An Autonomous Multi-Robot System for Stigmergy-Based Construction

Michael Allwright

A dissertation submitted to the
Department of Computer Science
Paderborn University

for the degree of
Doktor der Naturwissenschaften
(doctor rerum naturalium)

September 8, 2017

Thesis supervisor: Prof. Marco Dorigo, Ph.D.
Thesis co-supervisor: Dr. Navneet Bhalla, Ph.D.

Swarm Intelligence Group
Department of Computer Science
Paderborn University

Heinz Nixdorf Institute
Fürstenallee 11
33102 Paderborn, Germany
Phone +49 5251 60-6200
hni@hni.upb.de
www.hni.uni-paderborn.de

Abstract

Original version in English

An autonomous construction system (ACS) is envisioned to be a solution to the construction of structures in environments that are too hazardous for humans and where remote operation is not possible. Drawing inspiration from the construction behavior of social insects, we advance the state of the art by proposing a completely decentralized control strategy for a multi-robot ACS. This control strategy is based on previous work by Theraulaz and Bonabeau, who demonstrated multi-agent construction using a three-dimensional lattice-based simulation.

In order to study the proposed decentralized control strategy, we set about designing a multi-robot ACS that consists of two components, a building material and autonomous robots. In our multi-robot ACS, an autonomous robot is capable of locating the building material in its environment, picking it up, transporting it, and attaching it to a structure. The multi-robot ACS is designed to be completely autonomous and capable of constructing three-dimensional structures. To test and evaluate our multi-robot ACS and its decentralized control strategy, we provide two implementations of our system. The primary implementation is realized using hardware and the secondary implementation is realized using simulation.

Through the use of both implementations, we demonstrate how our decentralized control strategy can be used to coordinate the autonomous construction of three-dimensional structures. After discussing these demonstrations, we conclude this thesis by suggesting future research directions.

Übersetzung ins Deutsche

Ein autonomes Bausystem ist eine Lösung für den Bau von Strukturen in Umgebungen, die für Menschen zu gefährlich ist und wo auch das Fernbedienen nicht möglich ist. Inspiriert durch das Bauverhalten von sozialen Insekten, treiben wir den Stand der Technik voran und schlagen eine vollständig dezentralisierte Steuerungsstrategie für ein Multi-Roboter-Bausystem vor. Diese Vorgehensweise basiert auf der bisherigen Arbeit von Theraulaz und Bonabeau, die eine dreidimensionale Gitter-basierte Simulation verwendet haben um Bauverfahren mit einem Multiagentensystem zu demonstrieren.

Um die vorgeschlagene dezentralisierte Steuerungsstrategie zu untersuchen, wurde ein Multi-Roboter-Bausystem entwickelt. Dieses Bausystem besteht aus zwei Komponenten: dem Baustoff und autonomen Robotern. Ein autonomer Roboter in unserem Multi-Roboter-Bausystem ist in der lage den Baustoff in seiner Umgebung zu lokalisieren, aufzuheben, zu transportieren, und an einer Struktur anzubringen. Das Multi-Roboter-Bausystem wurde mit dem Schwerpunkt auf ein völlig autonomes Bauen von dreidimensionalen Strukturen entwickelt. Um unser Multi-Roboter-Bausystem und seine dezentralisierte Steuerungsstrategie zu testen und zu bewerten, werden zwei Implementierungen vorgestellt: Eine Hardware-basierte Realisierung und eine Implementierung mittels Simulation.

Durch den Einsatz beider Implementierungen wird demonstriert, wie unsere dezentralisierte Steuerungsstrategie autonome Konstruktion von dreidimensionalen Strukturen koordinieren kann. Nach der Diskussion dieser Demonstrationen schließen wir diese Arbeit mit Ausblick auf zukünftige Forschungsmöglichkeiten ab.

Acknowledgements

The research presented in this thesis was completed under the supervision of Marco Dorigo and Navneet Bhalla. I am sincerely grateful to Marco and Navneet for their time and continuous support over the past years while this research was being undertaken. Their support has assisted me with keeping the research on track, formulating my ideas clearly, and preparing content for submission in highly regarded journals and conferences.

The simulation work in this thesis was made possible by Carlo Pinciroli, who assisted me with his ongoing development of the ARGoS simulator. I am also very grateful to Haitham El-faham, who completed the mechanical design for a prototype of the autonomous robot's manipulator as part of his Master's thesis. Furthermore, I would like to thank Haitham El-faham for developing the magnetism plugin for the ARGoS simulator. I am also grateful to Anthony Antoun, with whom I collaborated to create the first prototypes of the stigmergic block.

Furthermore, I would like to thank Christoph Scheytt and Uwe von der Ahe from the System and Circuit Technology Research Group and Eric Klemp and Michael Brand from the Direct Manufacturing Research Center (DMRC) at Paderborn University for their technical support and for providing access to the facilities which were used to evaluate the first prototypes of the hardware described in this thesis.

I greatly appreciate the assistance from Juergen Maniera at Paderborn University, who supported this research by organizing our budget, tracking our purchase orders, and managing our outsourced manufacturing requests. I would also like to thank the International Graduate School of Dynamic Intelligent Systems and, in particular, Eckhard Steffen and Astrid Canisius for the organization and support with the administrative aspects of the program at Paderborn University.

Finally, I am very grateful to my mother, Rosemarie Allwright, who raised me to always ask questions, nurtured my curiosity about the world, and continuously supported me from the very beginning.

Contents

CHAPTER 1

Introduction

1.1 Construction by autonomous robots

1.1.1 Motivation

Robotic construction systems are envisioned to be one solution for building permanent and temporary structures in environments that are too hazardous for humans and as such, are an area of research of NASA's Jet Propulsion Laboratory [31–33, 90–92]. Such construction systems could be autonomous or could be remotely operated by a human. While an autonomous construction system (ACS) is more complex to design and program, it is able to function without a human operator. This autonomy is necessary in environments where wireless signals can not penetrate (e.g. the ocean floor, or in underground mines) or where the latency of the signal prevents real-time operation (e.g. between Earth and a non-terrestrial body).

1.1.2 Classification of ACS configurations

We classify the configuration of an ACS using the following criteria: the number of types of robots used; the total number of robots performing construction; and the type of building material. We categorize building materials as either passive, semi-active, or active. In this work, we define passive building materials as neither containing electronics nor being capable of locomotion. Furthermore, we define semi-active building materials as containing electronics but not being capable of locomotion, and active building materials as containing electronics and being capable of locomotion. There are three common ACS configurations:

Single-robot ACS: consists of a single robot that arranges passive or semi-active building materials into structures.

Self-assembly ACS: consists of active building materials (or robots) that rearrange themselves into structures[1].

Multi-robot ACS: consists of multiple robots that cooperate to arrange building materials into structures. The building materials may be passive or semi-active.

[1]We restrict our definition of a self-assembly ACS to only cover systems that use active building materials, i.e. building materials that are capable of locomotion.

Configuration	Advantages	Disadvantages
Single-robot ACS	• Less complex to develop and program • Low unit cost when scaling to larger structures	• Low redundancy (robot is an SPOF)
Self-assembly ACS	• High redundancy	• High unit cost when scaling to larger structures • Highly complex to develop and program
Multi-robot ACS	• Moderate redundancy • Moderate unit cost when scaling to larger structures	• Moderately complex to develop and program

Table 1.1: Advantages and disadvantages of various ACS configurations.

Each of these configurations has advantages and disadvantages as summarized in Table 1.1. The use of passive building materials in a single-robot ACS makes this configuration a cost effective solution when scaling to large structures. These passive building materials are often rigid, although there is also research into the use of compliant or amorphous materials (Figure 1.1). For example, Khoshnevis discussed the application of contour crafting, an additive fabrication technology, to automated construction [37]. Furthermore, Napp et al. investigated ramp construction using three types of amorphous building material: toothpicks and glue, sand bags, and casting foam [68, 69].

The disadvantage of a single-robot ACS, however, is that the robot is a single point of failure (SPOF), which may require human intervention in the case of a malfunction. This intervention is not always possible and may lead to significant economic loss, as was the case with the Mars rover, Spirit [55].

Self-assembly ACSs overcome SPOFs by distributing the required autonomy across the building materials. This distribution of the autonomy makes self-assembly ACSs a potentially robust solution as such ACSs may continue to self assemble even if some of their components have malfunctioned. For a self-assembly ACS to be completely autonomous, the building materials must be active and capable of locomotion. One example of such a self-assembly ACS is in the work of O'Grady et al., who demonstrated morphogenesis using *S-bot* robots [70]. In a sense, the robots in this work were building materials, which moved around the arena, forming physical connections with each other to assemble into a larger, composite robotic entity (Figure 1.2). In [79, 80], Romanishin et al. demonstrated an alternative approach to locomotion using *M-Blocks* (Figure 1.3).

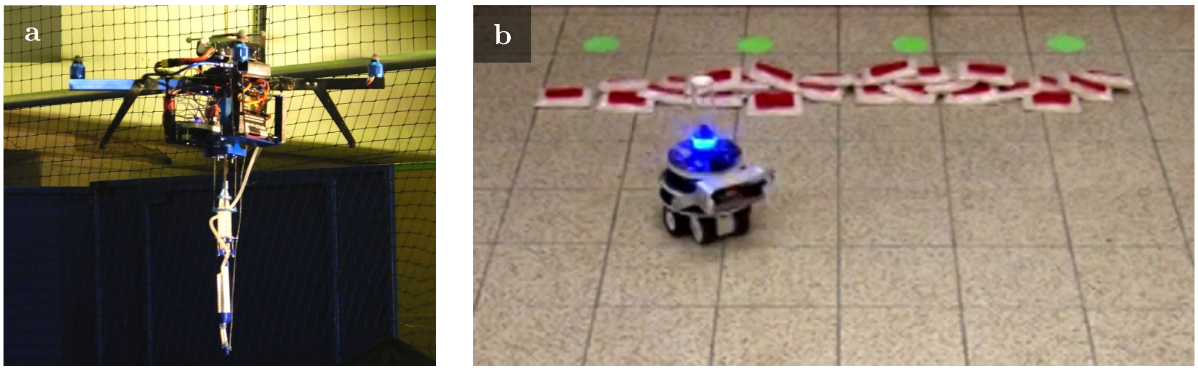


Figure 1.1: Construction using compliant or amorphous materials. (a) A hovering quadcopter equipped with a casting foam extruder (reprinted from [30] with permission, © 2014, IEEE). (b) A wall built from compliant pockets filled with dried rice (reprinted from [84] with permission from Marco Dorigo).

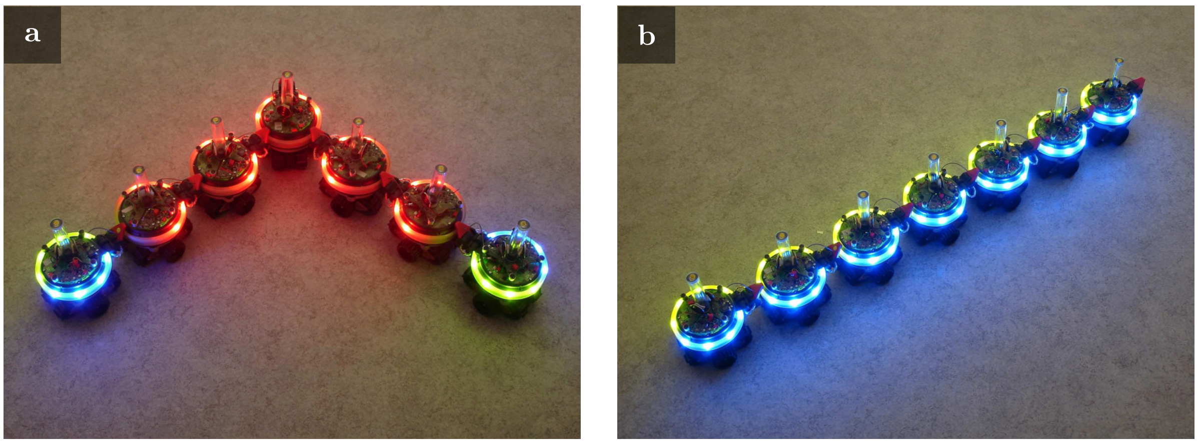


Figure 1.2: A self-assembly ACS consisting of S-Bot robots assemble to form (a) an arrow and (b) a line (reprinted from [14] with permission, © 2007, IEEE).

An *M-Block* stores angular momentum internally using its flywheel, which it uses to spin, jump, and move around its environment.

Since the autonomy and locomotive capabilities of a self-assembly ACS are distributed across all the build materials, this approach is not suitable for scaling to larger, more permanent structures due to the high unit cost associated with active building materials. Furthermore, the complexity to develop and program a self-assembly ACS may increase with the size of the structure due to the increase in the total number of active components.

We define a multi-robot ACS as an ACS with at least two distinct components: a building material, which is either passive or semi-active, and autonomous robots, which cooperate to assemble the building material into structures. A multi-robot ACS may

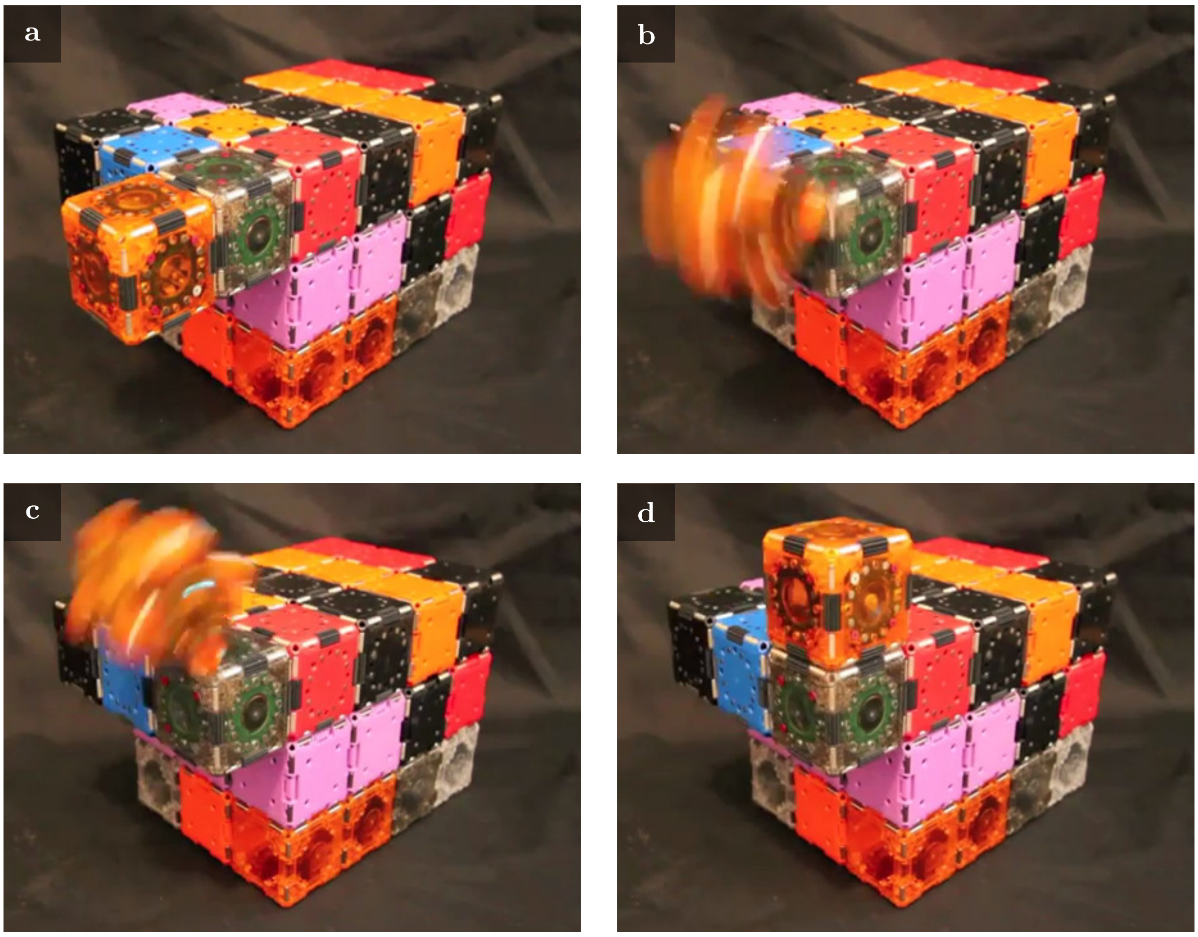


Figure 1.3: The M-Blocks are an example of a building material with locomotion capabilities, which is a requirement of a completely autonomous self-assembly ACS (reprinted from [79] with permission, © 2015, IEEE).

be seen as a compromise between a single-robot ACS and a self-assembly ACS. The use of multiple robots in a multi-robot ACS potentially increases robustness while the use of either passive or semi-active building materials reduces the overall system cost when scaling to larger structures.

1.1.3 Control strategies

Self-assembly and multi-robot ACSs require more complex control strategies than single-robot ACSs. This increase in complexity arises from the requirement that the actions of the individual robots in a self-assembly or multi-robot ACS must be coordinated. For a multi-robot ACS, a class of control strategies exists, which relies on centralized infrastructure, consisting of a positioning system and a central server. This server monitors a multi-robot ACS using the positioning system and regulates it by scheduling and assigning actions to individual robots over a wireless network. A disadvantage of this control strategy class is that the centralized infrastructure introduces one or more SPOFs into an ACS.

