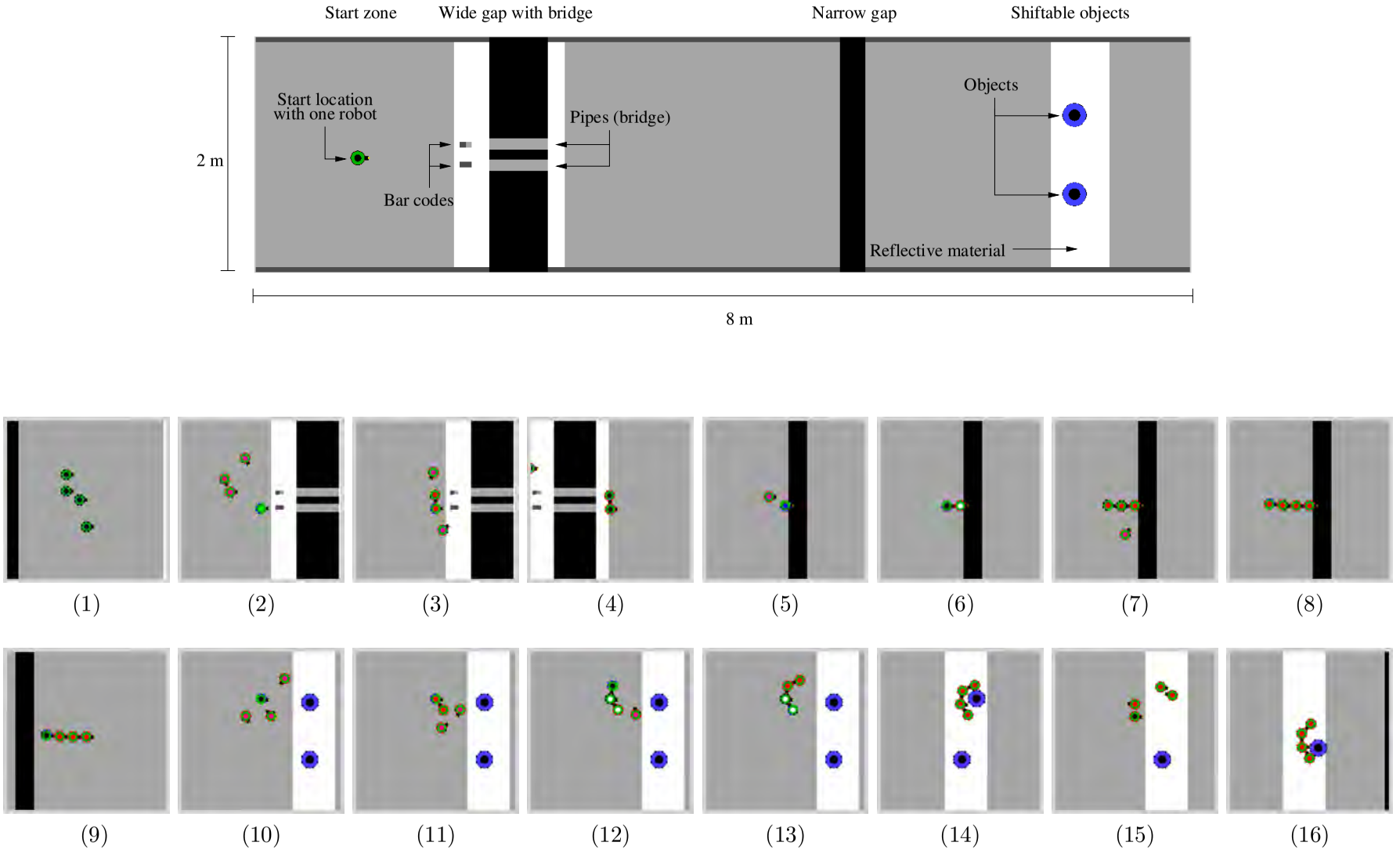


Figure 10.3: Top: A simulated arena (8 m x 2 m). Bottom: Four s-bots completing the scenario by first crossing the bridge, then the narrow gap, and finally shifting the two objects. In simulation, the light source is placed on the far right of the arena (not shown). For debugging purposes, the centres of the s-bots visually indicate the current controller state of the s-bots.

is placed on the far right of the arena (not shown), grey indicates normal arena floor, black indicates a hole, while white indicates reflective material. In Figure 10.3(1), the four s-bots start by performing phototaxis. One of the s-bots detects a bar code indicating the presence of a bridge; the s-bot retreats off the reflective material and opens a connection slot (Figure 10.3(2)). The other s-bots detect the invitation to connect and one of them manages to attach (Figure 10.3(3)). The s-bot that detected the bar code communicates with the newly attached s-bot and instructs it to start crossing the bridge. When the two s-bots have crossed the bridge (Figure 10.3(4)), they disassemble and continue performing phototaxis. One of the two s-bots to first cross the bridge then detects the narrow gap and initiates the formation of a line morphology. The other two s-bots have also crossed the bridge and contribute to the line morphology (Figure 10.3(5-8)). After crossing the gap (Figure 10.3(9)), the s-bots disassemble and continue moving towards the light source. One of the s-bots detects an object (Figure 10.3(10)) and starts the formation of a shovel morphology (Figure 10.3(11-13)). When assembled into the shovel morphology, the s-bots then push the object across the reflective material (Figure 10.3(14)), disassemble and retreat back across the reflective material (Figure 10.3(15)). The other object is encountered, a morphology is assembled, and the other object is pushed (Figure 10.3(16)).

10.4 Results: Simulation Based Experiments

In this section, we describe a series of experiments that we carried out in different arenas. We first test our script in a basic task execution experiment, where four robots carry out the three subtasks in sequence. We then test the negative influence of interference in our system, by increasing the number of robots while keeping the arena configuration constant. Finally, we test the scalability of our system by creating a series of progressively larger arenas that allow the tasks to be carried out in parallel, and conduct experiments with correspondingly larger numbers of robots.

10.4.1 Basic Task Execution

We conducted 100 experiments with 4 s-bots in a 8 m x 2 m arena containing a bridged gap, a narrow gap, and two pushable objects. We used 4 s-bots, as this is the minimum number of robots able to complete the scenario — four s-bots are needed both to cross the narrow gap (line morphology) and to push the object (shovel morphology). In each experiment, we recorded the time it took the four s-bots to navigate through the arena and to push both of the two objects 30 cm.

At the start of each experiment, the s-bots were placed in the starting zone and oriented to face the light source. We let each experiment run for 6,000 simulated seconds (= 100 minutes). The results are summarised in Table 10.1. In ninety-four of the experiments, the four s-bots succeeded in navigating the arena and pushing both of the objects the required distance.

We witnessed two types of failure that prevented one or both of the objects from being pushed in six of the experiments. Firstly, there are sometimes 'robot casualties' during task execution. We consider a robot to be a casualty if it falls into one of the gaps. When one or more robots fall into a gap before both objects have been pushed the requisite distance, there are then insufficient remaining s-bots to complete the scenario. Secondly, if the s-bots form a slightly misaligned shovel morphology, the object can slide off the side of the shovel before it has been shifted 30 cm. As a result, the object remains in the centre of the reflective band and can no longer be detected by the s-bots.

Table 10.1: Results summary of experiment with 4 s-bots in an 8 m x 2 m arena and two objects to push.

0 objects pushed	1 experiments
1 object was pushed	5 experiments
2 objects were pushed	94 experiments
Average time, 1st object	1,150 s (st.dev 310 s)
Average time, 2nd object	1,803 s (st.dev 332 s)

In one experiment, neither of the two objects were successfully pushed. This occurred due to a misaligned shovel morphology in both cases. In another five experiments, only one of the two objects was pushed (see Table 10.1). One of these experiments failed due to robot casualties, and four of these experiments failed due to misaligned morphology growth.

10.4.2 Negative Influence of Interference

Both types of failure that we saw in the previous section are caused by interference (for more details on interference and its potential role in behavioural controller design, see [60]). Interference occurs when a high local density of robots results in collisions (although the robots perform obstacle avoidance, this mechanism is overwhelmed when the density is sufficiently high). Collisions lead to robot casualties when one of the colliding robots is pushed into a gap. Collisions lead to misalignment when a robot that is inviting a connection is displaced or rotated by a collision with another s-bot.

To determine the influence of interference on task completion performance, we ran an additional set of experiments with a varying number of s-bots in the same 8 m x 2 m arena that we used in Section 10.4.1 (see Figure 10.3(top left)). In each experiment, the s-bots were initially placed in the starting zone and oriented to face the light source.

The results are summarised in Figure 10.4. For each experimental setup with a given number of s-bots, we performed 100 replications. In each replication, we varied the initial placements and initial seed for the random number generator. The results for each set of experiments are summarised by two bars. The wide bars indicate the average task completion time and standard deviation observed in 100 replications of the experimental setup. The narrow bars denote the percentage of robot casualties.