An alternative approach to the coordination of a multi-robot ACS is swarm robotics. Similar to social insects in nature, the coordination in a swarm robotics system emerges as a result of individual robots communicating directly with their neighbors and indirectly through their environment.

1.2 Construction in nature

Swarm robotics is based on the principles of swarm intelligence. A swarm intelligence system can be identified by the following characteristics [21]:

- it is composed of many individuals,
- the individuals are relatively homogeneous (i.e., they are either all identical or they belong to a few typologies),
- the interactions among the individuals are based on simple behavioral rules that exploit only local information that the individuals exchange directly or via the environment, and
- the overall behavior of the system results from the interactions of individuals with each other and with their environment, that is, the group behavior self-organizes.

In nature, swarm intelligence describes collective animal behavior such as the flocking of birds, the schooling of fish, and the self-organization of social insects. Figure 1.4 provides two examples of social insects who cooperate to construct their nests. These nests provide advanced architectural features such as defense against predators and homeostasis through thermoregulation and gas exchange [12]. These nests are built by swarms of insects, which primarily coordinate their activities through stigmergy. Stigmergy is a communication mechanism, which was first observed in nature by Grassé during a study of the collective construction behavior of termites [25]. Grassé noted that termites communicate indirectly through modifications of a shared environment.

There are two forms of stigmergy: quantitative and qualitative [99]. In quantitative stigmergy, indirect communication is mediated by the intensity of a stigmergic signal. This means the probability to respond to a stimulus is proportional to the intensity of that stimulus. This form of stigmergy is often facilitated by pheromones and is used to explain the formation of various structural elements in termite nests. For example, Bruinsma observed that there were three types of pheromones used to regulate the construction of a termite royal chamber: (i) a pheromone emitted by termites to form a trail to a construction site, which recruits other termites, (ii) a pheromone emitted by a queen termite, which creates a template for a royal chamber, and (iii) a pheromone added by termites to soil pellets during construction, which signals recent construction activity [11]. This last use of pheromones is depicted in Figure 1.5(a). These observations

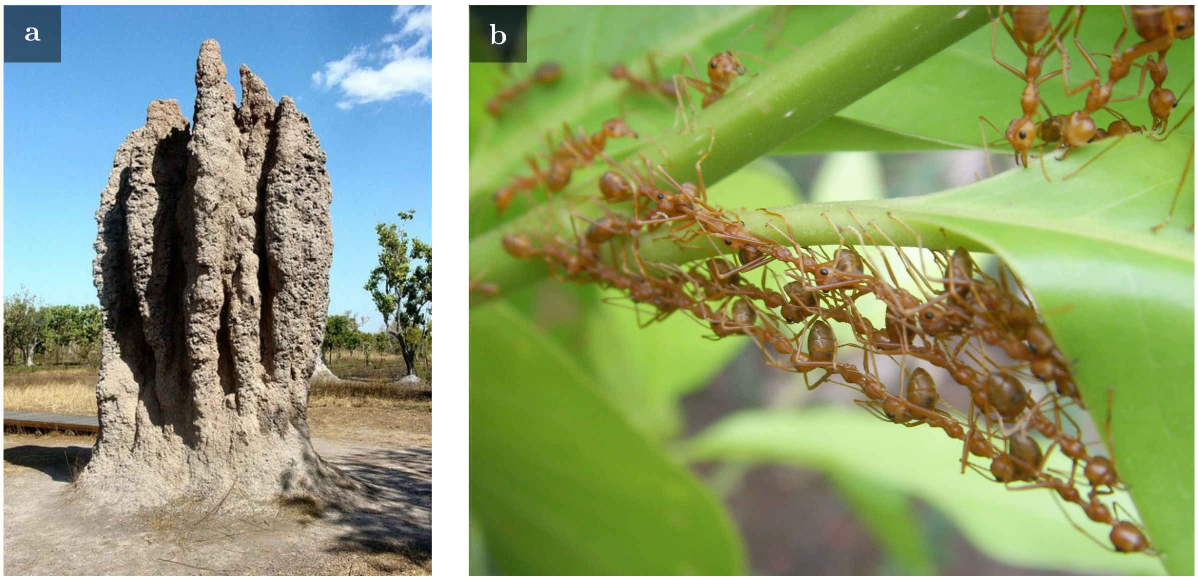


Figure 1.4: Collective construction by social insects. (a) A termite cathedral in Northern Territory, Australia (photo by Brian Voon Yee Yap, reprinted with permission, CC-BY-SA 3.0) and (b) weaver ants *Oecophylla smaragdina* stitch leaves together to build their nests in Thailand (photo by Sean Hoyland, reprinting permission not required).

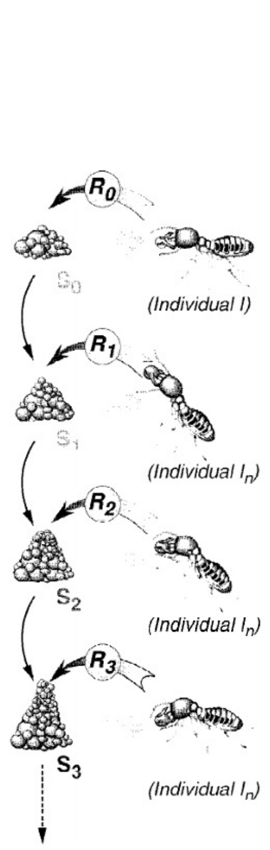
led to mathematical models for the emergence of pillars, walls and royal chambers in termite nests [8], enabling further work in simulation, which demonstrated the role of physical constraints on the construction of chambers and tunnels [42, 43].

In contrast to quantitative stigmergy, qualitative stigmergy is mediated by the spatial characteristics of a stigmergic signal. In other words, the probability of performing a given action is a function of a perceived configuration in an environment. An example of this type of stigmergy is depicted in Figure 1.5(b) where the construction behavior of a social wasp is regulated by the shape of a partially-built comb [36].

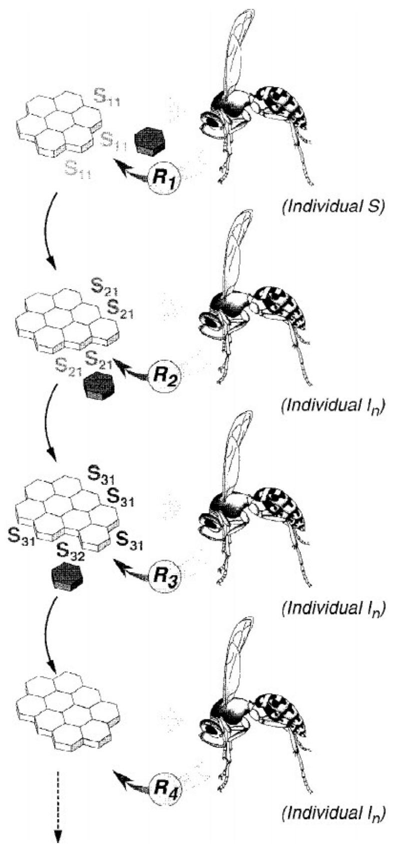
1.3 Problem statement

Despite examples of decentralized construction in nature by social insects, there is limited research on how a decentralized control strategy, in particular, one based on the principles of swarm intelligence, could be implemented on a multi-robot ACS. In this thesis, we propose a decentralized control strategy for a multi-robot ACS, which is based on the work of Theraulaz and Bonabeau [98].

Theraulaz and Bonabeau presented a decentralized control strategy for multi-agent construction based on the behavior of social insects. They simulated agents that searched a three-dimensional lattice for predefined patterns consisting of two types of bricks.



(a)



(b)

Figure 1.5: Coordination of construction through stigmergy. (a) The construction of a pillar by termites is coordinated through quantitative stigmergy. The strength of the stigmergic signal increases with the size of pillar, stimulating further construction. (b) The construction of a comb by social wasps is coordinated through qualitative stigmergy. The spatial characteristics of the stigmergic signal regulate further construction (reprinted from [99] with permission, © 1999, MIT).

When an agent detected a predefined pattern, it placed another brick at its current location. This behavior resulted in a feedback loop where brick placement modified the lattice, and modifications to the lattice resulted in the formation of different patterns. By carefully selecting the patterns that triggered the placement of a brick, Theraulaz and Bonabeau demonstrated a control strategy for coordinated construction, which led to the structures in Figure 1.6.

As the simulated agents in the lattice influenced each other's behavior by placing bricks, these agents were coordinating their actions by communicating indirectly through a shared environment. As discussed in Section 1.2, this type of communication is called stigmergy. Furthermore, since the communication was facilitated through the spatial arrangement of the bricks in the lattice, this type of communication is an example of qualitative stigmergy.

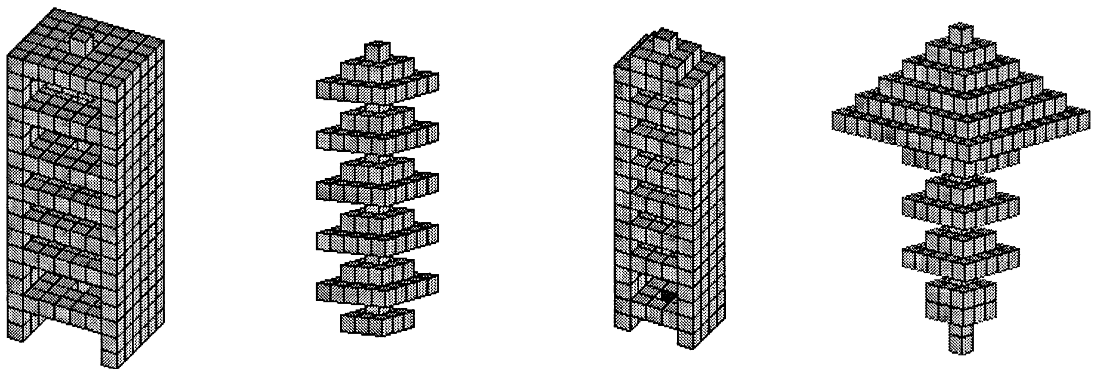


Figure 1.6: Generated structures by Theraulaz and Bonabeau (reprinted from the Journal of Theoretical Biology, 177.4, Guy Theraulaz and Eric Bonabeau, Modelling the Collective Building of Complex Architectures in Social Insects with Lattice Swarms, pages 381–400, © 1995, with permission from Elsevier).

Several adaptations are required to implement Theraulaz and Bonabeau's control strategy on a multi-robot ACS. These adaptations must, for example, compensate for the differences in perception and in maneuverability between simulated agents and physical robots. The decentralized control strategy that we propose in this thesis is the result of these adaptations. Since it is our long-term objective to develop a swarm robotics construction system, we additionally require that the implementation of our decentralized control strategy is compatible with the following design constraints for swarm robotics systems as described by Brambilla et al. [10]:

- the robots are autonomous,
- the robots are situated in the environment and can act to modify it,
- sensing and communication capabilities of the robots are local,
- the robots do not have access to centralized control and/or global knowledge,
- the robots cooperate to perform a given task.

While these design constraints may seem restrictive, the literature on multi-robot clustering suggests that multi-robot systems, which are compatible with these constraints, have several advantages. For example, Beckers et al. have noted from their work that the swarm robotics approach to clustering was robust, as failed robots became obstacles; and scalable, as robots could be added and removed from the system without reprogramming [4]. Furthermore, Deneubourg et al. have observed in their work on multi-robot clustering that since communication was indirect and occurs through the environment, the system was capable of performing clustering in various settings without specific programming [16].

To demonstrate that our decentralized control strategy can coordinate construction, we propose the design of a multi-robot ACS, which has two implementations. The first implementation is based on hardware and consists of autonomous robots and a building material, which the robots can assemble into structures. This implementation aims to demonstrate that our decentralized control strategy can be used to coordinate construction in a physical system. The second implementation is built on top of a multi-robot simulator, for which we develop several extensions to support our work. These extensions enable us to accurately model the hardware from the first implementation in simulation. This implementation in simulation aims to provide an environment for efficiently developing, debugging, and testing our decentralized control strategy.

Both of these implementations come with their respective challenges. The hardware implementation, for instance, must be reliable and should be manufactured within tolerances that do not require the calibration of each robot. Alternatively, if calibration is unavoidable, it should at least be partially automated to ease the running of large experiments. These design constraints should be achieved while minimizing cost, as the unit cost of the autonomous robots and the building material directly impacts the extent to which we can scale up to larger experiments. The implementation in simulation requires accurate modeling of the three-dimensional dynamics and behavior of the hardware. This is a necessary requirement if the simulations are to provide meaningful insights during the development, debugging, and testing of our decentralized control strategy. To this end, the range and accuracy of the virtual sensors and actuators used in the simulation must also be tuned to match the performance of their counterparts in the hardware. Furthermore, since we intend to use simulation to gather data from experiments with numerous autonomous robots and large amounts of building material, we should also select a simulator that is capable of running such experiments in real-time or faster than real-time if possible.

1.4 Contributions and publications

In the following sections, we outline the contributions that are presented in this thesis. We also indicate, where applicable, the conference paper or journal article in which we have published a given contribution.

1.4.1 A review of multi-robot ACSs

In this thesis, we provide a comprehensive review and comparison of multi-robot ACSs that have been implemented using hardware. Over the course of this review, we present the state of the art and discuss the advantages and disadvantages of the current solutions. We have published a related review as part of the following conference paper:

> Michael Allwright, Navneet Bhalla, Haitham El-faham, Anthony Antoun, Carlo Pinciroli, and Marco Dorigo. "SRoCS: Leveraging Stigmergy on a Multi-robot Construction Platform for Unknown Environments". In: *Proceedings of the Ninth International Conference on Swarm Intelligence.* Springer, 2014, pages 158–169.

In this paper, we discussed the concept of autonomous construction in the context of *unknown environments*. We defined unknown environments as potential construction sites, which were *unstructured* (no pre-existing infrastructure) and *uncharted* (no survey or mapping data available). Based on observational research of social insects, we proposed in this paper that using a decentralized control strategy based on stigmergy would enable construction in these environments. To this end, we reviewed the existing literature on multi-agent construction, highlighting where and discussing how stigmergy had been used. We concluded that the systems that used a control strategy that was viable in unknown environments were limited to the construction of loose walls and clusters. Following the review, we proceeded to discuss the concept of a multi-robot ACS, which we implement in this thesis.

1.4.2 A decentralized control strategy

The main contribution of this thesis is our decentralized control strategy for coordinated construction, which we base on the work of Theraulaz and Bonabeau. We demonstrate our decentralized control strategy using a multi-robot ACS, which we design and implement using hardware and simulation. For the hardware implementation, we detail the design of the two base components of our system: the stigmergic block and the autonomous robot. Furthermore, we verify the functionality of these components using two tasks. An abridged version of the hardware design and verification work in this thesis is available in our upcoming conference paper, which at the time of writing is under review:

> Michael Allwright, Navneet Bhalla, and Marco Dorigo. "Structure and Markings as Stimuli for Autonomous Construction". In: *The Eighteenth International Conference on Advanced Robotics.* (Under review)

The implementation in simulation uses ARGoS, which is a modular, multi-engine multi-robot simulator designed for running experiments with large numbers of robots [77, 78]. We develop a number of extensions for this simulator, which allow us to accurately model the stigmergic block and the autonomous robot. These extensions are detailed in the following article, which at the time of writing is under review:

> Michael Allwright, Navneet Bhalla, Carlo Pinciroli, and Marco Dorigo. "Towards Autonomous Construction using Stigmergic Blocks". In: *Autonomous Robots.* (Under review)

In this article, we also demonstrate how our decentralized control strategy scales to multi-robot construction using the implementation in simulation.

1.4.3 Design of a stigmergic block

In this thesis, we present the design of a stigmergic block, an advanced cubic building material capable of computation, data storage, and communication. The stigmergic blocks are the building material, which the autonomous robots assemble into structures. Each stigmergic block contains four multi-color LEDs on each of its faces. The robots can detect and configure the color of these LEDs, which facilitates a medium for indirect communication between the robots (stigmergy).

We have designed the stigmergic blocks to simplify the actuation and sensing requirements of the autonomous robots so that we can focus our work on developing decentralized control strategies for a multi-robot ACS. This simplification is partially achieved by attaching localizable tags to a stigmergic block, which enables an autonomous robot to accurately locate it in an environment. Furthermore, we have also added a freely-rotating, spherical magnet into each corner of a block to enable self-alignment and to reduce cumulative misalignment during construction. These spherical magnets also increase the strength of a structure.

To enable other researchers to develop similar hardware, this thesis includes a comprehensive overview of the electronics, mechanical design, and software of the stigmergic block.

1.4.4 Design of an autonomous robot

Our autonomous robot consists of a specialized manipulator and a mobile robotics platform, which is a significantly upgraded version of the BeBot [24, 26]. The upgrade to the BeBot was required due to its main microprocessor, whose performance was inadequate with respect to our computer vision requirements.