As the results indicate, the average performance initially increases as more s-bots are added. However, at a group size of 18 s-bots, the average performance begins to decrease. Furthermore, the percentage of robot casualties (the narrow bars in Figure 10.4) increases monotonically with the robot density. In the four s-bots experiments, we observed one robot casualty in a single experiment, yielding a robot casualty percentage of 0.25%. When 30 s-bots are present in the same arena, the robot casualty percentages is 20.90% ($\approx$ 6 s-bots/experiment on average).

10.4.3 Scalability

In order to evaluate the scalability of our approach, we ran a series of experiments with progressively larger numbers of s-bots and correspondingly larger arenas. We varied the size of the initial robot population from 100 robots to 1,000 robots in increments of 100. We performed 100 replications for each population size. Each experiment was run for

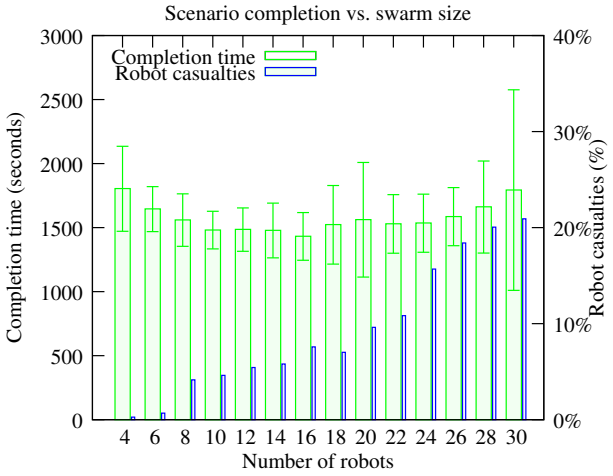


Figure 10.4: Scenario completion times and robot casualties for swarms of different sizes in an 8 m x 2 m arena with pushable objects.

1,500 simulated seconds (= 25 minutes).

For each population size, we set the width of the arena, the number of bridges and the number of objects as a function of the number of robots in the population. If n is the number of s-bots in a given experimental setup, the arena is 8 m long, $w = n/5$ meters wide, contains $b = n/10$ bridges, and $o = n/5$ pushable objects. The bridges and pushable objects are uniformly distributed along two lines running the width of the arena. Note that to obtain the arena that was used in the experiments of the previous two sections, we would need an initial population size of 10 robots ($n = 10$). An example of an arena corresponding to an initial robot population of 50 s-bots ($n = 50$) is shown in Figure 10.5.

The results are summarised in Figure 10.6. Each bar denotes the average number of objects pushed by a swarm of a fixed size over the 100 replications of the experiment. Each bar is annotated with the standard deviation for the result set.

In Figure 10.6, we have added a least squares fit line ($y = 0.117 \cdot x$). As the results show, the task execution performance scales linearly with the number of s-bots. Linear scalability should not come as a surprise: the behavioural control is completely decentralised and each s-bot acts based only on what it senses in its immediate vicinity. We therefore expect that the linear scalability trend would continue beyond swarms of 1,000 s-bots (however, given the computational resources required, we have been unable to confirm this).

10.5 Results: Real World Experiments

Our real world experiments take place in a simplified version of the arena used for simulation based experiments, containing only two obstacles — the gap obstacle and the bridge obstacle. The arena can be seen in Figure 10.7. All other aspects of the simulation based experimentation are the same. As before, the order of the obstacles in the arena is variable, and the robots do not know a priori which obstacle they will encounter first.

We conducted a set of 5 experiments with 4 robots encountering the gap obstacle[2]. Figure 10.8 shows a series of frames from a successful experiment. In all five experiments, the system successfully detected the gap and formed the appropriate morphology. In four out of five experiments, the signalling mechanisms successfully allowed the four robot line morphology to start collective phototaxis in a coordinated manner, and to disassemble

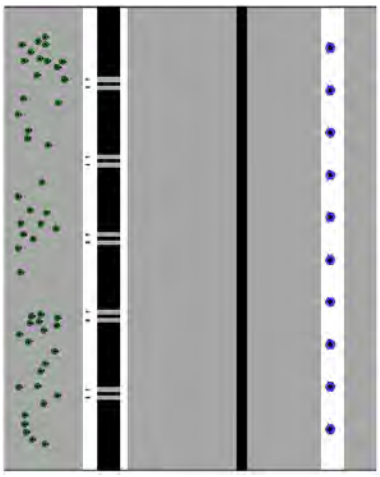


Figure 10.5: An example of an arena for scalability experiments with 50 s-bots ($n = 50$). The width w of the arena is 10m, the number of bridges b is 5, and the number of pushable objects o is 10.

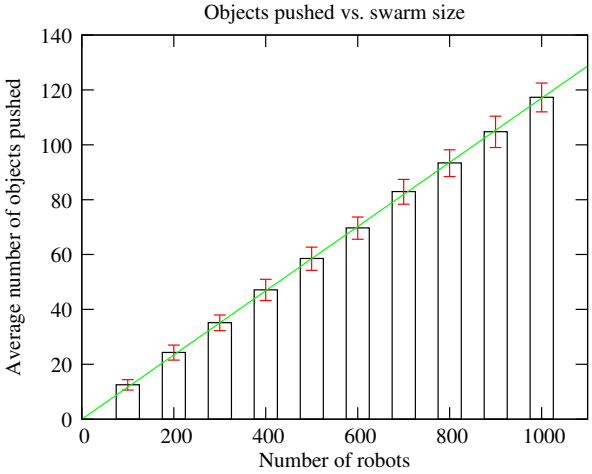


Figure 10.6: Scalability

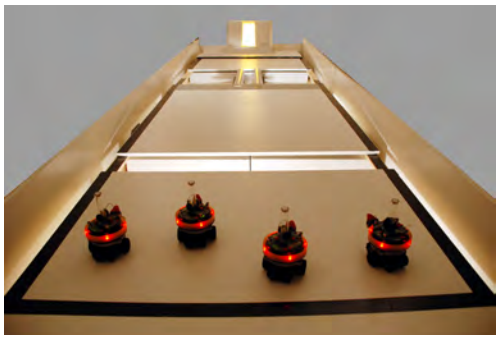


Figure 10.7: The real world arena.

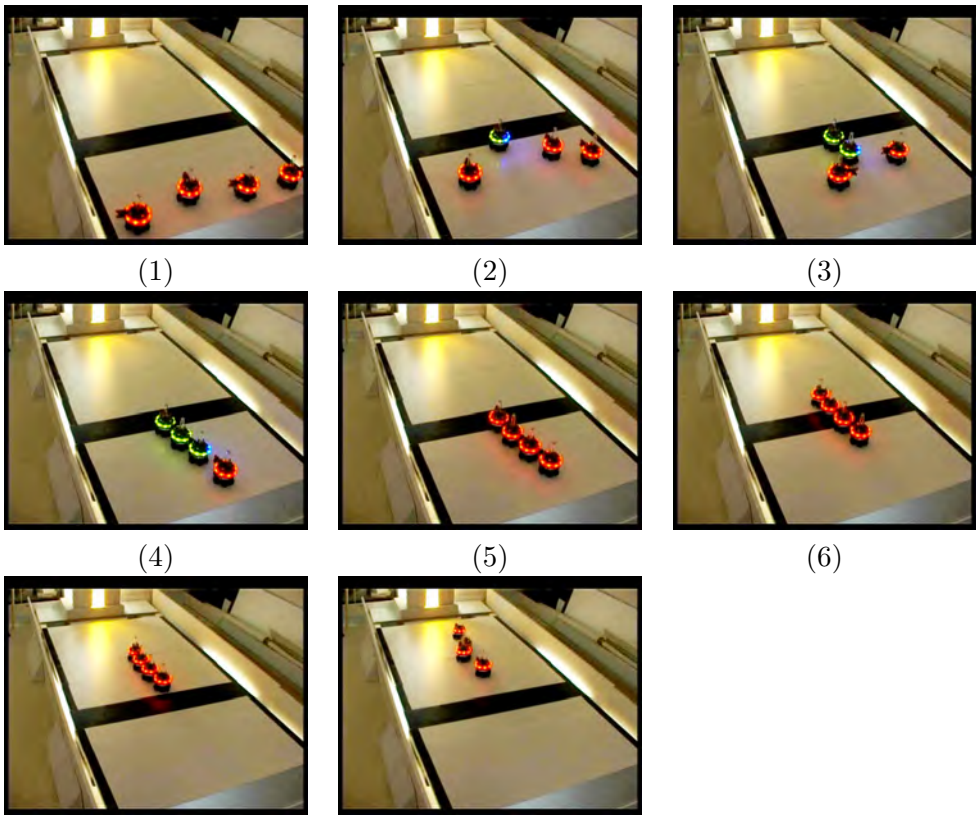


Figure 10.8: Video frames from an experiment in which four robots form a line morphology and cross a gap.

once the final robot had crossed the gap. In a single experiment, a failed notification signal between the third and fourth robots in the line morphology resulted in the first two robots in the morphology commencing collective phototaxis while the rear two robots were still engaged in (unsuccessful) signalling—the experiment was manually aborted.