The specialized manipulator is designed to pick up an unused stigmergic block and to assemble it into a structure. The manipulator operates by controlling the position of an end-effector consisting of four semi-permanent electromagnets. These electromagnets couple with the spherical magnets in the corners of a stigmergic block.

In this thesis, we discuss the design, manufacturing, and assembly of the upgraded mobile robotics platform and the specialized manipulator. We also detail the software that we have developed for these two components to enable autonomous construction. This software includes but is not limited to: a packet control interface, an image processing pipeline, a finite state machine, and a control loop.

1.4.5 Extensions to ARGoS

We present a number of extensions to ARGoS, which enable us to model the autonomous robot and the stigmergic block in simulation. By providing these models with controllers, we are able to implement our multi-robot ACS in simulation, which provides an environment for the development, debugging, and testing of our decentralized control strategy.

The extensions we have developed for ARGoS allow us to accurately model the three-dimensional dynamics of our multi-robot ACS. This stands in contrast to the simulation tools used in the literature for multi-robot construction, which are lattice-based. This level of accuracy aims to enable the use of the same controller for the autonomous robot on both the hardware and in simulation.

1.4.6 Demonstrations of our control strategy

Through the use of both hardware and simulation, we demonstrate our decentralized control strategy by showing how an autonomous robot is able to perform construction in response to visual cues in an environment.

Using both hardware and simulation, we demonstrate the construction of a staircase, containing six stigmergic blocks, by an autonomous robot. In addition to proving that our decentralized control strategy can be used for autonomous construction, this demonstration also shows that the same controller for an autonomous robot can be used on both the hardware and in simulation.

We also demonstrate how our decentralized control strategy scales to multi-robot construction by using two robots to build a column using the hardware and by using four robots to build a stepped pyramid in simulation. Based on these results, we provide insights into the strengths and the shortcomings of our decentralized control strategy and our multi-robot ACS.

1.5 Thesis structure

This thesis is structured as follows. Chapter 2 provides a comprehensive review and comparison of multi-robot ACSs. In this chapter, we present the state of the art and discuss the advantages and disadvantages of the presented solutions. Following this review, we discuss in Chapter 3 the different approaches to the design of a multi-robot ACS and how we adapt Theraulaz and Bonabeau's control strategy so that it can be implemented on a multi-robot ACS.

Chapter 4 details the implementation of our multi-robot ACS using hardware. This discussion is split into two sections for the two main components of our system: the stigmergic block and the autonomous robot. Each of these sections details the electronics, mechanical design, and software of the respective component. In Chapter 5, we discuss the implementation in simulation. This discussion begins with an overview of ARGoS, before discussing the extensions, which enable the implementation of our multi-robot ACS in simulation.

In Chapter 6, we provide demonstrations of our decentralized control strategy the hardware implementation and the implementation in simulation. Conclusions are presented in Chapter 7, where we summarize the contributions of this thesis and discuss research directions for future work.

CHAPTER 2

Related Work in Multi-Robot Construction

2.1 Overview

2.1.1 Scope

In this section, we provide an extensive survey of the literature on multi-robot ACSs. This survey is, however, limited to systems which have been implemented using hardware. To this end, the use of simulation for studying multi-robot ACSs is only reported if it is used to support the development of hardware and/or a control strategy or if it demonstrates how a multi-robot ACS scales to larger experiments, involving numerous robots and large amounts of building material.

We have included this survey to provide context for our multi-robot ACS and its decentralized control strategy, which we present in the later sections of this thesis.

2.1.2 Structure

We have split the surveyed literature into two parts. Section 2.2 discusses multi-robot ACSs that leverage some form of centralized infrastructure for coordination. As discussed in Section 1.1.3, this centralized infrastructure typically consists of a positioning system and a central server, which schedules and delegates construction tasks to robots.

Section 2.3 discusses multi-robot ACSs that do not rely on any form of centralized infrastructure. These systems coordinate construction using a decentralized control strategy based on local information. This information may include perception of the surrounding environment using a robot's onboard sensors, direct communication with nearby robots, and indirect communication with other robots through the environment.

We make this distinction in part due to complexities of coordinating construction without centralized infrastructure. In this case, any information about the state of an environment is distributed among the robots and can not be directly accessed by any individual robot. This information includes the number, the location, and the state of the robots and the structures in an environment.

Furthermore, multi-robot ACSs that use centralized infrastructure aim to answer different research questions to multi-robot ACSs that do not use centralized infrastructure. In the case of multi-robot ACSs that use centralized infrastructure, the research questions can often be categorized as one of the following:

- perceiving the task and localizing parts,
- partitioning, scheduling, and allocating tasks to robots,
- detecting faults in the assembly process, and
- collectively transporting and manipulating parts.

In contrast, the research into multi-robot ACSs that do not use centralized infrastructure aim to answer research questions that can usually be categorized as one of the following:

- studying self-organization mechanisms found in nature and applying them as a decentralized control strategy for a multi-robot ACS
- measuring and tuning the performance of decentralized control strategies for clustering, sorting, and the formation of structures,
- coordinating the construction process based on local information,
- implementing construction which is adaptive to templates in the environment, and
- collectively transporting and manipulating parts.

2.1.3 Focus

In the following sections, we provide a high-level description and detail the components of each of the relevant multi-robot ACSs in the literature. In addition, we detail the contributions of the published work associated with these multi-robot ACSs.

For the high-level description, we detail the layout of the construction environment such as the locations of the parts for construction and whether a positioning system or wireless network was set up to facilitate coordination. We also note the types of structures that can be built by a given multi-robot ACS, and how those structures are represented by the system prior to construction.

For the robots, we report the number of robots used in the experiment and whether the robots were homogeneous or heterogeneous. For each type of robot, we describe the types of sensors and actuators used. We specify the roles of each robot in the system, drawing attention to whether these roles were assigned statically or dynamically. In the case of the building materials, we report whether the building material was homogeneous or heterogeneous and whether it was passive, semi-active, or active.

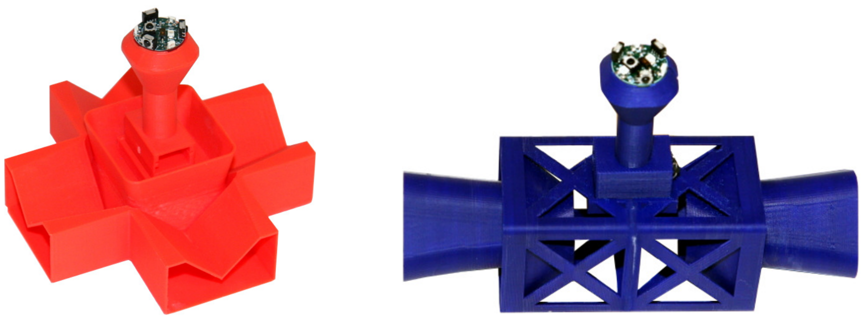


Figure 2.1: Smart parts used by the DRL for coordinated construction. (a) A connector and (b) a truss (modified and reprinted from [81] with permission from Daniela Rus).

2.2 Construction using centralized infrastructure

2.2.1 Coordinated construction by the Distributed Robotics Laboratory (DRL) at CSAIL MIT

The iRobot Create [15] is a mobile robotics platform based on the Roomba autonomous vacuum cleaner. Bolger et al. equipped four iRobot Creates with CrustCrawler robotic arms and Dell Inspiron Mini 10s netbooks for their work in coordinated construction [6]. The main contribution of this work was an algorithm for the *equal-mass* partitioning of a construction task such that the tasks were distributed evenly among the robots. The robots were assigned the role of either part delivery or part assembly. There were two types of building materials: trusses and connectors (Figure 2.1), which were made available from a part cache. The building materials contained a small infrared (IR) transmitter, which helped the gripper align with a part. In addition, a VICON motion capture system[1] tracked the locations of all robots and broadcasted this information over a mesh network. A wireless router facilitated robot-to-robot communication.

In [130], Yun et al. demonstrated their algorithm for equal-mass partitioning in simulation, constructing a two-dimensional A-shaped bridge and a three-dimensional airplane. Several extensions to the equal-mass partitioning algorithm were presented in related work. For example, Yun and Rus extended the algorithm to allow for robot failures and dynamic constraints [129] and Stein et al. extended the algorithm to respect constraints that avoided construction deadlocks during assembly [86]. Yun and Rus also provided an adaptive variant of the algorithm and demonstrated the construction of a loose square using the hardware [126].

[1]VICON motion capture systems: `https://www.vicon.com/`

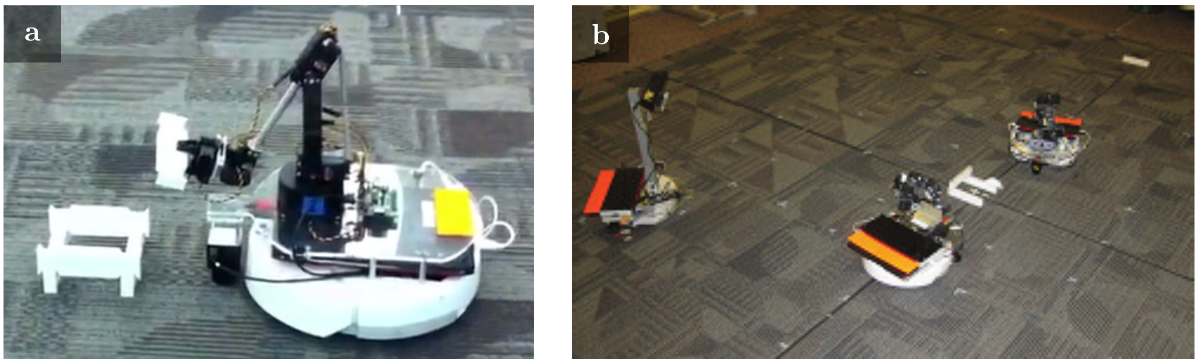


Figure 2.2: Distributed assembly of a Lincoln-log style structure at the SAS lab. (a) A robot attaches a block to the structure (reprinted from [28] with permission from M. Ani Hsieh). (b) A robot monitors the construction using an RGB-D sensor (reprinted from Experimental Robotics, 3-Dimensional Tiling for Distributed Assembly by Robot Teams, Volume 88 of the series Springer Tracts in Advanced Robotics, 2013, pages 143–154, James Worcester, Rolf Lakaemper, and Mong-ying Ani Hsieh, with permission of Springer).

2.2.2 Distributed assembly by Worcester et al.

Similar to the work presented in [130], Hsieh and Rogoff presented a metric for the partitioning of an assembly task among multiple robots. Based on the geometry of a target structure, this metric enabled a partitioner to maximize parallelism by considering the distances between the robots and the construction sites [29].

Worcester et al. demonstrated this work using two iRobot Create mobile robots, which they equipped with Lynxmotion five degree-of-freedom arms and Hokuyo scanning laser rangefinders [122]. An overhead camera system was used for localization. In [119], Worcester and Hsieh applied ant colony optimization (ACO) as an alternative approach to finding the optimal partitioning for a construction task.

Worcester et al. also ran experiments with a heterogeneous group of robots (Figure 2.2). This group consisted of two assembly robots and a scanner robot, which was tasked with error detection. Worcester et al. equipped the scanner robot with an RGB-D sensor (an Xbox Kinect [131]), which the scanner robot used to compare the state of a structure with its internal model [121]. The scanner robot reported any errors to the assembly robots over a wireless network. The structure, which was held together by magnets, consisted of three-dimensional tiles of different shapes and sizes. Worcester et al. extended their work to include online load balancing by allowing the robots to exchange tasks over a wireless network [120].

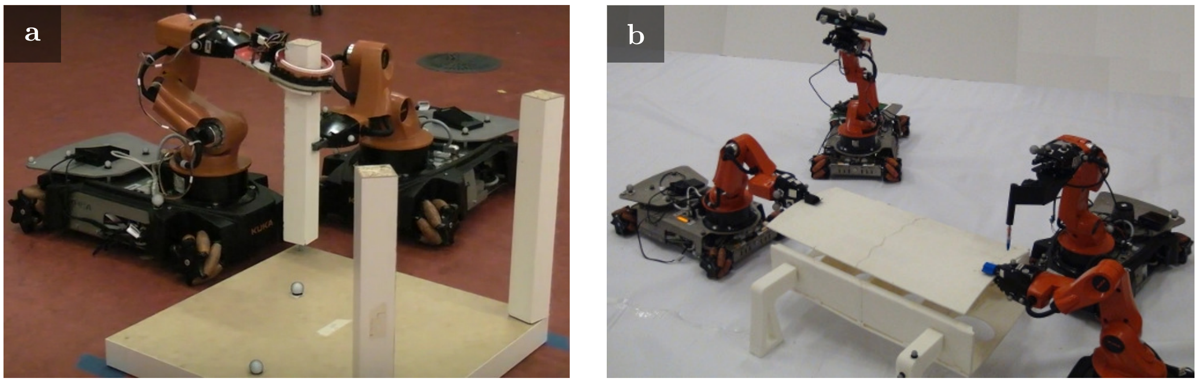


Figure 2.3: KUKA youBots assembling (a) an IKEA table (reprinted from [41] with permission, © 2013 IEEE) and (b) a model airplane wing (reprinted from Experimental Robotics, Towards Coordinated Precision Assembly with Robot Teams, Volume 109 of the series Springer Tracts in Advanced Robotics, 2016, pages 655–669, Mehmet Dogar, Ross A Knepper, Andrew Spielberg, Changhyun Choi, Henrik I. Christensen, and Daniela Rus, with permission of Springer).

2.2.3 Autonomous assembly by the Distributed Robotics Laboratory (DRL) at CSAIL MIT

The DRL extended their previous work, which we discussed in Section 2.2.1, to three-dimensional autonomous assembly. Experiments in this work used two to four KUKA youBot robots [5]. By default, these robots consisted of a base with an omnidirectional drive system and a five degree-of-freedom robotic arm. The robotic arm was equipped with a parallel plate gripper. The setup of the environment included a VICON motion capture system and a wireless network, which facilitated localization and communication respectively.

This work made several contributions to the research on multi-robot construction. For example, the localization of parts [106], automatic deduction of assembly steps [41], hierarchical perception of an assembly task [18, 19], multi-robot grasping and manipulation [20], and visual verification of an assembly task [13].

In [41], Knepper et al. tasked the robots with assembling an IKEA table (Figure 2.3). They fitted a specialized end-effector to one of the KUKA youBots, which provided continuous rotational motion to screw the table legs in place. The system automatically deduced the assembly steps from a provided CAD model. Following the assembly, the robots cooperatively grasped the table and turned it upright.

While the VICON motion capture system could accurately locate the robots and assembly parts in an environment, it suffered from occlusions and was not accurate enough for *peg-in-hole* assembly tasks. To this end, Dogar et al. attached an RGB-D camera (an Xbox Kinect) to one of the robots to allow for object-based tracking and a Hokuyo laser scanner to another robot for functional-feature-based tracking. These

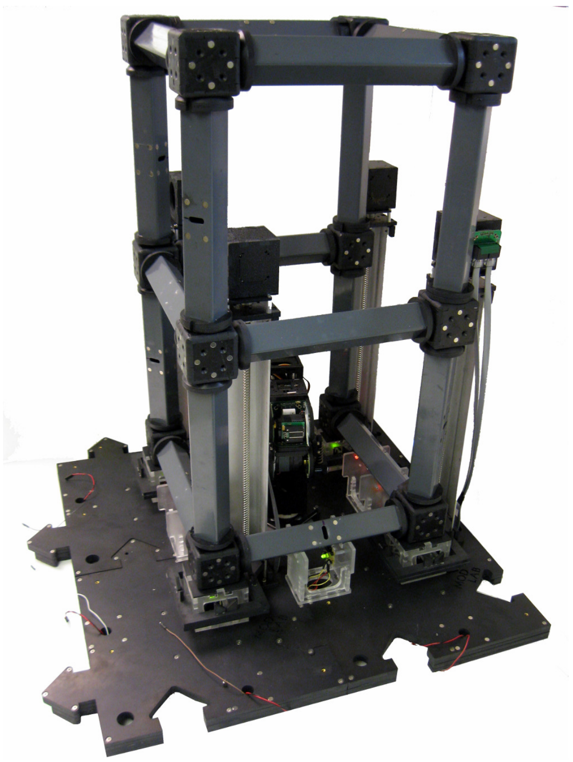


Figure 2.4: Four robotic tiles from the Factory Floor construction system assemble a cubic lattice-based structure (reprinted with permission from Kevin Galloway and Mark Yim).

inputs were arranged into a hierarchical perception system. The system used the highest resolution input available, falling back to the lower-resolution and broader-range inputs when tracking was lost. The robot with the laser scanner was tasked with placing the pegs into the assembly. In this work, Dogar et al. demonstrated the assembly of a model airplane wing (Figure 2.3) using four robots [18, 19].

In related work, Dogar et al. used three robots to assemble a chair [20]. The contribution of this paper was an algorithm for multi-robot grasping. The chair was held together using strong magnets. Using the same hardware, Choi and Rus demonstrated a technique using an RGB-D camera for visual verification of an assembly task by exploiting prior knowledge of the parts [13].