We conducted 5 trials in which two robots encountered the bridge. In all 5 trials, the robots successfully detected the environmental cues and formed the correct morphology with which to cross the bridge. In three out of 5 experiments, the robots successfully crossed the bridge. In 2 out of 5 experiments, inaccuracies in the detection of the target light source caused the robots to veer sideways off the bridge, and they had to be manually placed back on the bridge. In every experiment, the robots successfully detected that they had crossed the bridge and disassembled.

Frames from an experiment with 4 robots crossing the bridge 2-by-2 with real robots are shown in Figure 10.9. The frames in the figure are taken from a proof-of-concept video in which we ran four robots over both obstacles in sequence. The figure shows two sets of two robots crossing the bridge, and parallel task execution as the first pair of robots over the bridge encounter the gap obstacle, and start forming the line morphology, while the second set of robots forms the support morphology to cross the bridge. Videos of experiments in this section can be found in the supplementary material [101].

10.6 Discussions and Conclusions

The results in this chapter fulfil the goals of this thesis as stated in the introduction. We showed that our approach allows a group of robots to solve different tasks by self-assembling into appropriate morphologies. We integrated the different research strands that we presented throughout this thesis. We conducted experiments in which a group of robots overcame obstacles faced in an a priori unknown order by self-assembling into obstacle-specific morphologies.

Additional work in this chapter included showing 1) how dedicated morphologies are grown from a single seed robot when an obstacle or an environmental clue is encountered, 2) how the local visual notifications are used to coordinate the navigation over an obstacle, and 3) how the robots disassemble once the obstacle has been overcome and continue individually until the next obstacle is encountered. In our experiments, the robots were all homogeneous and autonomous.

In this chapter, we arrived at a morphologically responsive self-assembling system. However, much more research is required before self-assembling systems are ready for practical real-world applications. In particular, we have still not achieved genuine adaptivity. In our experiments, the response to the different types of obstacles – namely the formation of dedicated morphologies – was preprogrammed. Whenever a gap is encountered, for instance, a linear morphology of four robots is always self-assembled – regardless of the width of the gap. In order for the robots to be truly adaptive, they should be able to respond to previously unseen obstacles by forming new and appropriate morphologies during task-execution. In the next chapter, we discuss ongoing and future work dedicated to achieving this kind of true morphological adaptivity with a self-assembling system.

[2] As noted previously, to prevent damage to the robots we used a black surface instead of a gap. This presents the same sensory information to the robots' ground sensors as the real gap obstacle.

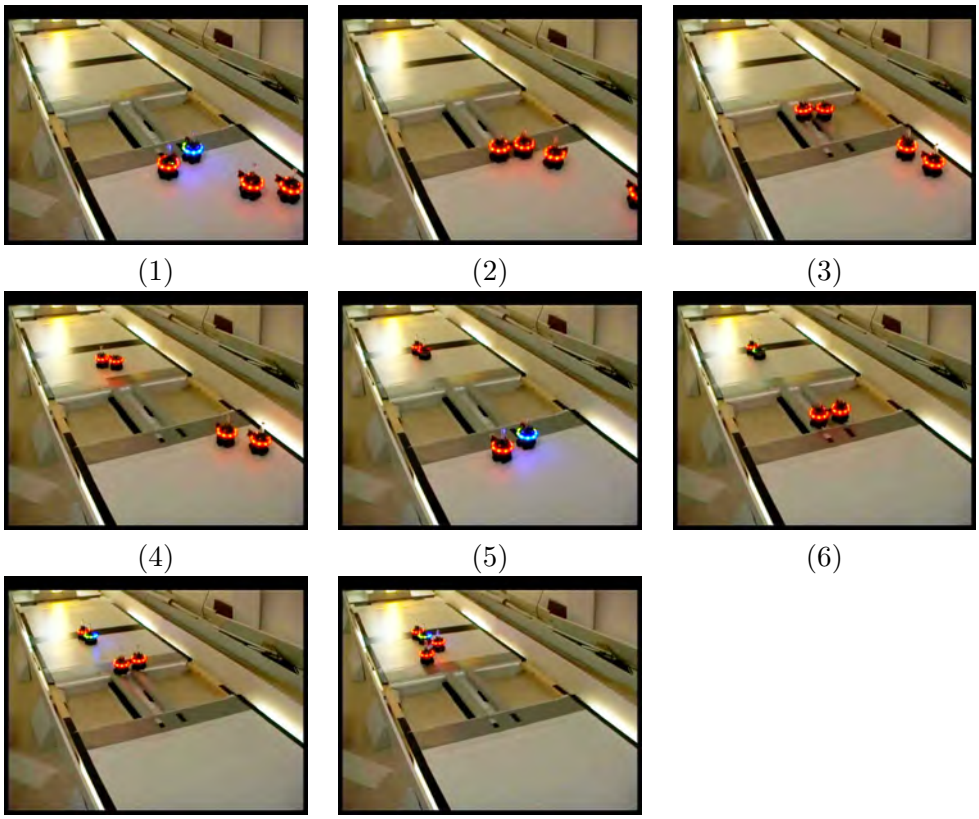


Figure 10.9: Video frames from an experiment in which four robots cross the bridge obstacle by forming two support morphologies.

Chapter 11

Discussion and Conclusions

In this chapter, we conclude the thesis. We first present some ongoing and potential future work. We then present our conclusions.

11.1 Ongoing and Future Work

The long term vision that we outlined in the introduction to this thesis described future self-assembling systems that would respond to their task and environment by allocating resources and creating appropriate robotic entities when needed. In this section, we discuss how future work might build on the work presented in this thesis and start achieving this kind of adaptivity. We consider adaptivity to be firstly the ability to detect the salient features of a task and environment and secondly the ability to respond to these features appropriately.

We first present a transmission mechanism for behavioural control logic which allows s-bots to transmit whole or partial scripts to each other. This frees swarmorph-script from the constraints of having all morphology generation logic pre-coded into a script. Using behavioural control logic transmission, robots can formulate an appropriate morphology on the fly, in response to never seen before conditions, and communicate the relevant instructions to create the morphology to other robots. We also discuss future work in the context of the swarmanoid project [39]. This ongoing project studies the interaction of heteregeneous robotic swarms. In particular, aerial robots might be used in the future to direct the morphology generation activities of ground based robots.

11.1.1 Behavioural Control Logic Transmission

Behavioural control logic transmission allows robots with no a priori knowledge of a particular morphology to nonetheless take part in morphology growth. The morphologies presented in Chapters 9 and 10 relied on the communication of a single number between connected robots — an instruction sequence identifier that mapped to specific sequences of behavioural control on the recipient robot. This approach is efficient in terms of restricting communication to a minimum, but has the disadvantage that the set of morphologies that can be generated is limited by the set of instructions given to the robots at the start of an experiment.

In some adaptive task-execution scenarios it is infeasible to store a finite set of pre-programmed morphologies to match every possible environmental contingency. Imagine, for example, a scenario in which one robot encounters a previously unseen obstacle. By

analysing the obstacle, the robot calculates the appropriate morphology to overcome the obstacle. This morphology is a function of the obstacle encountered — the morphology cannot be pre-programmed as there are too many possible obstacle shapes that may be encountered.

In scenarios in which pre-programmed morphologies are not feasible, instruction sequence IDs are insufficient as a means to communicate which morphology should be formed. The transmission of whole or partial swarmorph-script behavioural control programs solves this problem. In the example above, the robot that encounters the obstacle above can go and recruit other robots (that have no knowledge of the obstacle and therefore no knowledge of the appropriate morphology) by sending the other robots a script with instructions on how to construct the relevant morphology.

11.1.1.1 Script Communication Instructions

`SendScript(` *`[encoded instruction sequence]`* `|` `'self'` `),`
`ReceiveScript(), ExecuteReceivedScript()`

To overcome these restrictions, we introduced dedicated instructions into the swarmorph-script language to enable the transmission of whole or partial scripts. In Chapter 9, we used a LED based communication mechanism to transmit instruction sequence identifiers. The same mechanism is capable (without modification) of communicating arbitrary strings between connected robots. By creating an appropriate mapping, we use this mechanism to communicate whole or partial scripts. To send a script, each swarmorph-script instruction is compiled to a 5-bit string literal before being sent. The robot that receives a script maps these 5-bit string literals back to swarmorph-script instructions, and then executes the script thus received. The `SendScript` command takes a list of swarmorph-script instructions as a parameter. Alternatively, a special parameter, `'self'`, can be provided to the `SendScript` instruction which tells the sending robot to transmit a copy of the whole script it is currently executing.

We demonstrate the communication of morphology control logic with three different morphologies — the *communicated-line-replicator*, the *communicated-triangle-replicator* and the *communicated-step-replicator*. Our experimental setup mimics the scenario described above: we program a single 'master' robot with morphology control logic to create a morphology and to communicate the same behavioural control to other robots. All other robots are 'slave' robots that start without any a priori morphology creation logic, except the basic logic required to attach to a connection slot and execute any morphology control logic received through communication. The script that we execute on these 'slave' robots is shown in Script 7.

11.1.1.2 Results: Morphologies Generated using Behavioural Control Logic Transmission

To demonstrate the communication of both partial behavioural control programs and whole behavioural control programs we use replicating morphologies. Partial behavioural control programs to form segments of the morphology are communicated as the morphologies are developing. The whole behavioural control program (to generate the entire morphology and subsequently to repeat the replication process) is communicated when morphology replication occurs. The three *replicator* scripts (as executed on the seed robot) and the result when running them on real *s-bots* are shown in Figure 11.1, Figure 11.2

```
Disconnect();
Retreat(10 s);
InviteConnection(back-left);
SendScript(
  StopExecution();
);
InviteConnection(back-right);
SendScript(
  StopExecution();
);
InviteConnection(front-left);
SendScript(self);
StopExecution();
```

Figure 11.1: *Communicated-triangle-replicator* script and result.

and Figure 11.3, respectively.

Script 7: Script executed by the robots without any preprogrammed morphology specific instructions.

```
FindSlotThenConnect();
ReceiveScript();
ExecuteReceivedScript();
```

In the *communicated-triangle-replicator* (Figure 11.1), the seed robot first executes the `Disconnect` and `Retreat` instructions before opening a connection slot. This is true for all of the morphologies in this section, and is done to maintain homogeneity of behavioural control, as these are the first instructions that a 'slave' robot will need to execute when it receives instructions to replicate the morphology. The seed robot then opens a connection slot to its rear left and sends a `StopExecution` instruction to the connecting robot. Thus, the first connected robot takes no further actions. The seed then opens a second connection slot to its rear right and sends a `StopExecution` instruction to the robot that connects to the slot. When three robots are assembled in this way, their morphology resembles a triangle (see Figure 11.1). Once the three robots are assembled the seed opens a temporary connection slot to its front left and sends a copy of its own script (`SendScript(self)`)to the robot that connects. This connecting robot disconnects, retreats and restarts the process.

In the *communicated-step-replicator* (Figure 11.2), the seed robot first opens a connection slot to its left and sends a script to the connecting robot that instructs it to open a connection slot to its left and send a `StopExecution` instruction to the robot that connects. The seed robot then opens a connect slot to its right and sends a copy of its own script to the robot that connects.

The *communicated-line-replicator* (Figure 11.3), uses a similar principle - a script is sent to the first robot that connects which in turn opens a new connection slot and sends a `StopExecution` instruction to the robot that connects. The seed robot then opens another connect slot and sends a copy of its own script to the robot that connects, and a new line morphology is started. The result of the *communicated-line-replicator* is similar to the *lines-of-three* script in Chapter 9 (see Figure 11.3 and Figure 9.6). However, in the *lines-of-three* experiment, all of the robots started the experiment with the complete

```
Disconnect();
Retreat(10 s);
InviteConnection(right);
SendScript(
  InviteConnection(left);
  SendScript(
    StopExecution();
  );
  StopExecution();
);
InviteConnection(left);
SendScript(self);

StopExecution();
```

Figure 11.2: *Communicated-step-replicator* script and result. NB. In this experiment, the pattern on the right is slightly malformed. This was due to the seed robot's connection slot being mis-sensed (due to camera noise) by the first robot to attach.

```
Disconnect();
Retreat(10 s);
InviteConnection( back );
SendScript(
  InviteConnection( back );
  SendScript(
    StopExecution();
  );
  StopExecution();
);
InviteConnection(left);
SendScript(self);

StopExecution();
```

Figure 11.3: *Communicated-line-replicator* script and result.

Figure 11.4: A heterogeneous robot swarm uses adaptive morphology control in a disaster zone. Debris in an afflicted office needs to be cleared to facilitate subsequent access for search and rescue workers. An aerial robot locates an object that needs to be moved (the toppled chair) and instructs ground based robots to self-assemble into an appropriate morphology capable of moving the object.

set of instructions required to create a line morphology. By contrast, at the start of the *communicated-line-replicator* experiment, only a single robot started the experiment with the instructions required to generate a line morphology.

11.1.2 Morphology Control in a Heterogeneous Swarm

In this thesis, the detection of environmental features performed by our system was restricted by the limits of the s-bot's sensing apparatus. In Chapters 4 and 5, our system could detect simple environmental cues such as presence of a hill, steepness of a hill, orientation of a hill. In the experiments presented in Chapter 10, we were forced to place explicit cues in the arena to allow the robots to distinguish between the two types of obstacle.

For future systems to overcome such limitations, one possibility would be to give each agent in the system better sensing hardware. However, the simplicity of individual agents is often cited as a key strength of distributed self-assembling systems. An alternative might be to introduce heterogeneity into the system, and have some robots specialised in sensing that could then communicate relevant information to the other robots. In such a heterogeneous swarm, supervisory robots with more sophisticated sensory equipment could direct the morphogenesis activities of self-assembling robots.

This type of heteregeneous cooperation is currently being investigated by the Swarmanoid project (the successor to the swarm-bots project) [39]. In the swarmanoid project, flying robots with a wider vision can direct the wheeled robots on the ground. In particular, some studies have already addressed the possibility of communication and cooperation between aerial and ground based robots [92, 115]. Ongoing research effort is being applied to the problem of adapting these heterogeneous communication and cooperation modalities to enable supervised morphogenesis. Figure 11.4 shows an example of how such super-

vised morphogenesis might enable a heterogeneous group of robots to react adaptively to its environment in a future search and rescue environment.

11.1.3 Other Possible Research Directions

A different approach to overcome the sensory limitations of individual robots in a self-assembly system might be to leverage group sensing capabilities. In such a scenario, the sensory data of individual distributed agents would somehow be combined to build up a more sophisticated composite picture of the environment and of the agents' own spatial configuration [57].

Other factors limiting the range of responses available to a self-assembling morphologically flexible system include hardware limitations, the ingenuity of the human system designers, and the ability of the system to learn appropriate responses from trial and error. Learning in particular is an important avenue for future research in self-assembling systems. Learning, whether off-line or on-line, depends on the ability to detect success and failure. Although, in this study, we have not considered learning, some of the success/failure detection mechanism we designed, especially those at the group level, could be used as a starting point for future learning based systems. For example, in the robot rescue experiment (Chapter 6), the robots can already stochastically try different patterns until they find one that succeeds. The robots use their torque sensors to implement a distributed success / fail criterion that applies to the whole group with minimal communication. The next step might be to make this stochastic process more explicit — to find a way for the system to map configurations that have succeeded into swarmorph-script instruction sequences, and thus to remember them and be able to recreate them in the future when faced with similar environmental contingencies.

11.2 Conclusions

The principal contribution of this thesis has been to advance the state of the art in self-assembling robotic systems. For the first time, a robotic system has been demonstrated that can solve different tasks by creating specific appropriate morphologies. This is a major step forward, as existing self-assembly literature has for the most part focused purely on the autonomous creation of physical connections between robots.

To arrive at this point, our approach was to proceed step-wise, developing increasingly sophisticated response mechanisms. Initially, we just considered stochastic self-assembly, but showed how different strategies could enhance system efficiency and performance. We then moved towards a morphologically adaptive self-assembling response mechanism. Initially we researched the morphology control mechanism independently. We first showed how we could create distributed morphologies, then increased the sophistication of our approach by abstracting our instructions into a morphology control language. Finally, we integrated the morphology control mechanisms into a task execution scenario. We showed in both simulated and real-world experiments how a self-assembling robotic system could respond to different tasks by forming different appropriate morphologies.

The research we conducted to solve each of these sub-problems resulted in a number of corresponding contributions to self-assembly research. We proposed a novel decisional autonomy mechanism to allow a group of robots to make a distributed choice about when and if to self-assemble on the basis of task parameters. We performed the first explicit analysis of the costs and benefits of self-assembly for a multi-robot system. We developed a distributed morphology control system (swarmorph) for self-assembling robots, that is

potentially applicable to a range of self-assembling platforms. We showed how morphology control logic could be abstracted into a high level scripting language (swarmorph-script), resulting in rapid prototyping of behavioural logic.