2.2.4 Factory Floor by Galloway et al.

Factory Floor differed from many multi-robot construction systems as it consisted of stationary robotic tiles (Figure 2.4). The robotic tiles were arranged to cover the footprint of the target structure and built the structure layer by layer, pushing the completed layers upwards. The system assembled truss-based cubic structures from nodes and members, which hypothetical collector robots delivered to the perimeter of a structure [23].

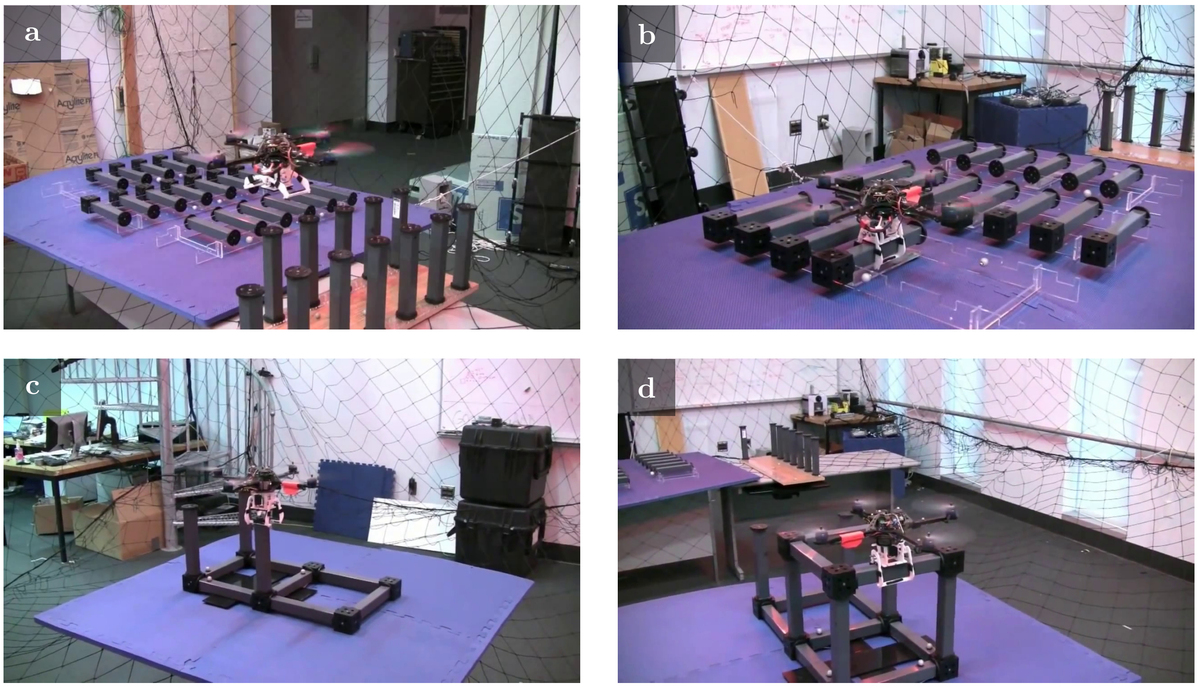


Figure 2.5: Assembly of cubic lattice-based structures by quadcopters. A quadcopter (a) hovers over the part storage area, (b) picks up a part, (c) transports it to the construction site, and (d) attaches it to a structure (reprinted from [47] with permission from Vijay Kumar).

Napp and Klavins described an approach to programming a multi-robot ACS, such as Factory Floor, using guarded command programs with rates [65–67]. Napp and Klavins showed that these programs could be composed and interpreted as Markov processes. Napp and Klavins noted that the stationary robotic tiles were an SPOF in the system as no other robotic tile could perform construction directly above the footprint of a failed tile [65].

2.2.5 Quadcopter construction by Lindsey et al.

Lindsey et al. investigated the application of quadcopters to multi-robot construction. They fitted Hummingbird quadcopters with a gripper, which picked up and placed construction materials (Figure 2.5). Inspired by the Factory Floor system, Lindsey et al. used truss members, which attached to each other using cubic nodes, as the construction material [45]. The members attached to one of the six faces of a node and were held in place using magnets. The nodes were pre-attached to some of the members, constituting a configuration referred to as a module. Prior to construction, the members and modules were placed in a designated area in the environment.

A VICON motion capture system tracked the quadcopters, providing positional data to a central computer running ROS and MATLAB. The central computer controlled the behavior of the quadcopters, which in turn regulated their altitude through onboard control loops. In this system, the quadcopters verified the placement of a member by applying a small moment and measuring the response. However, neither the quadcopters nor the VICON motion capture system monitored the state of the structure. Instead, the central computer inferred the state of the structure based on previous actions.

Lindsey et al. implemented an algorithm that decomposed cubic structures into assembly steps that were executed sequentially by the quadcopters [45]. In [46], Lindsey et al. removed the constraint that an internal cubic subassembly of a structure must be completed prior to starting another cubic subassembly using an alternative decomposition algorithm. Lindsey and Kumar also proposed an extension to the original algorithm that facilitated concurrent placement of members and modules into a structure [44]. They also generalized their algorithm to use different types of lattices and verified these extensions in theory and through numerical simulations.

2.2.6 Aerial construction by the Flying Machine Arena (FMA) at ETH Zurich

Augugliaro et al. demonstrated the construction of a six-meter-tall tower (Figure 2.6) using four quadcopters in 18 hours during a four-day-long live exhibition at the Fonds Régional d'Art Contemporain du Centre in Orléans, France [3, 116].

Similar to Lindsey et al. [45], Augugliaro et al. also used the Hummingbird quadcopter for construction, but instead of trusses, assembled the tower from 1500 polyurethane foam bricks. Two quadcopters performed the construction in parallel, allowing the other two quadcopters to recharge their batteries at the provided charging pads. A central computer with a blueprint tracked the positions of the quadcopters using a VICON motion capture system, dispatching commands to the quadcopters using pulse position modulation (PPM) and XBee radio links.

Figure 2.7 shows related work by Augugliaro et al., involving a similar setup that used ropes as a building material [1]. Mirjan et al. contended the value of this research on the basis that quadcopters could perform construction at heights that exceed the reach of cranes and could cooperate to combine tensile materials such as ropes, cables, and wire into three-dimensional structures [63]. In [2], Augugliaro et al. presented a framework for aerial knot tying, using the *munter hitch* as an example knot. This work used iterative learning to improve the accuracy and the efficiency of knot tying. Mirjan et al. combined this work into a demonstration that used three quadcopters to build a bridge [62].

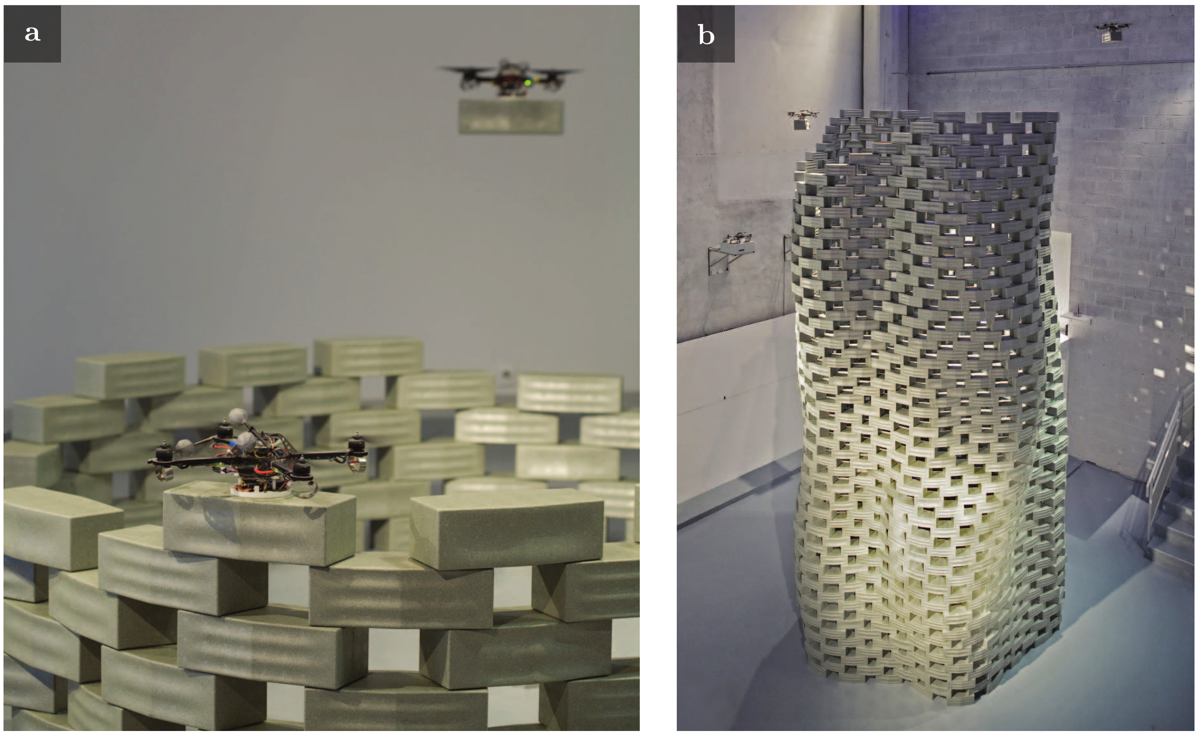


Figure 2.6: Quadcopters use (a) polyurethane foam bricks to construct (b) a tower at the Fonds Régional d'Art Contemporain du Centre in Orléans, France (reprinted from [3] with permission, © 2014, IEEE).

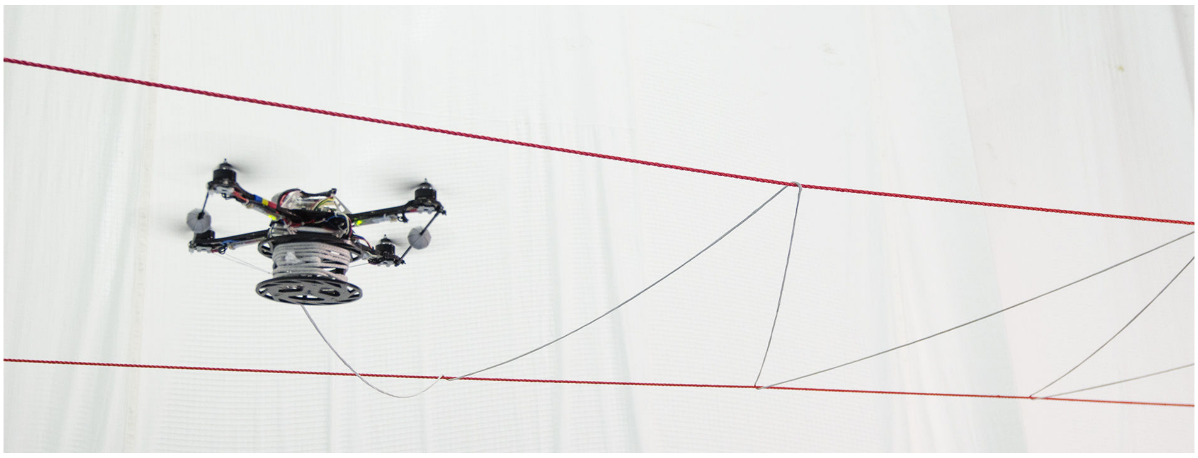


Figure 2.7: A quadcopter assembling a rope-based structure (reprinted from [1] with permission, © 2013, IEEE).

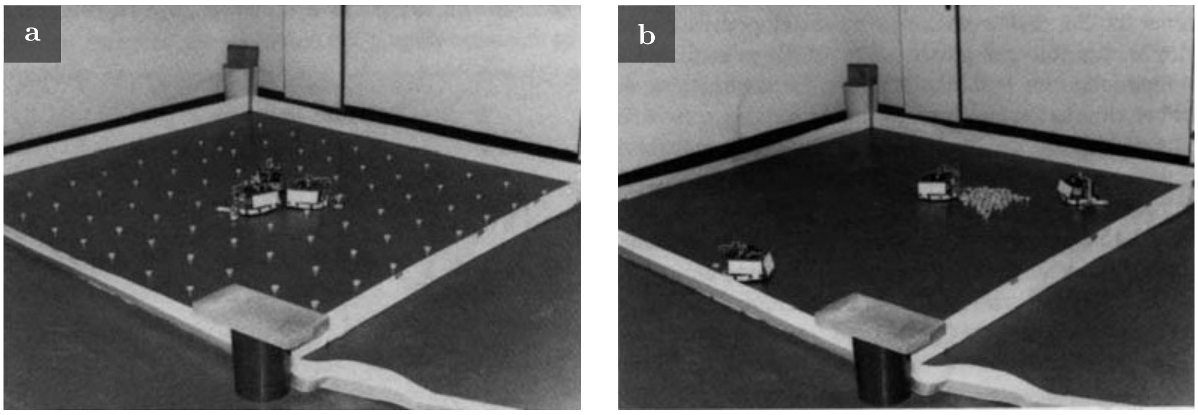


Figure 2.8: Autonomous clustering by Beckers et al. (a) The initial setup and (b) the completed task (reprinted from Prerational Intelligence, From Local Actions to Global Tasks: Stigmergy and Collective Robotics, Volume 26 of the series Studies in Cognitive Systems, 2000, pages 1008–1022, Ralph Beckers, Owen E. Holland, and Jean-Louis Deneubourg, © 1994, MIT, with permission of Springer).

2.3 Construction without centralized infrastructure

2.3.1 Clustering by Beckers et al.

Based on the patch sorting work by Deneubourg et al. in simulation [16], Beckers et al. implemented a clustering system using small autonomous robots, which they equipped with a C-shaped shovel to move pucks around an arena [4]. The autonomous robots consisted of two infrared (IR) proximity sensors and a small switch that triggered when three or more pucks accumulated in the shovel of a robot. In the experiments, the robots drove forwards, accumulating pucks until the small switch triggered, causing the robot to reverse and turn a random angle. After running experiments for an hour, Beckers et al. noted that a large cluster began to form. The experiments were terminated once the robots arranged the pucks into a single large cluster.

Beckers et al. ran experiments using one to five robots in a 250 cm × 250 cm square arena. They initially distributed the pucks for clustering evenly in the environment (Figure 2.8). In this setup, the most efficient number of robots in terms of the time taken to form a single large cluster was three.

The main contribution of this work is the first application of stigmergy to a multi-robot ACS. The robots communicated their previous work indirectly through the environment via the clusters. Although the robots could not directly sense the size of a cluster, the clustering converged as the robots tended to relocate pucks from small clusters to larger clusters. This relocation was due to the ratio between the probability of a

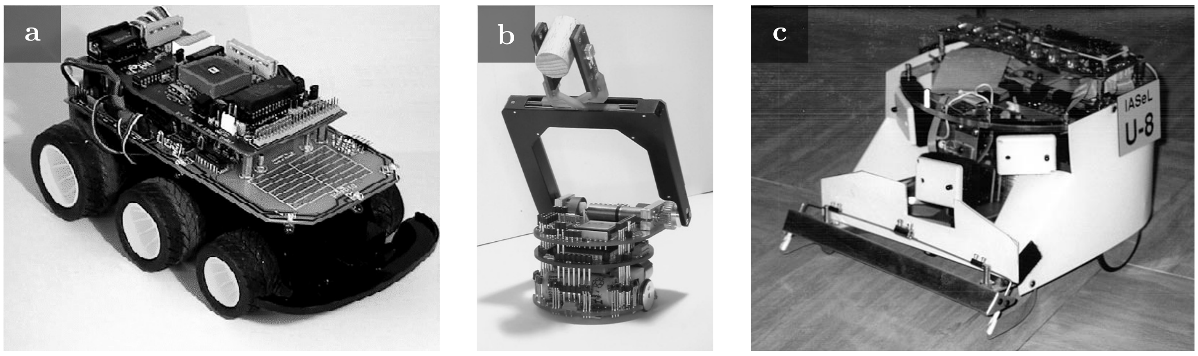


Figure 2.9: Examples of robots used for autonomous clustering and sorting. (a) The Didabot (reprinted from [49] with permission, © 1996, IEEE), (b) the Khepera robot (reprinted from Robotics and Autonomous Systems, 29.1, Alcherio Martinoli, Auke Jan Ijspeert, and Francesco Mondada, Understanding Collective Aggregation Mechanisms: From Probabilistic Modelling to Experiments with Real Robots, pages 51–63, © 1999, with permission from Elsevier), (c) the U-bot (reprinted from [27] with permission, © 1999, MIT).

tangential collision with a cluster, which removed pucks, and a non-tangential collision, which activated an obstacle avoidance behavior. This ratio was lower for larger clusters.

2.3.2 Clustering by Maris and te Boekhorst

Maris and te Boekhorst demonstrated clustering with autonomous Braitenberg-like robots [49]. The robots utilized five of the six proximity sensors positioned around their perimeter. The front proximity sensor was given a weighting of zero, causing the robots to push white polystyrene cubes around a 230 cm × 260 cm arena.

In contrast to the work by Beckers et al. [4], Maris and te Boekhorst randomized the positions of the cubes before starting experiments. Furthermore, they considered any cube that the robots pushed against the wall of the arena as *lost*. The main contribution of this work was the observation that multiple vehicles were required to form a single cluster. In a further experiment, Maris and te Boekhorst biased the vehicles to turn slightly to the right when no obstacle was detected. This modification resulted in the vehicles circling the arena and improved clustering performance.