Much remains to be done before self-assembling systems can be used in practical real-world applications. Of course self-assembling systems share many of the same problems that makes any robotic system hard to implement in messy real-world environments. However, self-assembling systems have particular issues of their own that future self-assembly research will need to address. Building on the work of this thesis, the creation of three-dimensional self-propelled self-assembling systems will be a requirement. And such systems will need to display a higher level of autonomy than is currently possible. As discussed earlier in this chapter, such systems will need to be genuinely adaptive, possibly learning about new environments as they are exposed to them. And the inherently distributed nature of self-assembling systems makes them potentially much harder to control externally. However autonomous a robotic system is, to be useful it must still be able to follow human commands at some level. In a highly distributed system it is not clear how such commands can be issued, or how human level representation of problems can be mapped into distributed control logic. The swarmorph-script approach we proposed already makes morphology generation logic more accessible, but it still requires a human operator to think at the level of individual robots, which is not intuitive.

Nonetheless, the promise held by self-assembling systems makes the continued research effort worthwhile. The inherent robustness of such systems, combined with their ability to physically cooperate when necessary makes them potentially ideal for hazardous environments such as those found in search and rescue operations or in space exploration missions. The relative simplicity of individual agents also makes self-assembling systems an attractive prospect as the size of electrical, mechanical and computational components continues to shrink. Distributed self-assembling systems of the future might operate at micro or nano scales, perhaps even inside the human body.

Bibliography

[1] C. Anderson, G. Theraulaz, and J.-L. Deneubourg. Self-assemblages in insect societies. *Insectes Sociaux*, 49(2):99–110, 2002.

[2] J. Bachrach, J. McLurkin, and A. Grue. Protoswarm: a language for programming multi-robot systems using the amorphous medium abstraction. In *Proceedings of the 7th international Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2008*, volume 3, pages 1175–1178, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[3] T. Balch and R. C. Arkin. Behavior-based formation control for multirobot teams. *IEEE Transactions on Robotics and Automation*, 14(6):926–939, 1998.

[4] R. Beckers, O. E. Holl, and J.-L. Deneubourg. From local actions to global tasks: Stigmergy and collective robotics. In *Proceedings of the 4th Workshop on Artificial Life*, pages 181–189. MIT Press, Cambridge, MA, 1994.

[5] J. Bishop, S. Burden, E. Klavins, R. Kreisberg, Malone W, N. Napp, and T. Nguyen. Programmable parts: A demonstration of the grammatical approach to self-organization. In *Proceedings of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2644–2651. IEEE Computer Society Press, Los Alamitos, CA, 2005.

[6] H. Bojinov, A. Casal, and T. Hogg. Emergent structures in modular self-reconfigurable robots. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, pages 1734–1741. IEEE Computer Society Press, Los Alamitos, CA, 2000.

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

[8] Jarle Breivik. Self-organization of template-replicating polymers and the spontaneous rise of genetic information. *Entropy*, 3:273–279, 2001.

[9] H. B. Brown, M. V. Weghe, C. Bererton, and P. Khosla. Millibot trains for enhanced mobility. *IEEE/ASME Transactions on Mechatronics*, 7(4):452–461, 2002.

[10] Z. Butler, K. Kotay, D. Rus, and K. Tomita. Generic decentralized control for lattice-based self-reconfigurable robots. *International Journal of Robotics Research*, 23(9):919–937, 2004.

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

[12] 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.

[13] 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.

[14] D. L. D. Caspar. Design principles in organized biological structures. In G. E. W. Wolstenholme and M. O'Connor, editors, *Principles of Biomolecular Organization*, pages 7–34. J. & A. Churchill Ltd., London, 1966.

[15] A. Castano, A. Behar, and P. M. Will. The Conro modules for reconfigurable robots. *IEEE/ASME Transactions on Mechatronics*, 7(4):403–409, 2002.

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

[17] A. Christensen, R. O'Grady, M. Birattari, and M. Dorigo. Fault detection in autonomous robots based on fault injection and learning. *Autonomous Robots*, 24(1):49–67, 2008.

[18] A. L. Christensen. Efficient neuro-evolution of hole-avoidance and phototaxis for a swarm-bot. Mémoire de DEA, technical report TR/IRIDIA/2005-14, Université Libre de Bruxelles, Bruxelles, Belgium, 2005.

[19] A. L. Christensen, R. O'Grady, M. Birattari, and M. Dorigo. Automatic synthesis of fault detection modules for mobile robots. In *Proceedings of the 2007 NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2007)*, pages 693–700. IEEE Computer Society Press, Los Alamitos, CA, 2007.

[20] A. L. Christensen, R. O'Grady, M. Birattari, and M. Dorigo. Exogenous fault detection in a collective robotic task. In *Proceedings of the 9th European Conference on Artificial Life, ECAL 2007*, volume LNAI 4648 of *Lecture Notes in Artificial Intelligence*, pages 555–564. Springer-Verlag, Berlin, Germany, 2007.

[21] A. L. Christensen, R. O'Grady, and M. Dorigo. A mechanism to self-assemble patterns with autonomous robots. In *Advances in Artificial Life, Proceedings of ECAL 2007*, volume LNAI 4648 of *Lecture Notes in Artificial Intelligence*, pages 716–725. Springer-Verlag, Berlin, Germany, 2007.

[22] A. L. Christensen, R. O'Grady, and M. Dorigo. Morphogenesis: Shaping swarms of intelligent robots. In *AAAI-07 Video Proceedings*. AAAI Press, Menlo Park, CA, 2007. Winner of the "Best Video" award.

[23] A. L. Christensen, R. O'Grady, and M. Dorigo. Morphology control in multirobot system. *IEEE Robotics and Automation Magazine*, 14(4):18–25, 2007.

[24] A. L. Christensen, R. O'Grady, and M. Dorigo. SWARMORPH-script: A language for arbitrary morphology generation in self-assembling robots. IRIDIA Supplementary page. `http://iridia.ulb.ac.be/supp/IridiaSupp2007-009`, 2007.

[25] A. L. Christensen, R. O'Grady, and M. Dorigo. SWARMORPH-script: A language for arbitrary morphology generation in self-assembling robots. *Swarm Intelligence*, 2(2-4):143–165, 2008.

[26] A. L. Christensen, R. O'Grady, and M. Dorigo. Synchronization and fault detection in autonomous robots. In Raja Chatila, J. P. Merlet, and C. Laugier, editors, *Video Proceedings of the 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4139–4140. IEEE Computer Society Press, Los Alamitos, CA, 2008.

[27] A. L. Christensen, R. O'Grady, and M. Dorigo. From fireflies to fault tolerant swarms of robots. *IEEE Transactions on Evolutionary Computation*, 13(4):754–766, 2009.

[28] A. L. Christensen, R. O'Grady, and M. Dorigo. Parallel task execution, morphology control and scalability in a swarm of self-assembling robots. In *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions, Robótica 2009*, pages 127–133, Castelo Branco, Portugal, 2009. IPCB-Instituto Politcnico de Castelo Branco.

[29] A. L. Christensen, R. O'Grady, and M. Dorigo. Parallel task execution, morphology control and scalability in a swarm of self-assembling robots. IRIDIA Supplementary page. `http://iridia.ulb.ac.be/supp/IridiaSupp2009-002/`, 2009.

[30] D. J. Christensen. Evolution of shape-changing and self-repairing control for the ATRON self-reconfigurable robot. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, pages 2539–2545, Los Alamitos, CA, 2006. IEEE Computer Society Press.

[31] D. J. Christensen. Experiments on fault-tolerant self-reconfiguration and emergent self-repair. In *In Proceedings of the IEEE Symposium on Artificial Life*, pages 355–361, Piscataway, NJ, 2007. IEEE Press.

[32] R. Damoto, A. Kawakami, and S. Hirose. Study of super-mechano colony: concept and basic experimental set-up. *Advanced Robotics*, 15(4):391–408, 2001.

[33] A. K. Das, R. Fierro, V. Kumar, J. P. Ostrowski, J. Spletzer, and C. J. Taylor. A vision-based formation control framework. *IEEE Transactions on Robotics and Automation*, 18(5):813–825, 2002.

[34] 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.

[35] J.-L. Deneubourg, S. Goss, G. Sandini, F. Ferrari, and P. Dario. Self-organizing collection and transport of objects in unpredictable environments. In *Proceedings of the Japan–U.S.A Symposium on Flexible Automation*, pages 1093–1098. ISCIE, Kyoto, Japan, 1990.

[36] 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.

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

[38] 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.

[39] M. Dorigo. The swarmanoid project. `http://www.swarmanoid.org`, 2009.

[40] M. Dorigo and E. Şahin. Guest editorial. Special issue: Swarm robotics. *Autonomous Robots*, 17(2–3):111–113, 2004.