2.3.3 Clustering by Martinoli et al.

Martinoli and Mondada performed clustering experiments using the Khepera robot [64], which they equipped with a gripper to pick up and place small cylinders [53]. The robots searched for cylinders that were placed in the 80 cm × 80 cm arena. If a robot found a cylinder while laden, it placed its cylinder near the found cylinder. If the robot was unladen, it picked up the found cylinder and continued its search.

Martinoli and Mondada implemented a discriminating behavior on the robots to improve the robot's ability to identify the cylinders [54]. They ran experiments in two square arenas (80 cm × 80 cm and 113 cm × 113 cm) with 1–10 robots and 10–40 cylinders. They compared their results to a probabilistic model that simulated the clustering process using a Markov chain. Martinoli and Mondada found a discrepancy between the hardware experiments and the probabilistic model. They hypothesized that this discrepancy occurred as a result of not modeling several aspects of the hardware. The main contribution of this work was the observation that the number of robots, the number of cylinders, the arena size, and the interaction geometry between a robot and a cylinder all impacted the performance of the clustering.

Martinoli et al. presented an improved probabilistic model that agreed with the results from the original hardware experiments [51]. They also performed the experiments in simulation using the Webots simulator [61] and demonstrated the generality of their probabilistic model by replicating the results from the clustering work by Beckers et al. [4]. Martinoli et al. summarized their findings in [52].

2.3.4 Clustering of square boxes by Kim et al.

Kim et al. demonstrated the clustering of 35 cm × 35 cm square boxes using iRobot Create mobile robots [40]. They performed their clustering experiments in a 450 cm × 450 cm square arena with chamfered corners to prevent the square boxes from becoming stuck. Kim et al. developed a boundary-aware controller for clustering that combined two operations: *twisting* and *digging*. They showed that this boundary-aware controller outperformed a reference controller by preventing the formation of clusters at the boundary of the arena. This work represented the first application of a behaviorally heterogeneous group of robots to a clustering task.

Song et al. showed how the division of labor, through the assignment of the twisting and digging operations, affected clustering performance [85]. Kim and Shell then improved the performance of this system by dynamically assigning the two operations [38]. They also investigated how different sequences of assignments of the two operations affected clustering performance by modeling the growth of the central cluster using a Markov chain. This Markov chain was calibrated with data from previous experiments.

Kim and Shell developed a model that explained how sets of clusters merged into a single central cluster without the formation of boundary clusters [39]. They revealed how the local densities of the robots and the physical packing of the clusters impacted cluster formation.

2.3.5 Blind bulldozing by Parker et al.

Franks et al. used the term *blind bulldozing* to describe the behavior of the ants *Leptothorax albipennis* as they cleared an area prior to nest construction [22]. Parker and Zhang implemented this behavior using toy bulldozers, which they converted into robots [72]. They showed that using two robot bulldozers was twice as fast as one robot and cleared an area with a significantly higher success rate. Parker and Zhang suggested that the interactions between the two robots allowed for an unobstructed workspace to be quickly obtained, enabling the robots to maneuver the environment with greater ease. Parker et al. developed a mathematical model of the blind bulldozing using a Markov chain [74]. The model showed agreement with the experiments conducted on the hardware. In [73], Parker and Zhang summarized their work and discussed the impact of adjusting the plowing force of the robots on the clearing of a circular area.

2.3.6 Collective sorting by the Intelligent Autonomous Systems (ISA) Lab at UWE Bristol

Melhuish et al. designed an autonomous system to sort red and yellow frisbees [56]. The robots in the system had no memory or means of localization and could only sense the color of a frisbee when it was in the gripper of the robot. The sorting was implemented using a *pull-back* behavior that was activated when a yellow frisbee was detected. This work was the first application of qualitative stigmergy to a multi-robot ACS. Holland et al. built upon these experiments, investigating the impact of arena size and the formation of boundary clusters. They also conducted further experiments, modifying the pull-back behavior, noting that the cluster of frisbees to which they applied the pull-back behavior was relatively sparse [27].

Melhuish et al. extended their hardware with an additional color sensor to enable patch sorting [59]. They verified this extension using a simulator, which they developed to evaluate potential sorting behaviors for the hardware. They used the simulator to evaluate patch sorting with 1–20 different types of objects. Melhuish et al. implemented the patch sorting on their hardware to sort three types of frisbees [60]. In later work, Melhuish et al. utilized computer vision to enhance patch sorting by enabling robots to sample the local density of frisbees [57].

Wilson et al. devised an approach to annular sorting, a sorting technique where different types of objects are clustered into concentric rings [117]. They implemented the sorting by setting the pull-back distance for each type of frisbee with respect to the contents of leaky integrators that tracked the average density of each type of frisbee. Wilson et al. compared this approach with two other sorting techniques: one based on an object's size and the other with a constant pull-back distance [118]. They found that the approach based on leaky integrators outperformed the other two approaches and improved it by tuning its parameters using a genetic algorithm. Scholes et al. compared these results to previous observational research on brood sorting by the ant *Leptothorax albipennis* [82]. They concluded that although the algorithms used by the robots and the ants produced the similar structures, they were fundamentally different.

In an earlier experiment, Melhuish et al. demonstrated the collective construction of a loose wall, using a bank of halogen lights and white tape to form a template in an environment [58]. The main contribution of this experiment was the use of a static template to regulate the construction behavior of a multi-robot ACS.

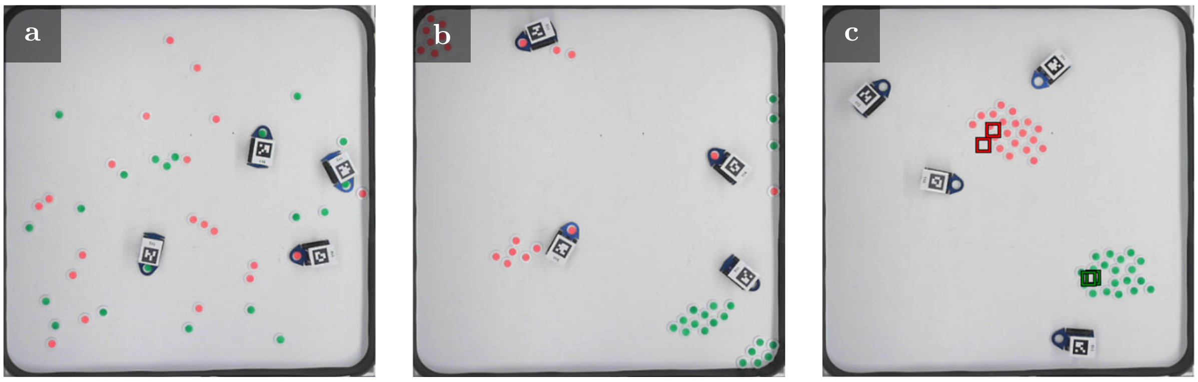


Figure 2.10: Patch sorting by Vardy et al. (a) The beginning of a patch sorting task, (b) patch sorting with the *color-seek* behavior, and (c) patch sorting using local consensus (reprinted from Swarm Intelligence, Cache Consensus: Rapid Object Sorting by a Robotic Swarm, 8.1, 2014, pages 61–87, Andrew Vardy, Gregory Vorobyev, and Wolfgang Banzhaf, with permission of Springer).

2.3.7 Patch sorting by the Bio-inspired Robotics (BOTS) Lab at Memorial University

Vardy implemented patch sorting in a simulation, which used computer vision to approximate the size and the contents of a cluster [101]. He based the simulation on the hardware of the SRV-1 mobile robot. Vardy compared two algorithms, called proximal and seeker. In contrast to the proximal algorithm, the seeker algorithm contained an additional behavior referred to as *color-seek*. In the case of a laden robot, the color-seek behavior searched for clusters of the same color as the carried puck. When the robot was unladen, the color-seek behavior searched for the most isolated puck. Vorobyev et al. extended the patch sorting algorithm from the work of Deneubourg et al. [16] by allowing the agents to localize themselves [104]. Their algorithm featured a collective decision-making component, which enabled the agents to reach a consensus about the type of object they were clustering and the location of its cluster. In [102], Vardy et al. ran experiments in simulation and with hardware to compare the clustering algorithm by Beckers et al. [4] with the two patch sorting algorithms from their previous work [101, 104]. The results from simulation showed that the patch sorting algorithms outperformed the clustering algorithm by Beckers et al. and the hardware experiments showed that the formation of a consensus improved patch sorting performance. This work represented the first application of collective decision making to a multi-robot patch sorting task.

In related work, Vorobyev et al. demonstrated the use of a neural network as a controller for multi-robot clustering [105]. Using computer vision to detect the number of pucks in front of a robot, they trained the neural network to select one of the predefined behaviors. Vorobyev et al. validated this control strategy in simulation.

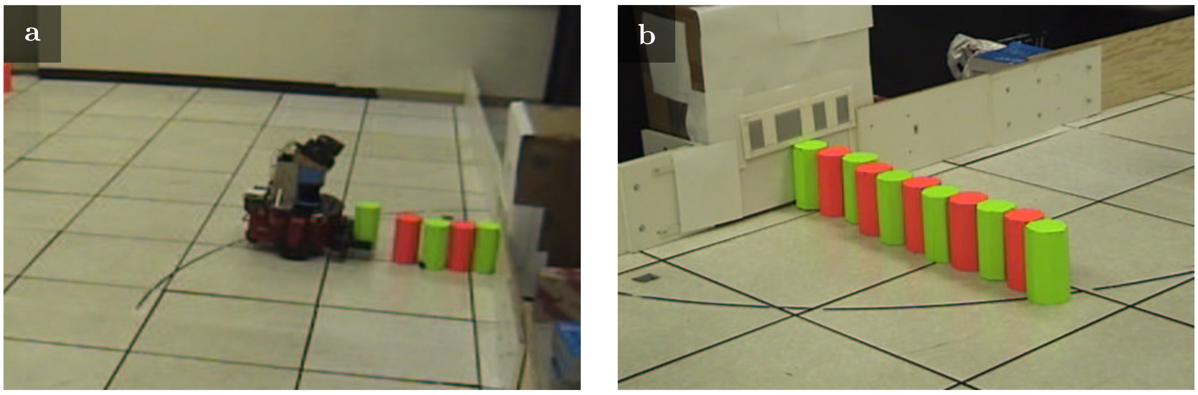


Figure 2.11: Construction of a wall by Wawerla et al. The robot (a) attaches Velcro blocks to (b) construct a wall (reprinted from [107] with permission, © 2002, IEEE).

2.3.8 Patch sorting by Verret et al.

Verret et al. designed a multi-robot patch sorting system, which they used to investigate the relationship between patch sorting, perceptual range, and local communication [103]. They emulated the field of view of a robot using an overhead camera system. They enabled local communication by allowing a robot R_1 to access the field of view of a robot R_2, given that robot R_2 was in the field of view of robot R_1. The main contribution of this work was the result that the introduction of local communication to a patch sorting task may improve performance, particularly when the robots have a limited perceptual range.

2.3.9 Wall construction by Wawerla et al.

Wawerla et al. demonstrated the construction of a wall using a laser template [107]. The robot searched an environment for two types of Velcro blocks, attaching them in alternating order to a partially-built wall (Figure 2.11). Wawerla et al. verified their controller for the construction using an ActivMedia Pioneer DX2 robot. They also performed multi-robot experiments using the same controller in simulation. This work demonstrated the first application of qualitative stigmergy to the construction of a wall. Wawerla et al. showed that the construction performance improved when they allowed the robots to communicate the last type of block attached to the wall. Furthermore, Wawerla et al. showed that the performance of the system improved with the number of robots until the attachment site became congested.

2.3.10 Wall construction by Stewart and Russell

Stewart and Russell used a dynamic template to construct a loose wall [87]. The template was created using a light source that was attached to an *organizer* robot. This robot

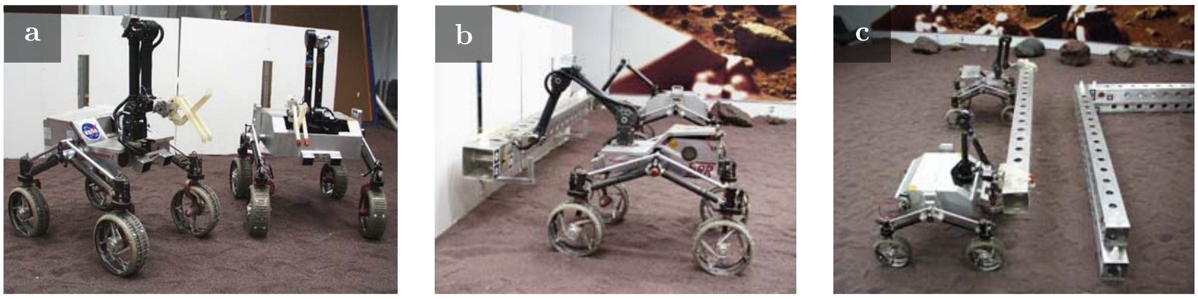


Figure 2.12: Multi-robot construction by the JPL. (a) The Robot Construction Crew (RCC) (b) locates parts from a storage area and (c) transports them to a construction site to assemble a photovoltaic tent (reprinted from [91] with permission, © 2005, IEEE).

was assigned the task of moving the light source in a straight line at regular intervals. *Worker* robots searched the environment for and picked up unused blocks, placing them underneath the moving light source. This behavior resulted in the construction of a loose wall.

In [88], Stewart and Russell enhanced their multi-robot wall construction system using a distributed feedback mechanism, which functioned as follows. The worker robots placed blocks underneath the light source created by the organizer robot. As construction progressed, this construction site became congested, preventing the worker robots from placing more blocks. Upon being unable to place a block, a worker robot incremented its internal frustration counter. When this counter reached a given value, the worker robot emitted a bright flash. The organizer robot counted these flashes and moved the light source after a given number of flashes were detected. Stewart and Russell demonstrated the robustness of the system by removing robots and introducing an obstacle into the environment. Stewart et al. also implemented the system in simulation and modeled the deposition process with a Markov chain [89]. This work demonstrated the first application of dynamic templates to a multi-robot construction scenario.

2.3.11 Multi-robot construction by the Jet Propulsion Laboratory (JPL) at NASA

CAMPOUT was a hierarchical control architecture used by the JPL to enable a group of robots to break a high-level task down into individual behaviors, which could be performed by a single robot. Huntsberger et al. described the CAMPOUT architecture and demonstrated its capabilities using a collective transportation task [31, 32]. Stroupe et al. extended this work by implementing a collective pick and place task [90, 91]. The pick and place task proceeded as follows. On initialization, the robots located a beam in a storage compartment using a computer-vision algorithm. After the two robots arrived at a storage compartment, they collectively transported a beam to a construction site prior

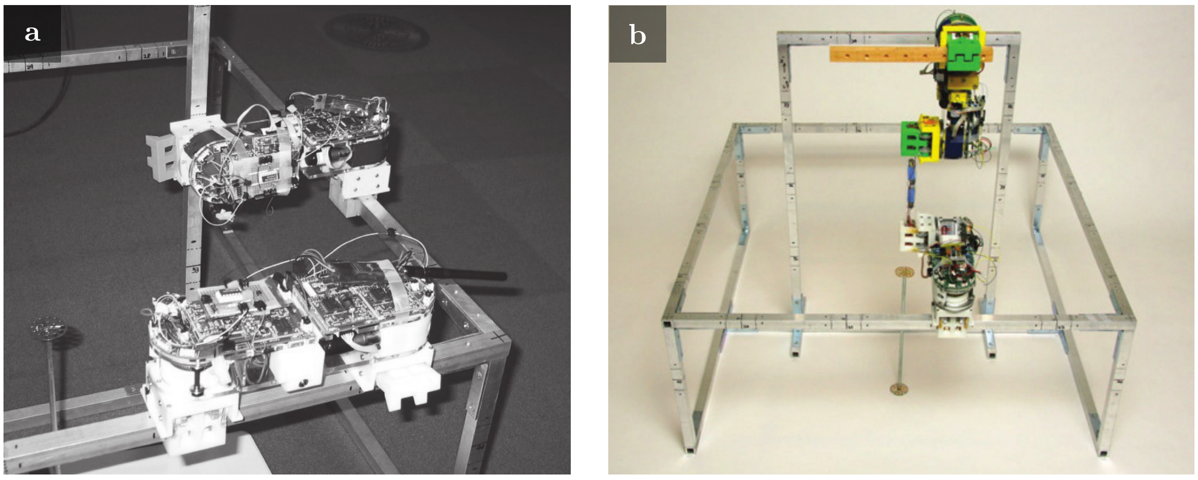


Figure 2.13: The Shady3D truss-climbing robots. (a) The hardware. (b) A modular manipulator consisting of two Shady3D modules connected through a passive bar (reprinted from [17] with permission, © 2007, IEEE).

to attaching the beam to an existing frame (Figure 2.12). Stroupe et al. later discussed an extension of this work to include placement of a flexible panel [92]. Huntsberger et al. discussed the results of this project in the context of autonomous construction in space, describing the subsystems required for a practical implementation [33]. The main contribution of this work was the hierarchical control architecture and the algorithms for collective manipulation and transportation.

2.3.12 Construction with truss-climbing robots by the Distributed Robotics Laboratory (DRL) at CSAIL MIT

Yoon et al. described their hardware implementation of Shady3D, a truss-climbing robot [123]. The robot moved through truss-based structures and was equipped with sensors that enabled error detection and correction during its maneuvers. Yoon et al. demonstrated an algorithm for finding the shortest path between two points in a structure.

In related work, Yun et al. demonstrated an online algorithm that allowed robots to navigate a truss-based structure while avoiding collisions with each other [128]. They emulated local sensing by allowing the robots to communicate their position in the structure once within a specified range. In [17], Detweiler et al. showed how the Shady3D robots could self-assemble into modules consisting of active and passive components to form a six degree-of-freedom modular manipulator.

In an extension to [17, 128], Yun and Rus designed algorithms that allowed the robots to self-assemble into dynamic structures, which they verified in a simulation based

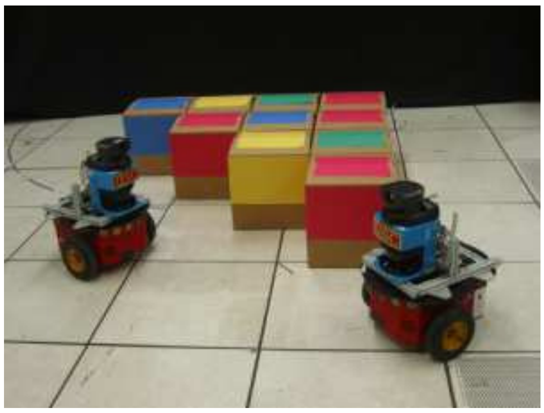


Figure 2.14: Construction of a planar structure by Jones and Matarić (reprinted from Experimental Robotics IX, Synthesis and Analysis of Non-Reactive Controllers for Multi-Robot Sequential Task Domains, Volume 21 of the series Springer Tracts in Advanced Robotics, 2006, pages 417–426, Jones and Matarić, with permission of Springer).

on the real hardware [124]. Yun and Rus proposed a redesign of their truss-climbing robot and a new building material, which consisted of struts and nodes [127]. In the proposal, the robots would reconfigure a structure by adding and removing the struts. Yun and Rus demonstrated using simulation how the proposed system would enable the implementation of reconfigurable structures. Yun and Rus provided a comprehensive review of their work in [125], summarizing previous experiments and demonstrating the locomotion of a six degree-of-freedom modular manipulator consisting of two Shady3D modules and a passive bar (Figure 2.13).

2.3.13 Multi-robot construction by Jones and Matarić

Jones and Matarić developed a tool for the automatic synthesis of communication-based controllers for multi-robot systems [34]. They demonstrated this tool by creating controllers for multi-robot construction. They performed experiments in simulation and on hardware using colored bricks and ActivMedia Pioneer 2DX robots. Since the robots had no manipulation capabilities, the controller requested the simulation engine or human operator to place a colored brick into the structure as required.

A color camera and laser range-finder allowed the robots to perform two types of observations of a structure. These observations always involved two colored blocks in either a *flush* or a *corner* configuration. In the context of this thesis, the main contribution of this work was a demonstration of how qualitative stigmergy, based on visual cues in an environment, may be used to coordinate construction.

In [35], Jones and Matarić presented a model for calculating the probability that a multi-robot system, consisting of robots with internal state without the ability to communicate, correctly executed a sequential Markovian task such as collective construction.

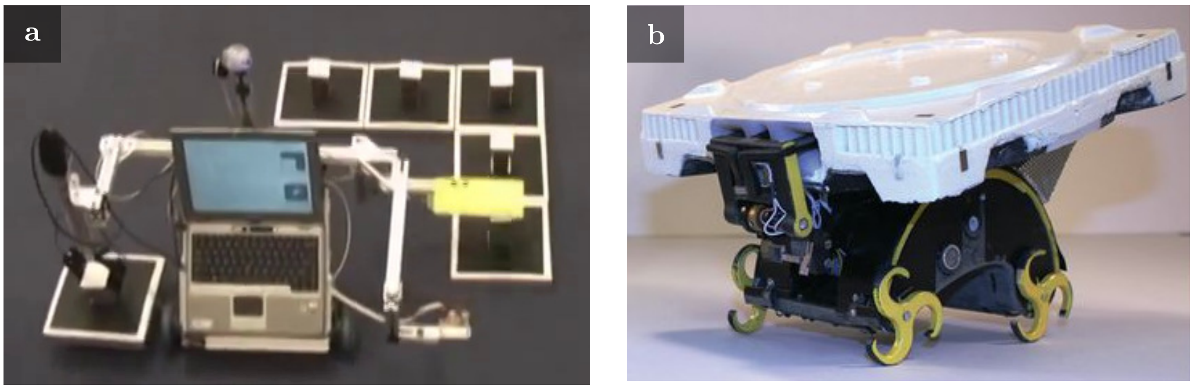


Figure 2.15: Robots for collective construction by Werfel et al. (a) A laptop-based robot navigates a two-dimensional structure labeling the semi-active tiles using RFID (reprinted from [111] with permission, © 2006, IEEE). (b) A robot from the TERMES system carrying a tile for three-dimensional construction (from Science, Designing Collective Behavior in a Termite-Inspired Robot Construction Team, 343.6172, 2014, pages 754–758, Justin Werfel, Kirstin Petersen, and Radhika Nagpal. Reprinted with permission from AAAS).

2.3.14 Collective construction by Werfel et al.

Werfel used a beacon to provide localization for a multi-robot construction system, which he simulated in a two-dimensional lattice [108]. This system was extended by Werfel et al. by using semi-active building materials, which enabled the robots to query the blocks in a structure. Each block maintained a complete map of the partially-built structure and informed an adjacent robot as to whether the placement of an adjacent block would result in a violation of either a geometrical or functional constraint [110].

Werfel and Nagpal introduced their concept of extended stigmergy [113]. The application of extended stigmergy gave the robots access to a complete map of a target structure and a shared coordinate system. Werfel et al. implemented a hardware prototype of a system that used extended stigmergy to coordinate two-dimensional lattice-based construction tasks [111]. The prototype reliably tracked its position around the perimeter of a structure and could successfully read and write the RFID tags in the semi-active tiles (Figure 2.15). Werfel et al. extended these algorithms to enable a group of simulated robots to build structures in a two-dimensional lattice, where the shape of a structure was adapted to obstacles in the simulated environment [109].

Werfel also discussed the challenges of extending collective construction to three-dimensional structures [112]. He concluded that the existence of a Hamiltonian cycle for a robot traversing all exposed faces of a given structure, at all stages of construction, was not guaranteed to exist. Werfel and Nagpal extended their simulation-based work to three-dimensional structures in [114]. Similar to their previous work in [110], the building material was semi-active and maintained a map of the partially-built structure. The building material communicated with the robots and signaled where to place further

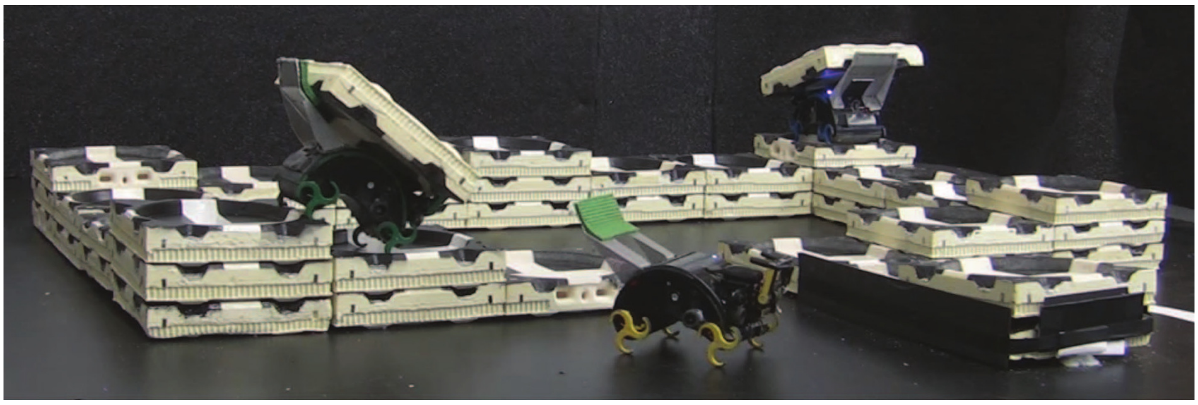


Figure 2.16: Three robots from the TERMES system building a castle from passive tiles (from Science, Designing Collective Behavior in a Termite-Inspired Robot Construction Team, 343.6172, 2014, pages 754–758, Justin Werfel, Kirstin Petersen, and Radhika Nagpal. Reprinted with permission from AAAS).

material. In this work, they introduced and compared three algorithms that enabled the robots to find construction sites.

Petersen et al. presented the hardware of TERMES, an autonomous construction system consisting of passive blocks and robots, which traversed a partially-built structure by climbing on top it [75]. This paper discussed the design decisions behind the TERMES system and demonstrated its capabilities through the construction of a staircase. Werfel et al. demonstrated three-dimensional construction using the TERMES system in [115]. They utilized an offline compiler that reduced a three-dimensional structure to a directed graph. The nodes in this graph constituted a height map of the structure and the directed edges between the nodes represented paths that a robot was allowed to traverse. A robot always entered the structure from a fixed location. This restriction allowed the robots to localize themselves on the directed graph and to share a common coordinate system. A robot used the directed graph and local sensing to determine if the placement of a tile at its current location was possible without causing a deadlock in the construction process. In this paper, they demonstrated the construction of a castle using hardware (Figure 2.16) and provided additional examples in simulation.

Werfel et al. demonstrated several techniques for how construction activity in multi-robot ACSs could be coordinated. The techniques involved applying extended stigmergy to the construction of two-dimensional and three-dimensional structures.

2.3.15 Collective construction by Sugawara and Doi

Based on the clustering work by Deneubourg et al. [16], Sugawara and Doi proposed an approach to collective construction. The approach used semi-active blocks equipped with a counter, a construction rule, and limited communication capabilities. The robots performed construction by moving randomly around an arena, picking up detected blocks, and placing them near blocks that signaled for construction.

In this work, the robots were unable to differentiate between unused blocks and blocks in a partially-built structure, resulting in the robots picking up both. Sugawara and Doi, however, referred to this characteristic as a *dynamic equilibrium* and maintained that it allowed the structure to be adaptive to an environment.

Sugawara and Doi implemented their system using a robot made from Lego Mindstorms NXT components and a semi-active block that communicated using IR LEDs [93]. Experiments using this hardware demonstrated the construction of a wall. Sugawara and Doi presented a refined implementation of their hardware, demonstrating the construction of a loose triangle and a loose wall [94].

2.4 Summary

2.4.1 Construction using centralized infrastructure

As summarized in Table 2.1, most multi-robot construction systems that use centralized infrastructure utilize a positioning system for localization. In these construction systems, a complete representation of the target structure is either provided to the individual robots or maintained on a central server, which can schedule and delegate construction actions to the individual robots.

Leveraging centralized infrastructure to coordinate a multi-robot ACS reduces the complexity of the control strategy. This reduction in complexity occurs as the centralized infrastructure is capable of monitoring the positions of the robots and the construction task. The trade off for this reduction in complexity is a reduction in the fault tolerance of the system, as any issues with the centralized infrastructure may cause a system-wide failure. The use of centralized infrastructure to support a multi-robot ACS is nonetheless a practical solution in structured environments such as factories, plants, and workshops. Prior to performing construction, however, this type of multi-robot ACS requires human technicians to fit an environment with a server, a wireless router, and a positioning system. Furthermore, these components should be connected to an uninterruptible power supply to ensure continuous and reliable operation. In unstructured environments that are either too remote or too dangerous for human technicians, however, this use of centralized infrastructure presents a problem. While it is possible in such an environment to set up the required infrastructure over time, the use of centralized infrastructure is unsuitable for rapid deployment.

2.4.2 Construction without centralized infrastructure

Avoiding centralized infrastructure in a multi-robot ACS increases the complexity of a control strategy. This increase in complexity occurs as the robots must now coordinate construction using only local information. This information may include perception of the surrounding environment using onboard sensors, direct communication with nearby robots, and indirect communication with other robots through the shared environment.

System	Building material	Robots	Locomotion	Infastructure	Structure Representation	Citations
Coordinated construction by the DRL	Semi-active Heterogeneous	4 Homogeneous	Differential drive	Positioning system & WLAN	Shared blueprint	[6, 86, 126, 129, 130]
Distributed assembly by Worcester et al.	Passive Heterogeneous	2-3 Heterogeneous	Differential drive	Positioning system & WLAN	Shared blueprint	[29, 119–122]
Autonomous assembly by the DRL	Passive Heterogeneous	2-4 Heterogeneous	Omnidirectional drive	Positioning system & WLAN	Shared blueprint	[13, 18–20, 41, 106]
Factory Floor by Galloway et al.	Passive Heterogeneous	4 Homogeneous	None	Not specified	Not specified	[23, 65–67]
Quadcopter construction by Lindsey et al.	Passive Heterogeneous	3 Homogeneous	Quadrotor	Positioning system & WLAN	Blueprint on a central server	[44–46]
Aerial construction by the FMA	Passive Homogeneous	4 Homogeneous	Quadrotor	Positioning system & WLAN	Blueprint on a central server	[1–3, 62, 63, 116]

Table 2.1: Comparison of multi-robot ACSs that use centralized infrastructure.

As discussed in Section 1.2, this indirect communication through the shared environment is referred to as stigmergy. Referring to Table 2.2 on Pages 36 to 38, we conclude from the surveyed literature that all of the multi-robot ACSs that avoid using centralized infrastructure and that are capable of construction, use some form of stigmergy.

The use of stigmergy in these systems was realized using a variety of sensors such as pressure sensitive switches, range finders, color sensors, light sensors, and image sensors. At this point, we introduce the concept of the bandwidth of a stigmergic signal and discuss how it influences the extent to which a multi-robot ACS can be coordinated.

An example of a low bandwidth stigmergic signal was in the clustering work of Beckers et al. [4]. In this work, a pressure sensitive switch provided a quantitative stigmergic signal that was binary, i.e. either there were enough pucks in front of a robot to trigger its switch or there were not. For a clustering task, a binary quantitative stigmergic signal is generally not sufficient as the robots are unable to discern the size of a cluster. As noted by Beckers et al., the convergence towards a single large cluster in their work was a function of the stigmergic signal and the interaction geometry between the environment, the robots, the pucks, and the clusters.

The use of a higher bandwidth stigmergic signal enables the coordination of more complex construction tasks. For example, Melhuish et al. implemented patch sorting using a color sensor [60] and in later work using an image sensor [57]. Furthermore, Jones and Matarić used computer vision to detect patterns of colored blocks, which enabled the construction of planar structures [34].

An alternative to using a higher bandwidth stigmergic signal is to supplement an existing stigmergic signal with additional information. For example, Verret et al. used local communication to extend the perceptual range of an individual robot during patch sorting [103]. Werfel and Nagpal introduced the concept of extended stigmergy, where a robot can localize itself on a map of a partially-built structure [113]. In contrast to increasing the bandwidth of a stigmergic signal, these two uses of additional information decrease the robustness of a multi-robot ACS. For example and in the case of local communication, individual robots must rely on each other to abstract their perception of a shared environment and to communicate it reliably to other robots. Furthermore, the use of extended stigmergy as proposed by Werfel and Nagpal relies on accurate odometry.

System	Building material	Robots	Coordination	Structure types	Citations
Clustering by Beckers et al.	Passive Homogeneous	1–5 Homogeneous	Quantitative stigmergy	Clusters	[4]
Clustering by Maris and te Boekhorst	Passive Homogeneous	1–5 Homogeneous	None	Clusters	[49]
Clustering by Martinoli et al.	Passive Homogeneous	1–10 Homogeneous	Quantitative stigmergy	Clusters	[50–54]
Clustering of square boxes by Kim et al.	Passive Homogeneous	5 Heterogeneous	Quantitative stigmergy	Clusters	[38–40, 85]
Blind bulldozing by Parker et al.	Passive Homogeneous	1, 2, 4 Homogeneous	Quantitative stigmergy	Cleared patches	[72–74]

Table 2.2: Comparison of multi-robot ACSs that do not use centralized infrastructure.

System	**Building material**	**Robots**	**Coordination**	**Structure types**	**Citations**
Collective sorting by the ISA Lab	Passive Heterogeneous	1–13 Homogeneous	Quantitative and qualitative stigmergy	Sorted patches, annular structures	[27, 56–60, 82, 117, 118]
Patch sorting by the BOTS Lab	Passive Heterogeneous	4 Homogeneous	Local communication, quantitative and qualitative stigmergy	Sorted patches	[101, 102, 104, 105]
Patch sorting by Verret et al.	Passive Heterogeneous	1, 2, 4 Homogeneous	Local communication, qualitative stigmergy	Sorted patches	[103]
Wall construction by Wawerla et al.	Passive Heterogeneous	1, 2, 4, 6, 8 Homogeneous	Static templates, local communication, qualitative stigmergy	Walls	[107]
Wall construction by Stewart and Russell	Passive Homogeneous	5 Heterogeneous	Dynamic templates, local communication, quantitative stigmergy	Loose walls	[87–89]

Table 2.2: Comparison of multi-robot ACSs that do not use centralized infrastructure (continued from Page 36).