[41] M. Dorigo, E. Tuci, V. Trianni, R. Groß, S. Nouyan, C. Ampatzis, T. H. Labella, R. O'Grady, M. Bonani, and F. Mondada. *SWARM-BOT: Design and Implementation of Colonies of Self-assembling Robots*, chapter 6, pages 103–135. IEEE Computational Intelligence Society, New York, 2006.

[42] A. Drogoul and J. Ferber. From tom thumb to the dockers: Some experiments with foraging robots. In *Proceedings of the 2nd International Conference on the Simulation of Adaptive Behavior*, pages 451–459. MIT Press, Cambridge, MA, 1992.

[43] 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 Press.

[44] T. A. Estlin, A. Gray, T. Mann, G. Rabideau, R. Castano, S. Chien, and E. 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.

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

[46] N. R. Franks. Teams in social insects: Group retrieval of prey by army ants. *Behavioral Ecology and Sociobiology*, 18:425–429, 1986.

[47] T. Fukuda, M. Buss, H. Hosokai, and Y. Kawauchi. Cell structured robotic system CEBOT: control, planning and communication methods. *Robotics and Autonomous Systems*, 7(2-3):239–248, 1991.

[48] T. Fukuda and S. Nakagawa. Dynamically reconfigurable robotic system (concept of a system and optimal configurations). In *Proceedings of the 1987 IEEE International Conference on Industrial Electronics, Control and Instrumentation*, pages 588–595. IEEE Computer Society Press, Los Alamitos, CA, 1987.

[49] T. Fukuda and S. Nakagawa. Approach to the dynamically reconfigurable robotic system. *Journal of Intelligent and Robotic Systems*, 1(1):55–72, 1988.

[50] T. Fukuda and S. Nakagawa. Dynamically reconfigurable robotic system. In *Proceedings of the 1988 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1581–1586, Los Alamitos, CA, 1988. IEEE Computer Society Press.

[51] T. Fukuda and S. Nakagawa. Method of autonomous approach, docking and detaching between cells for dynamically reconfigurable robotic system cebot. *JSME International Journal III-VIB. C.*, 33(2):263–268, 1990.

[52] T. Fukuda, S. Nakagawa, Y. Kawauchi, and M. Buss. Self organizing robots based on cell structures—CEBOT. In *Proceedings of the 1988 IEEE International Workshop on Intelligent Robots*, pages 145–150. IEEE Computer Society Press, Los Alamitos, CA, 1988.

[53] T. Fukuda, K. Sekiyama, T. Ueyama, and F. Arai. Efficient communication method in the cellular robotic system. In *Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 1091–1096, Piscataway, NJ, 1993. IEEE Press.

[54] T. Fukuda and T. Ueyama. *Cellular Robotics and Micro Robotic Systems*. 1994.

[55] T. Fukuda, T. Ueyama, and Y. Kawauchi. Self-organization in cellular robotic system (CEBOT) for space application with knowledge allocation method. In *Proceedings of the International Symposium on Artificial Intelligence Robotics and Automation in Space*, pages 101–104, Kobe, Japan, 1990.

[56] T. Fukuda, T. Ueyama, and K. Sekiyama. Distributed intelligent systems in cellular robotics. In S. G. Tzafestas and H. B. Verbruggen, editors, *Artificial Intelligence in Industrial Decision Making, Control and Automation*, pages 225–246. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1995.

[57] S. Funiak, P. Pillai, M. P. Ashley-Rollman, J. D. Campbell, and S. C. Goldstein. Distributed localization of modular robot ensembles. *International Journal of Robotics Research*, 28(8):946–961, 2009.

[58] S. Garnier, J. Gautrais, and G. Theraulaz. The biological principles of swarm intelligence. *Swarm Intelligence*, 1(1):3–31, 2007.

[59] S. Garnier, C. Jost, R. Jeanson, J. Gautrais, M. Asadpour, G. Caprari, and G. Theraulaz. Aggregation behaviour as a source of collective decision in a group of cockroach-like robots. *Advances in Artificial Life*, 3630:169–178, 2005.

[60] D. Goldberg and M. J. Matarić. Interference as a tool for designing and evaluating multi-robot controllers. In *Proceedings of the National Conference on Artificial Intelligence*, pages 637–642. John Wiley & Sons, Hoboken, NJ, 1997.

[61] R. Groß, , M. Dorigo, and M. Yamakita. Self-assembly of mobile robots. From swarm-bot to super-mechano colony. In T. Arai, R. Pfeifer, T. Balch, and H. Yokoi, editors, *Intelligent Autonomous Systems 9 – IAS 9*, pages 487–496. IOS Press, Amsterdam, The Netherlands, 2006.

[62] R. Groß, M. Bonani, F. Mondada, and M. Dorigo. Autonomous self-assembly in swarm-bots. *IEEE Transactions on Robotics*, 22(6):1115–1130, 2006.

[63] 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. Springer-Verlag, Berlin, Germany, 2004.

[64] R. Groß and M. Dorigo. Evolution of solitary and group transport behaviors for autonomous robots capable of self-assembling. *Adaptive Behavior*, 16(5):285–305, 2008.

[65] R. Groß and M. Dorigo. Self-assembly at the macroscopic scale. *Proceedings of the IEEE*, 96(9):1490–1508, 2008.

[66] R. Groß and M. Dorigo. Towards group transport by swarms of robots. *International Journal of Bio-Inspired Computation*, 1(1–2):1–13, 2009.

[67] R. Groß, R. O'Grady, A. L. Christensen, and M. Dorigo. *Handbook of collective robotics: Fundamentals and challenges*, chapter The swarm-bot experience: What you can get from a group of physically cooperating robots. Pan Stanford Publishing, Singapore. To appear.

[68] R. Groß, E. Tuci, M. Dorigo, M. Bonani, and F. Mondada. Object transport by modular robots that self-assemble. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, pages 2558–2564, Los Alamitos, CA, 2006. IEEE Computer Society Press.

[69] J. Halloy, G. Sempo, G. Caprari, C. Rivault, M. Asadpour, F. Tache, I. Said, V. Durier, S. Canonge, J.M. Ame, C. Detrain, N. Correll, A. Martinoli, F. Mondada, R. Siegwart, and J.-L. Deneubourg. Social Integration of Robots into Groups of Cockroaches to Control Self-Organized Choices. *Science*, 318(5853):1155–1158, 2007.

[70] S. Hirose. Super mechano-system: New perspective for versatile robotic system. In *Proceedings of the 7th International Symposium on Experimental Robotics*, pages 249–258, 2000.

[71] S. Hirose, T. Shirasu, and E. F. Fukushima. Proposal for cooperative robot "Gunryu" composed of autonomous segments. *Robots and Autonomous Systems*, 17(1–2):107–118, 1996.

[72] B. Hölldobler, R. C. Stanton, and H. Markl. Recruitment and food-retrieving behavior in novomessor (formicidae, hymenoptera). *Behavioral Ecology and Sociobiology*, 4(2):163–181, 1978.

[73] K. Hosokawa, I. Shimoyama, and H. Miura. Dynamics of self-assembling systems: analogy with chemical kinetics. *Artificial Life*, 1(4):413–427, 1994.

[74] T. L. Huntsberger, A. Trebi-Ollennu, H. Aghazarian, P. S. Schenker, P. Pirjanian, and H. D. Nayar. Distributed control of multi-robot systems engaged in tightly coupled tasks. *Autonomous Robots*, 17(1):79–92, 2004.

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

[76] H. Jacobson. On models of reproduction. *American Scientist*, 46:255–284, September 1958.

[77] R. Jeanson, C. Rivault, J.-L. Denebourg, S. Blanco, R. Fournier, C. Jost, and G. Theraulaz. Self-organized aggregation in cockroaches. *Animal Behavior*, 69(1):169–180, 2005.

[78] C. Jones and M. J. Matarić. From local to global behavior in intelligent self-assembly. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation, ICRA'03*, volume 1, pages 721–726. IEEE Computer Society Press, Los Alamitos, CA, 2003.

[79] A. Kamimura, H. Kurokawa, E. Yoshida, S. Murata, K. Tomita, and S. Kokaji. Automatic locomotion design and experiments for a modular robotic system. *IEEE/ASME Transactions on Mechatronics*, 10(3):314–325, 2005.

[80] Y. Kawauchi, M. Inaba, and Fukuda T. A principle of distributed decision making of celluar robotic system (CEBOT). In *Proceedings of the 1993 IEEE International Conference on Robotics and Automation*, pages 833–838, Piscataway, NJ, 1993. IEEE Press.

[81] E. Klavins, R. Ghrist, and D. Lipsky. A grammatical approach to self-organizing robotic systems. *IEEE Transactions on Automatic Control*, 51(6):949–962, 2006.

[82] M. J. Krieger, J. B. Billeter, and L. Keller. Ant-like task allocation and recruitment in cooperative robots. *Nature*, 406(6799):992–995, 2000.

[83] C. R. Kube and H. Zhang. Stagnation recovery behaviours for collective robotics. In *Proceedings of the 1994 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 1883–1890, Los Alamitos, CA, 1994. IEEE Computer Society Press.

[84] C. R. Kube and H. Zhang. Task modelling in collective robotics. *Autonomous Robots*, 4(1):53–72, 1997.

[85] R. Kube and H. Zhang. Collective robotics: From social insects to robots. *Adaptive Behavior*, 2:189–218, 1993.

[86] J. R. T. Lawton, R. W. Beard, and B. J. Young. A decentralized approach to formation maneuvers. *IEEE Transactions on Robotics and Automation*, 19(6):933–941, 2003.

[87] M. A. Lewis and K. H. Tan. High precision formation control of mobile robots using virtual structures. *Autonomous Robots*, 4(4):387–403, 1997.

[88] A. Lioni and J.-L. Deneubourg. Collective decision through self-assembling. *Naturwissenschaften*, 91(5):237–241, 2004.

[89] A. Lioni, C. Sauwens, G. Theraulaz, and J.-L. Deneubourg. Chain formation in oecophylla longinoda. *Journal of Insect Behaviour*, (15):2001, 2001.

[90] A. Martinoli and F. Mondada. Collective and cooperative group behaviours: Biologically inspired experiments in robotics. In *Proceedings of the 4th International Symposium on Experimental Robotics*, pages 3–10, Berlin, Germany, 1995. Springer-Verlag.

[91] A. Martinoli, M. Yamamoto, and F. Mondada. On the modelling of bio-inspired collective experiments with real robots. In *In Proceeding of the Third International Symposium on Distributed Autonomous Robotic Systems*, pages 25–27. MIT Press, Cambridge, MA, 1997.

[92] N. Mathews, A. L. Christensen, E. Ferrante, R. O'Grady, and M. Dorigo. Establishing spatially targeted communication in a heterogeneous robot swarm. In van der Hoek, Kaminka, Lespérance, Luck, and Sen, editors, *Proceedings of the 9th international Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2010*, pages 939–946. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2010.

[93] M. W. Moffett. Cooperative food transport by an asiatic ant. *National Geographic Research*, 4(3):386–394, 1988.

[94] M. W. Moffett. Ant foraging. *National Geographic Research and Exploration*, 8(2):386–394, 1992.

[95] F. Mondada, M. Bonani, A. Guignard, S. Magnenat, C. Studer, and D. Floreano. Superlinear physical performances in a SWARM-BOT. In *Proceedings of the 8th European Conference on Artificial Life, ECAL 2005*, volume 3630 of *Lecture Notes in Artificial Intelligence*, pages 282–291. Springer-Verlag, Berlin, Germany, 2005.

[96] K. Motomura, A. Kawakami, and S. Hirose. Development of arm equipped single wheel rover: Effective arm-posture-based steering method. *Autonomous Robots*, 18(2):215–229, 2005.

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

[98] S. Murata, K. Kakomura, and H. Kurokawa. Docking experiments of a modular robot by visual feedback. In *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 625–630, Los Alamitos, CA, 2006. IEEE Computer Society Press.

[99] 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.

[100] E. Mytilinaios, M. Desnoyer, D. Marcus, and H. Lipson. Designed and evolved blueprints for physical self-replicating machines. In *Proceedings of the 9th International Conference on the Simulation and Synthesis of Living Systems (Artificial Life IX)*, pages 15–20. MIT Press, Cambridge, MA, 2004.

[101] R. O'Grady, , A. L. Christensen, C. Pinciroli, and M. Dorigo. Robots autonomously self-assemble into dedicated morphologies to solve different tasks. IRIDIA Supplementary page. `http://iridia.ulb.ac.be/supp/IridiaSupp2009-010`, 2009.

[102] R. O'Grady. Adaptive swarm robotics in rough terrain navigation. Mémoire de DEA, Université Libre de Bruxelles, Bruxelles, Belgium, 2005.

[103] R. O'Grady, A. L. Christensen, and M. Dorigo. *Morphogenetic Engineering: Toward Programmable Complex Systems*, chapter SWARMORPH: Morphogenesis with Self-Assembling Robots. "Studies on Complexity" Series (To Appear). Springer-Verlag.

[104] R. O'Grady, A. L. Christensen, and M. Dorigo. Self-assembly and morphology control in a swarm-bot. In E. Grant and T. C. Henderson, editors, *Video Proceedings*

of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 2551–2552. IEEE Computer Society Press, Los Alamitos, CA, 2007.

[105] R. O'Grady, A. L. Christensen, and M. Dorigo. Autonomous reconfiguration in a self-assembling multi-robot system. In M. Dorigo, M. Birattari, C. Blum, M. Clerc, T. Stützle, and A. F. T. Winfield, editors, *Ant Colony Optimization and Swarm Intelligence, Sixth International Conference, ANTS 2008*, volume 5217 of *Lecture Notes in Computer Science*, pages 259–266. Springer-Verlag, Berlin, Germany, 2008.

[106] R. O'Grady, A. L. Christensen, and M. Dorigo. Autonomous reconfiguration in a self-assembling multi-robot system. IRIDIA Supplementary page. `http://iridia.ulb.ac.be/supp/IridiaSupp2008-004`, 2008.

[107] R. O'Grady, A. L. Christensen, and M. Dorigo. SWARMORPH: Multi-robot morphogenesis using directional self-assembly. IRIDIA Supplementary page. `http://iridia.ulb.ac.be/supp/IridiaSupp2008-017`, 2008.

[108] R. O'Grady, A. L. Christensen, and M. Dorigo. SWARMORPH: Multi-robot morphogenesis using directional self-assembly. *IEEE Transactions on Robotics*, 25(3):738–743, 2009.

[109] R. O'Grady, A. L. Christensen, C. Pinciroli, and M. Dorigo. Robots autonomously self-assemble into dedicated morphologies to solve different tasks. In van der Hoek, Kaminka, Lespérance, Luck, and Sen, editors, *Proceedings of the 9th international Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2010*, pages 1517–1518. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2010.

[110] R. O'Grady, R. Groß, A. L. Christensen, and M. Dorigo. Self-assembly strategies in a group of autonomous mobile robots. IRIDIA Supplementary page. `http://iridia.ulb.ac.be/supp/IridiaSupp2008-016`, 2008.

[111] R. O'Grady, R. Groß, A. L. Christensen, and M. Dorigo. Distributed control to implement self-assembly strategies for the hill crossing task. Technical Report TR/IRIDIA/2009-022, IRIDIA, Faculté des Sciences Appliquées, Université Libre de Bruxelles, 2009.

[112] R. O'Grady, R. Groß, A. L. Christensen, and M. Dorigo. Self-assembly strategies in a group of autonomous mobile robots. *Autonomous Robots*, 28(4):439–455, 2010.

[113] R. O'Grady, R. Groß, A. L. Christensen, F. Mondada, M. Bonani, and M. Dorigo. Performance benefits of self-assembly in a swarm-bot. In E. Grant and T. C. Henderson, editors, *Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-2007)*, pages 2381–2387. IEEE Computer Society Press, Los Alamitos, CA, 2007.

[114] R. O'Grady, R. Groß, F. Mondada, M. Bonani, and M. Dorigo. Self-assembly on demand in a group of physical autonomous mobile robots navigating rough terrain. In *Proceedings of the 8th European Conference on Artificial Life, ECAL 2005*, volume 3630 of *Lecture Notes in Artificial Intelligence*, pages 272–281. Springer-Verlag, Berlin, Germany, 2005.

[115] R. O'Grady, C. Pinciroli, A. L. Christensen, and M. Dorigo. Supervised group size regulation in a heterogeneous robotic swarm. In *9th Conference on Autonomous Robot Systems and Competitions, Robótica 2009*, pages 113–119. IPCB-Instituto Politcnico de Castelo Branco, Castelo Branco, Portugal, 2009.

[116] R. O'Grady, C. Pinciroli, R. Groß, A. L. Christensen, and M. Dorigo. Swarm-bots to the rescue. In *10th European Conference on Artificial Life, ECAL 2009*. Springer-Verlag, In Press.

[117] E. H. Østergaard, K. Kassow, R. Beck, and H. H. Lund. Design of the atron lattice-based self-reconfigurable robot. *Autonomous Robots*, 21(2):165–183, 2006.

[118] M. Park, S. Chitta, A. Teichman, and M. Yim. Automatic configuration methods in modular robots. *International Journal of Robotics Research*, 27(3-4):403–421, 2008.

[119] K. Payne, B. Salemi, P. Will, and W.-M. Shen. Sensor-based distributed control for chain-typed self-reconfiguration. In *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 2074–2080, Los Alamitos, CA. IEEE Computer Society Press.