System	**Building material**	**Robots**	**Coordination**	**Structure types**	**Citations**
Multi-robot construction by the JPL	Passive Heterogeneous	2 Heterogeneous	Parts located via computer vision	Single pick-and-place operations only	[31–33, 90–92]
Construction with truss-climbing robots by the DRL	Passive and active Heterogeneous	2 Homogeneous	Blueprint of truss structure	Single pick-and-place operations only	[17, 123–125, 127, 128]
Multi-robot construction by Jones and Matarić	Passive Heterogeneous	1–3 Homogeneous	Local communication, qualitative stigmergy	Planar structures	[34, 35]
Collective construction by Werfel et al.	Semi-active, passive Homogeneous	1–3 Homogeneous	Blueprint, extended stigmergy	2D and 3D tiled structures	[75, 108–115]
Collective construction by Sugawara and Doi	Semi-active Homogeneous	1 Homogeneous	Qualitative stigmergy	Loose planar shapes	[93–95]

Table 2.2: Comparison of multi-robot ACSs that do not use centralized infrastructure (continued from Page 36).

2.4.3 State of the art

To enable sophisticated construction without centralized infrastructure requires either a high bandwidth stigmergic signal or supplementing a stigmergic signal with additional information. At the time of writing and to the best of our knowledge, there are three multi-robot ACSs that are capable of building structures beyond loose walls, sorted patches, and clusters:

- the work from the case study by Jones and Matarić, which demonstrated the construction of planar structures using colored blocks and robots with computer vision [34],
- the TERMES system by Werfel et al., which demonstrated the construction of three-dimensional structures using passive tiles [115], and
- the work by Sugawara and Doi, which extends clustering work of Deneubourg et al. to enable the construction of loose planar structures [95].

In this thesis, we present a multi-robot ACS that uses a decentralized control strategy to coordinate construction. This decentralized control strategy is based on the work of Theraulaz and Bonabeau [98].

In terms of the existing literature on multi-robot ACSs, the most similar control strategy to Theraulaz and Bonabeau's was used in a case study by Jones and Matarić [34]. In these control strategies, the perception of the environment was used to recognize configurations of blocks in the partially-built structure, which were used to trigger construction actions. Furthermore, coordinated construction occurred in both systems as a result of a feedback loop where construction actions modified a partially-built structure, and the modifications to a partially-built structure triggered construction actions.

The work by Jones and Matarić demonstrated in part that it is possible to implement a variant of Theraulaz and Bonabeau's control strategy on a multi-robot ACS. There are, however, several issues that were not addressed by Jones and Matarić. For example, the robots in this work relied on a human operator to perform manipulation on their behalf. While this simplification of a given construction task was sufficient for the contributions of Jones and Matarić's paper, it sidesteps several aspects of multi-robot construction such as locating building materials, transporting them to a construction site, and avoiding collisions with other robots while attaching them to a partially-built structure. Furthermore, the construction tasks in this work were limited to planar structures and the challenges of performing three-dimensional construction were not addressed.

We aim to address these issues by adapting Theraulaz and Bonabeau's control strategy and implementing it on the multi-robot ACS that we present in this thesis. This multi-robot ACS is capable of locating unused building materials, transporting them to a construction site, and attaching them with respect to the current configuration of a partially-built structure. Our multi-robot ACS is also fully autonomous and capable of three-dimensional construction.

CHAPTER 3
System Architecture

3.1 Design of a multi-robot ACS

A multi-robot ACS requires a minimum of two components: an autonomous robot and a building material. The selection of these two components is not independent as a robot must be able to locate and manipulate the building material. This requirement influences the design of both components.

The proposed multi-robot ACS is a research tool for investigating autonomous construction and is designed to be operated on a smooth surface (e.g. on top of a workbench) in a laboratory. As the focus of this thesis is the implementation of a decentralized control strategy on a multi-robot ACS, we design the hardware of our multi-robot ACS so that only simple sensing and actuation is required. This approach reduces the required signal processing onboard the robots and increases the reliability of the system.

3.1.1 Building materials

The building materials for a multi-robot ACS can be rigid or non-rigid. As an example, rigid materials may include blocks, trusses, nodes, and connectors. In contrast, examples of non-rigid materials may include deformable pockets with a granular material or flexible ropes. The selection of a building material depends on the environment, the target structures, and the associated cost and availability of the building material. For example, sand bags are a non-rigid building material and are often used to build temporary structures that contain flooding rivers since their non-rigid geometry adapts to the shape of the surrounding environment [83]. As a further example, ropes can be arranged using loops and knots to form sophisticated structures such as bridges [62].

In contrast with rigid materials, however, the autonomous manipulation of non-rigid materials is still a new area of research [68, 69]. Furthermore, the cost and availability of manufacturing technologies for rigid materials (e.g. rapid prototyping) are less expensive and more ubiquitous than for non-rigid materials.

It is also possible to use semi-active building materials that are capable of either computation or communication. While these capabilities may increase the cost of a building material, they also potentially allow for research into the autonomous construction and maintenance of smart structures.

3.1.2 Robots

Research into multi-robot ACSs has demonstrated the use of aerial-based, ground-based, and truss-based robots for construction. Aerial-based robots, for instance, are capable of moving around an environment in three-dimensions. This capability enables aerial-based robots to inspect a partially-built structure from a perspective that is not obtainable with ground-based or truss-based robots. Furthermore, due to their versatile mobility, aerial robots are potentially more capable of building three-dimensional structures, such as tall buildings.

Ground-based robots, however, have some advantages over aerial robots. For example, the control loops for ground-based robots require lower sample rates than aerial-based robots. These lower sample rates result in less computation and lower power consumption. Furthermore, a stationary ground-based robot requires less power than a hovering aerial-based robot and can lift heavier loads.

Similar to aerial-based robots, truss-based robots benefit from being able to move through a structure in three-dimensions. In contrast to aerial-based robots, however, truss-based robots use less power as they are always attached to a structure. In comparison to aerial-based and ground-based robots, however, truss-based robots require a fixture prior to starting a new structure and external infrastructure to supply them with building material.

3.2 Adaptation of Theraulaz and Bonabeau's decentralized control strategy

In the work by Theraulaz and Bonabeau, simulated agents searched a three-dimensional lattice for predefined patterns consisting of two types of bricks [98]. When an agent detected one of these patterns in the lattice, it placed a brick at its current location. This behavior resulted in a feedback loop where the placement of a brick modified the lattice, and modifications to the lattice caused different patterns to be detected, triggering further brick placement.

By carefully selecting the predefined patterns that triggered block placement, Theraulaz and Bonabeau implemented a control strategy that used this feedback loop to coordinate construction. This control strategy is fully decentralized and compatible with the design constraints for swarm robotic systems as described by Brambilla et al. [10]. These design constraints are important as it is our long-term goal to implement a swarm robotics construction system.

However, before Theraulaz and Bonabeau's control strategy can be implemented on a multi-robot ACS several adaptations are required. In this thesis, we aim to make these adaptations while remaining compliant with the design constraints for swarm robotic systems.

3.2.1 Different types of blocks

In related work to [98], Bonabeau et al. defined a fitness function that measured the *structure* produced by a given construction algorithm [7]. In this work, Bonabeau et al. noted that no interesting patterns could be generated using a single type of brick. In a lattice-based simulation, the use of multiple types of bricks can be implemented programmatically. However, in a physical system such as a multi-robot ACS, the use of multiple types of building material requires modifications to the control strategy.

In Section 2.3.13, we discussed the work of Jones and Matarić, which used a similar control strategy to enable coordinated construction [34]. In this work, however, the challenges of performing construction with multiple types of building materials were mostly circumvented since a human operator attached the building material to a partially-built structure on behalf of the robots.

If our multi-robot ACS is to be fully autonomous and capable of performing coordinated construction using a variant of Theraulaz and Bonabeau's control strategy, we must find a way for the robots to attach different types of building material to a structure. To this end, we consider three solutions.

The first solution involves each robot carrying multiple types of building material simultaneously. This solution is, however, somewhat inefficient as the robot must carry the weight of each type of material and the weight of the complex mechanical hardware that must pick the different types of building material up, store them during transport, and attach them to a structure.

The second solution involves a robot making an observation of a partially-built structure prior to locating the required building material. This solution is, however, algorithmically more complex and may suffer from congestion issues at a construction site. For example, following an observation, a robot may retrieve the required building material to perform a given construction task only to find that another robot has already completed the construction task. The laden robot then has three options: (i) wait at the construction site until the acquired building material is required again, (ii) find another construction site that requires the acquired building material, or (iii) abandon the acquired building material. This issue, however, may be partially mitigated by allowing local communication between the robots as demonstrated by Wawerla et al. [107].

The third solution is inspired by the construction behavior of termites. Bruinsma showed that termites coordinate the construction of a royal chamber in part by marking soil pellets with different types of pheremones [11]. It is possible to implement this mechanism of marking a homogeneous building material in a multi-robot ACS. For example, a robot may use a specialized manipulator to attach QR codes to the building materials. Alternatively, a robot may write to RFID tags embedded in the building material as demonstrated by Werfel et al. [111]. The proposed use of RFID, however, may be a complex solution to implement since an observation of a partially-built structure requires a robot to read and localize multiple RFID tags accurately.

3.2.2 Discretization of an environment

A significant adaptation to Theraulaz and Bonabeau's control strategy is required as a result of the inherent discretization in a lattice-based simulation. As noted in the work by Holland and Melhuish, this discretization influences an agent's ability to sense, to act, and to manipulate objects in an environment [27]. In the following paragraphs, we discuss our options for adapting each of these abilities with respect to what can be physically implemented in a multi-robot ACS.

Sensing In order for a robot to inspect a partially-built structure, a form of computer vision is required. This computer vision may be based on a two-dimensional image using an image sensor or a three-dimensional point cloud using a light detection and ranging (LIDAR) sensor or a red-green-blue and depth (RGB-D) sensor.

While LIDAR and RGB-D sensors provide valuable spatial information, they are often implemented as active sensors that emit a reference signal into an environment and measure its response. This emission of a reference signal may be problematic in a multi-robot system due to the interference of reference signals between two or more robots. There is also a passive solution to three-dimensional computer vision, which is realized using two image sensors in a stereo vision configuration. Combining these two images into a three-dimensional point cloud, however, is a computationally intense operation.

Irrespective of what type of computer vision is used to implement sensing in our decentralized control strategy, there is also a secondary issue of perspective. The simulated agents in the work of Theraulaz and Bonabeau could sense the state of all the cells within a unit Chebyshev distance of their position. While this can be partially accomplished on a robot using curved mirrors to implement an omnidirectional camera system [9], this solution may require additional processing to remove distortions from the captured scene prior to feature extraction. Furthermore and in contrast with the simulated agents, it is likely that the selected computer vision system will nonetheless have blind spots that impact both the perception of a partially-built structure and the avoidance of obstacles in an environment.

Actuation The agents in Theraulaz and Bonabeau's control strategy are capable of moving a unit Chebyshev distance in any direction on each step of the simulation. Although three-dimensional motion is possible with aerial-based robots, neither aerial-based robots or ground-based robots can instantly move to a target location. Rather, these robots must accelerate, maintain a given velocity, and decelerate as they arrive at a target location. Furthermore, during this motion sequence, a robot must monitor its surroundings for both static obstacles (e.g. walls or partially-built structures) and dynamic obstacles (e.g. other robots).

Manipulation When a simulated agent in Theraulaz and Bonabeau's control strategy detects a predefined pattern in the simulated lattice, it places a brick at its current location. The simulated agents in this work have the capability to store an infinite amount of bricks. This is obviously not possible for a physical robot. To this end, a robot must first locate unused building material that is either scattered throughout an environment or provided by an unused building material cache. Once a robot has located and picked up a piece of building material, it must locate a partially-built structure and attach the building material to it. Locating both the unused building material and a partially-built structure can be implemented using a similar approach to that in Theraulaz and Bonabeau's control strategy, where a simulated agent performs a random walk in an environment.

A significant adaptation to this control strategy is required as the observation and attachment operations are generally not performed in the same location. That is, a robot must first make an observation of a partially-built structure from a distance, determine based on its observation how the building material may be attached to the structure, approach the structure, and finally use its manipulator to attach the building material.

3.3 Architecture of our multi-robot ACS

In this section, we discuss the adaptations that we have made to Theraulaz and Bonabeau's decentralized control strategy and the architecture of our multi-robot ACS, which we implement using two components: a building material and an autonomous robot.

With respect to the requirement of using different types of building material, we implement a markable semi-active active building material, which we refer to as *stigmergic blocks*. The stigmergic blocks can be marked by the robots using a near field communication (NFC) interface. These markings are displayed using light-emitting diodes (LEDs) on each face of a block and are visible to nearby robots that are equipped with a color camera. We attach a localizable tag to each face of a stigmergic block to enable the implementation of the required perception of a partially-built structure without three-dimensional computer vision. This decision reduces the required processing power and extends the battery life of a robot.

To assemble the blocks into structures, we develop and manufacture autonomous robots. These autonomous robots are capable of performing all necessary signal processing on board, including computer vision. This capability of the autonomous robots enables our multi-robot ACS to operate without centralized infrastructure.

3.3.1 Stigmergic blocks

The stigmergic blocks are a rigid, cubic building material, which is printed using selective laser sintering (SLS). The choice of a cubic geometry enables an autonomous robot to attach a stigmergic block to a partially-built structure irrespective of the block's orientation. Meanwhile, the rigidity of a stigmergic block simplifies its manipulation. The selection of SLS as a manufacturing technology reduces the number of off-the-shelf components as mechanical functionality can be directly printed into the parts. Furthermore,

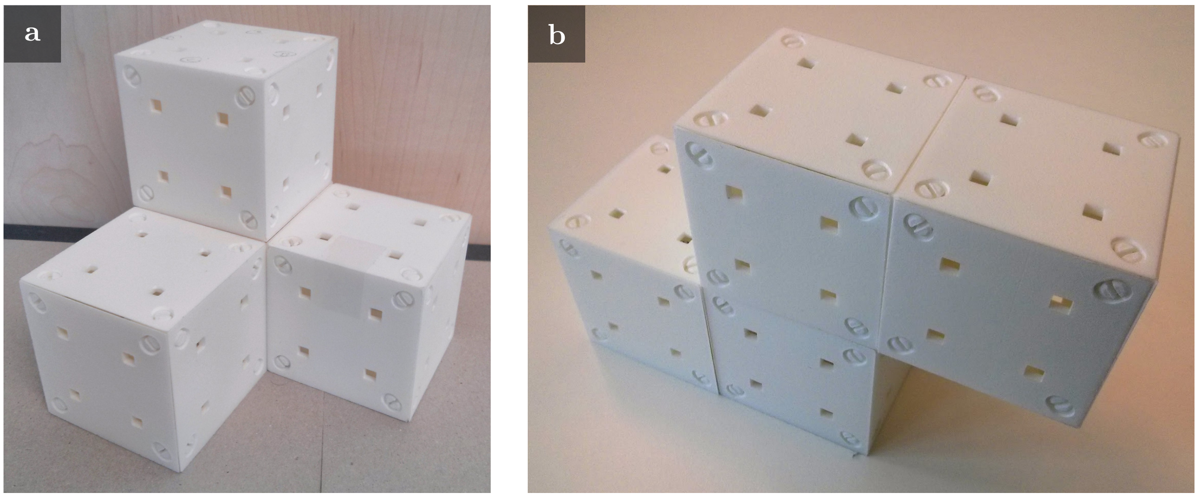


Figure 3.1: Structures made from an early prototype of the stigmergic blocks. The blocks are printed using selective laser sintering (SLS) and the structure is held together using freely-rotating spherical magnets that are located in the corners of a block.

since we use SLS for both prototyping and production, there should be no variations between the prototyped and production parts.

To simplify actuation, we place freely-rotating spherical magnets in the corners of a stigmergic block to allow it to align with the end-effector of an autonomous robot as it is picked up. These spherical magnets also strengthen the structures made from the blocks. Figure 3.1 shows an early prototype of the stigmergic blocks, which are arranged into two different structures.

The stigmergic blocks are designed to be semi-active and markable by a robot using an NFC interface. The markings are displayed using LEDs that are visible to nearby robots. To increase the bandwidth of the stigmergic signal from a block, we use four different colors for the LED markings. The structural arrangement of the stigmergic blocks and their LED markings form patterns in an environment that are detected by nearby robots. These patterns may be used to trigger construction actions.

3.3.2 Autonomous robots

For this work, we decide to use a homogeneous group of ground-based robots. Ground-based robots are easier to control than both aerial-based and truss-based robots and are therefore a more suitable configuration for our initial implementation of a decentralized control strategy on a multi-robot ACS. It would be, however, possible to include either aerial-based or truss-based robots in an extension to this work, potentially enabling construction using a heterogeneous group of robots.