[120] L. S. Penrose and R. Penrose. A self-reproducing analogue. *Nature*, 179(4571):1183, 1957.

[121] A. Perez-Uribe, D. Floreano, and L. Keller. Effects of group composition and level of selection in the evolution of cooperation in artificial ants. In *Proceedings of the 7th European Conference on Artificial Life, ECAL 2003*, volume 2801 of *Lect. Notes Artif. Int.*, pages 128–137. Springer Verlag, Berlin, Germany, 2003.

[122] R. Pfeifer, M. Lungarella, and F. Iida. Self-organization, embodiment, and biologically inspired robotics. *Science*, 318, 2007.

[123] C. Pinciroli, R. O'Grady, A. L. Christensen, and M. Dorigo. Self-organised recruitment in a heteregeneous swarm. In *Fourteenth International Conference on Advanced Robotics – ICAR 2009*, pages 1–8, 2009. Proceedings on CD-ROM, paper ID 176.

[124] J. Pugh and A. Martinoli. Relative Localization and Communication Module for Small-Scale Multi-Robot Systems. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, pages 188–193, 2006.

[125] M. Rubenstein, K. Payne, and P. Will. Docking among independent and autonomous conro self-reconfigurable robots. In *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, volume 3, pages 2877–2882. IEEE Computer Society Press, Los Alamitos, CA, 2004.

[126] M. Rubenstein, Y. Sai, C.-M. Chuong, and W.-M. Shen. Regenerative patterning in swarm robots: mutual benefits of research in robotics and stem cell biology. *International Journal of Developmental Biology*, 53:869–881, 2009.

[127] M. Rubenstein and W.-M. Shen. A scalable and distributed approach for self-assembly and self-healing of a differentiated shape. In *Proceedings of the 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nice, France, 2008. IEEE Computer Society Press.

[128] D. Rus and M. Vona. Self-reconfiguration planning with compressible unit modules. In *Proceedings of the 1999 IEEE International Conference on Robotics and Automation, ICRA'03*, volume 4, pages 2513–2520. IEEE Computer Society Press, Los Alamitos, CA, 1999.

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

[130] 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.

[131] B. Salemi, M. Moll, and W. M. Shen. SUPERBOT: A deployable, multifunctional, and modular self-reconfigurable robotic system. In *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3636–3641, Piscataway, NJ, 2006. IEEE Press.

[132] U. P. Schultz, M. Bordignon, and K. Stoy. Robust and reversible self-reconfiguration. In *Proceedings of the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5287–5294, Piscataway, NJ, 2009. IEEE Press.

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

[134] A.B. Sendova-Franks and N.R. Franks. Self-assembly, self-organization and division of labour. *Philosophical Transactions Royal Society B*, 354(1388):1395–1405, 1999.

[135] W.-M. Shen, M. Krivokon, H. Chiu, J. Everist, M. Rubenstein, and J. Venkatesh. Multimode locomotion for reconfigurable robots. *Autonomous Robots*, 20(2):165–177, 2006.

[136] W.-M. Shen, B. Salemi, and P. Wil. Hormone-inspired adaptive communication and distributed control for conro self-reconfigurable robots. *IEEE Transactions on Robotics and Automation*, 18(5):700–712, 2002.

[137] W. M. Shen, P. Will, A. Galstyan, and C. M. Chuong. Hormone-inspired self-organization and distributed control of robotic swarms. *Autonomous Robots*, 17(1):93–105, 2004.

[138] K. Støy. Using situated communication in distributed autonomous mobile robots. In *Proceedings of the 7th Scandinavian Conference on Artificial Intelligence*, pages 44–52. IOS Press, Amsterdam, The Netherlands, 2001.

[139] K. Støy. Using cellular automata and gradients to control self-reconfiguration. *Robotics and Autonomous Systems*, 54(2):135–141, 2006.

[140] K. Støy and R. Nagpal. Self-reconfiguration using directed growth. In *Proceedings of the International Conference on Distributed Autonomous Robot Systems (DARS-04)*, pages 1–10. Springer Verlag, Berlin, Germany, 2004.

[141] J. H. Sudd. The transport of prey by an ant, *Pheidole crassinoda* Em. *Behaviour*, 16(3–4):295–308, 1960.

[142] G. Theraulaz, E. Bonabeau, C. Sauwens, J.-L. Deneubourg, A. Lioni, F. Libert, L. Passera, and R. Sole. Model of droplet dynamics in the argentine ant linepithema humile (mayr). *Bulletin of Mathematical Biololgy*, 63(6):1079–1093, 2001.

[143] V. Trianni, S. Nolfi, and M. Dorigo. Cooperative hole avoidance in a swarm-bot. *Robotics and Autonomous Systems*, 54(2):97–103, 2006.

[144] V. Trianni, E. Tuci, and M. Dorigo. Evolving functional self-assembling in a swarm of autonomous robots. In *Proceedings of the 8th International Conference on the Simulation of Adaptive Behavior*, pages 405–414. MIT Press, Cambridge, MA, 2004.

[145] E. Tuci, R. Groß, V. Trianni, F. Mondada, M. Bonani, and M. Dorigo. Cooperation through self-assembly in multi-robot systems. *ACM Transactions on Autononomous and Adaptive Systems*, 1(2):115–150, 2006.

[146] G. Vakanas and B. Krafft. Regulation of the number of spiders participating in collective prey transport in the social spider anelosimus eximius (araneae, theridiidae). *Comptes Rendus Biologies*, 327(8):763–772, 2004.

[147] P. White, V. Zykov, J. Bongard, and H. Lipson. Three dimensional stochastic reconfiguration of modular robots. In *Proceedings of Robotics Science and Systems*, pages 161–168. MIT Press, Cambridge, MA, 2005.

[148] P. J. White, K. Kopanski, and H. Lipson. Stochastic self-reconfigurable cellular robotics. In *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, volume 3, pages 2888–2893. IEEE Computer Society Press, Los Alamitos, CA, 2004.

[149] M. Yamakita, Y. Taniguchi, and Y. Shukuya. Analysis of formation control of cooperative transportation of mother ship by SMC. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*, volume 1, pages 951–956. IEEE Computer Society Press, Los Alamitos, CA, 2003.

[150] M. Yim, D. Duff, and K. Roufas. Walk on the wild side. *IEEE Robotics and Automation Magazine*, 8(4), December 2002.

[151] M. Yim, D. Duff, and K. D. Roufas. Polybot: a modular reconfigurable robot. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, volume 1, pages 514–520, Piscataway, NJ, 2000. IEEE Press.

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

[153] M. Yim, K. Roufas, D. Duff, Y. Zhang, C. Eldershawand, and S. Homans. Modular reconfigurable robots in space applications. *Autonomous Robots*, 14:225–237, 2003.

[154] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian. Modular self-reconfigurable robot systems. *IEEE Robotics and Automation Magazine*, 14(1):43–52, 2007.

[155] M. Yim, Y. Zhang, and D. Duff. Modular robots. *IEEE Spectrum Magazine*, 39(2):30–34, Feb. 2002.

[156] M. Yim, Y. Zhang, K. Roufas, D. Duff, and C. Eldershaw. Connecting and disconnecting for chain self-reconfiguration with PolyBot. *IEEE/ASME Transactions on Mechatronics*, 7(4):442–451, 2002.

[157] E. Yoshida, S. Murata, A. Kamimura, K. Tomita, H. Kurokawa, , and S. Kokaji. A self-reconfigurable modular robot: Reconfiguration planning and experiments. *International Journal of Robotics Research*, 21(1011):903–915, 2002.

[158] C.-H. Yu and R. Nagpal. Sensing-based shape formation on modular multi-robot systems: a theoretical study. In *Proceedings of the 7th Internation Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, pages 71–78. ACM New York, NY, 2008.

[159] C.-H. Yu and R. Nagpal. Self-adapting modular robotics: A generalized distributed consensus framework. In *ICRA*, pages 1881–1888, 2009.

[160] C.-H. Yu, F.-X. Willems, D. Ingber, and R. Nagpal. Self-organization of environmentally-adaptive shapes on a modular robot. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems 2007 (IROS 2007)*, pages 2353–2360, Piscataway, NJ, 2007. IEEE Press.

[161] Y. Zhang, K. Roufas, C. Eldershaw, M. Yim, and D. Duff. Sensor computations in modular self reconfigurable robots. In *Proceedings of the 8th International Symposium on Experimental Robotics*, volume 5, pages 276–286. Springer Verlag, Berlin, Germany, 2003.

[162] V. Zykov, E. Mytilinaios, B. Adams, and H. Lipson. Self-reproducing machines. *Nature*, 435(7039):163–164, 2005.

List of Figures

List of Tables