Instead of creating a ground-based autonomous robot from scratch, we implement our autonomous robot as an extension to a mobile robotics platform called the BeBot [26]. At the time of writing, the current version of the BeBot does not have sufficient processing

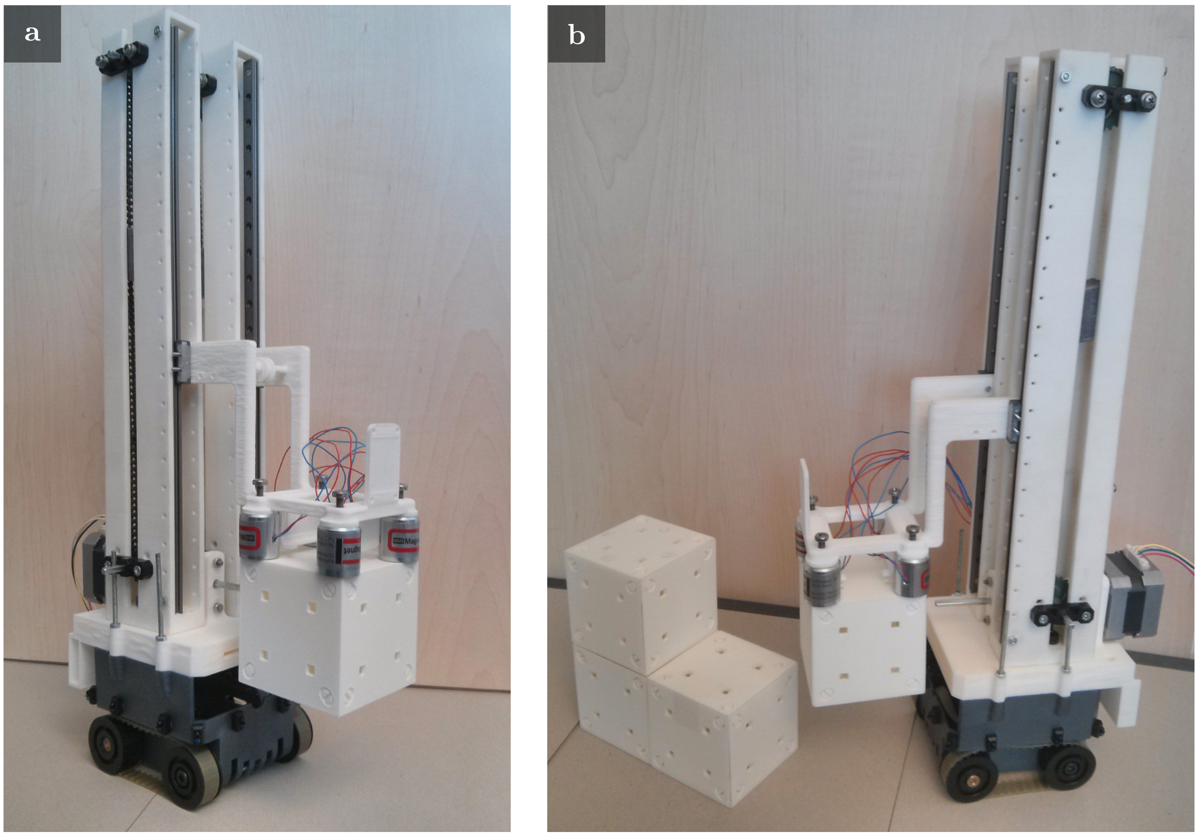


Figure 3.2: An early prototype of an autonomous robot constructed from printed and off-the-shelf components. Four semi-permanent electromagnets in the end-effector of an autonomous robot facillitate a magnetic coupling between its end-effector and a stigmergic block.

power with respect to the computer vision requirements of our system. To this end, we upgrade the mobile robotics platform by completely redesigning its two circuit boards.

To pick up and to attach the stigmergic blocks to a partially-built structure, we design a manipulator that attaches to the top of the mobile robotics platform. The manipulator controls the vertical position of an end-effector. This end-effector picks up a stigmergic block by attaching to its top face. This attachment is facilitated through the coupling of the four spherical magnets at the top of a stigmergic block with four semi-permanent electromagnets in the end-effector. Figure 3.2 shows an early prototype of an autonomous robot demonstrating this mechanism.

The manipulator and its end-effector are assembled from off-the-shelf parts and components printed using stereolithography (SLA). As we had a Form 1+ printer on site, using SLA allowed us to modify, print and evaluate the mechanical components of the autonomous robot within a 24 hour period. Furthermore, since we use SLA for both prototyping and production, there should be no variations between the prototyped and production parts.

A forwards facing camera is attached to the end-effector of the manipulator to capture the scene in front of an autonomous robot. In contrast to an omnidirectional camera, a forwards facing camera provides a relatively undistorted view of the environment in front of a robot. We consider the view in front of our robot to be more important than the view in other directions since the picking up and placement of a stigmergic block always takes place in front of our robot. Sensory input from other directions is provided by twelve rangefinders, which are positioned around the perimeter of the mobile robotics platform.

3.3.3 The decentralized control strategy

To compensate for the discretization in Theraulaz and Bonabeau's control strategy, we implement an observation and brick placement routine as follows. Since our robots do not have the capacity to carry multiple blocks, an unladen robot starts by searching an environment for an unused stigmergic block. Upon detecting an unused stigmergic block, the robot approaches it and picks it up. The laden robot now searches its environments for a partially-built structure. Upon finding a partially-built structure, a robot inspects it to determine if it contains a predefined pattern. A predefined pattern consists of a structural arrangement of stigmergic blocks with specific LED markings and has an associated construction action. An associated construction action consists of an attachment site and an LED marking for an unused stigmergic block. If a robot finds a predefined pattern, it approaches the partially-built structure by tracking a block near the attachment site.

During its approach, the robot monitors its environment using a combination of computer vision and input from rangefinders in order to detect nearby robots. If another robot is detected, the attachment operation is aborted and the robot moves away from the partially-built structure before restarting its search for another structure. This behavior is designed to avoid collisions between robots and to avoid conflicts at a construction site. If the robot arrives at the attachment site without detecting another robot, it configures the LED markings on the unused stigmergic block using the NFC interface. The robot then attaches the marked stigmergic block to the partially-built structure before reversing away from the partially-built structure and searching the environment for another unused stigmergic block.

This use of computer vision to search an environment for predefined patterns consisting of a structural arrangement of stigmergic blocks with specific LED markings, enables an autonomous robot to participate in coordinated construction. This coordinated construction occurs between one or more robots due to a feedback loop where the attachment of an unused stigmergic block modifies a partially-built structure, and where modifications to a partially-built structure may trigger the attachment of other blocks by the robots.

3.3.4 Simulation

To assist with the development and debugging of the decentralized control strategy, we develop a number of extensions to the ARGoS multi-robot simulator [78]. In contrast to the lattice-based simulation work by Theraulaz and Bonabeau, these extensions enable a comprehensive and realistic simulation of our multi-robot ACS. For example, these extensions enable realistic computer vision and modeling of the three-dimensional dynamics and magnetism in our system.

In addition to supporting the development and debugging of the decentralized control strategy, we intend to use the simulator to investigate how the decentralized control strategy scales to construction tasks involving many stigmergic blocks and autonomous robots.

While simulation is an invaluable tool for supporting the development and debugging of our decentralized control strategy, it is not a substitute for the hardware. Rather, the hardware of our multi-robot ACS and its simulation complement each other as follows. The hardware validates the results from simulation, while the simulation enables us to predict whether a construction task or configuration of the multi-robot ACS will function as expected on the hardware.

3.4 Summary

In this section, we have discussed different approaches to the implementation of a multi-robot ACS and the required adaptations to Theraulaz and Bonabeau's control strategy. Based on this discussion, we proposed the design of a multi-robot ACS, which consists of stigmergic blocks and autonomous robots. In the next chapter, we detail the electronics, mechanical design, and software of these components.

CHAPTER 4

Implementation of the Hardware

4.1 Stigmergic blocks

We have designed the stigmergic blocks to simplify the actuation and sensing requirements of the autonomous robots so that we can focus our work on developing decentralized control strategies for multi-robot construction.

In brief, a stigmergic block is an advanced cubic building material capable of computation, data storage, and communication. There are currently two types of communication used in our multi-robot ACS. The first type uses a near field communication (NFC) interface to enable robot-to-block communication. The second type utilizes the LEDs on the faces of a block to enable block-to-robot communication, which is facilitated by computer vision. The electronics of a block is implemented using a central circuit board and six face circuit boards. The exterior of a block has a side length of 55 millimeters and is printed using selective laser sintering.

The simplification of the actuation and sensing requirements is partially achieved by attaching localizable tags to the stigmergic blocks [71], which enables an autonomous robot to accurately locate a block in an environment. Furthermore, we have added a freely-rotating, spherical magnet into each corner of a block to enable self-alignment and to reduce cumulative misalignment during construction. These spherical magnets also increase the structural integrity of a structure.

4.1.1 Electronics

Figure 4.1 shows the microcontroller, which runs a block's software. We have mounted the microcontroller on the central circuit board to manage the power and the routing of data between various interfaces. The microcontroller is programmed using the Optiboot[1] bootloader, which enables reprogramming via the microcontroller's serial port. We have connected the serial port of the microcontroller to a USB-to-serial converter IC with USB battery charger detection. This configuration allows for recharging, reprogramming, and debugging a stigmergic block over a single USB connection. The USB-to-serial converter is configured over USB to provide control signals for a power management IC.

[1]Optiboot: `https://github.com/optiboot/optiboot`

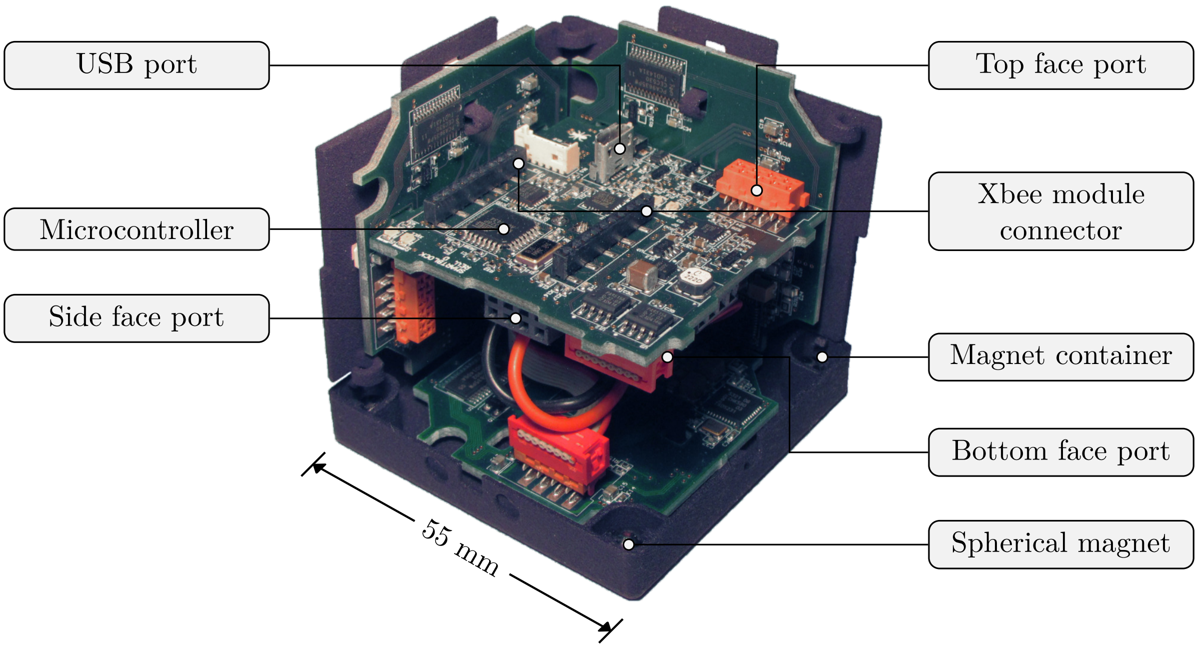


Figure 4.1: Internal view of a stigmergic block.

These signals inform the power management IC how much current may safely be drawn from the USB connection. The power management IC allocates this power to the system and to the attached lithium-ion battery for recharging. The system power is connected to two switching regulators, which can be switched on and off using the push button located near the USB port. The first regulator provides 3.3V for the digital electronics, while the second regulator provides 5V for the LEDs on the face circuit boards.

We provide a standard socket for a wireless module[2] on the central circuit board. The purpose of this wireless module is to enable remote debugging and monitoring of a stigmergic block. This wireless module communicates with the microcontroller via a serial port. As the microcontroller only has one serial port, which is used for reprogramming via USB, a second serial port is emulated by using a 16-bit timer with the AltSoftwareSerial[3] library.

The central circuit board provides six connectors for each of the face circuit boards. These connectors provide each face circuit board with power, an interrupt line, and an I^2C bus. As the face circuit boards are identical, the I^2C bus is segmented so that there are no address conflicts. Each face circuit board contains an NFC transceiver and an LED driver. The LED driver is used to set the brightness of the red, blue, and green channels of four multi-color LEDs on a face circuit board. An autonomous robot can sense the color of these LEDs from a distance while inspecting a structure. The NFC transceiver allows messages to be sent and received wirelessly to nearby robots or blocks.

[2]Xbee Wireless Modules: `https://www.digi.com/lp/xbee`

[3]AltSoftwareSerial: `https://github.com/paulstoffregen/altsoftserial`

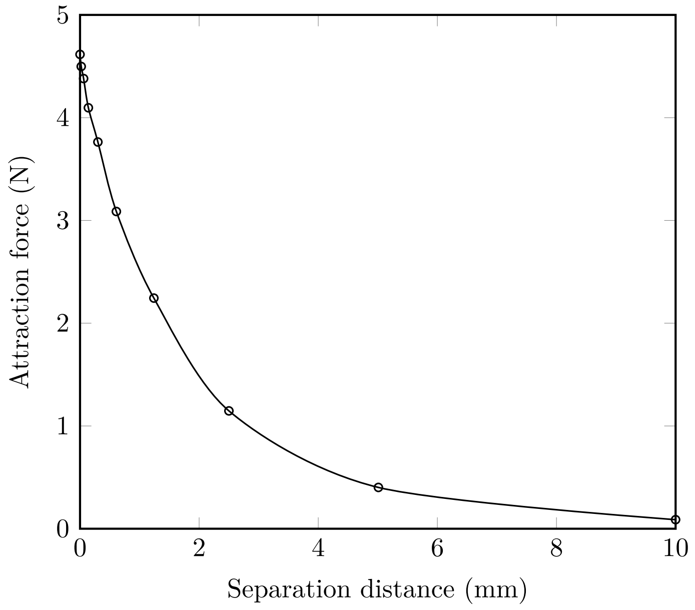


Figure 4.2: Plot of the attraction force between two spherical magnets.

4.1.2 Mechanical design

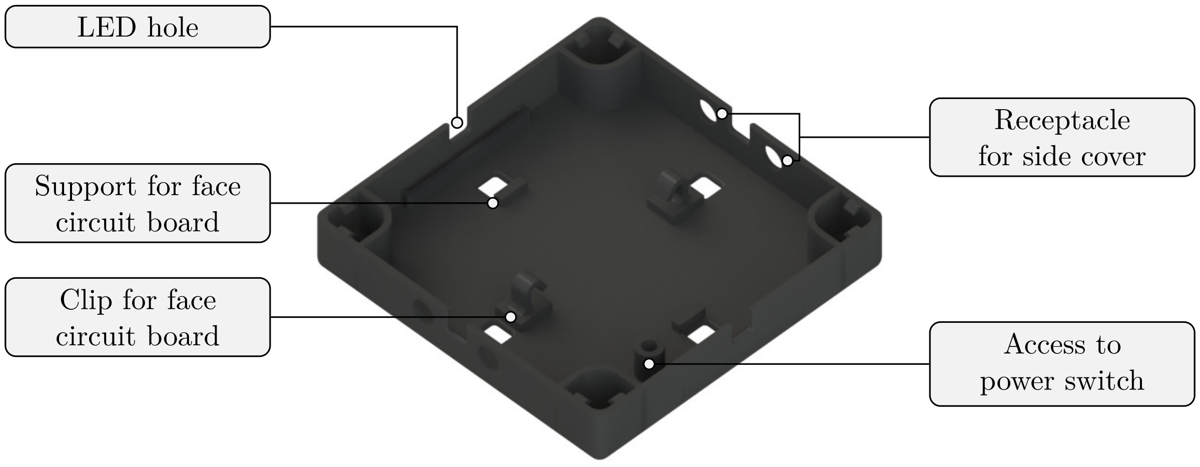
A stigmergic block is assembled from circuit boards, spherical magnets, and covers that we have printed using selective laser sintering. In this design, we use three types of covers: a side cover, a top cover, and a bottom cover. A stigmergic block is cubic in shape and has a side length of 55 millimeters.

Eight spherical neodymium magnets are located in the corners of a block and enable a self-alignment characteristic, which also increases the structural integrity of a structure. These magnets are six millimeters in diameter and weigh 0.9 grams each. Figure 4.2 shows the attraction force between two of these magnets with respect to the separation distance.

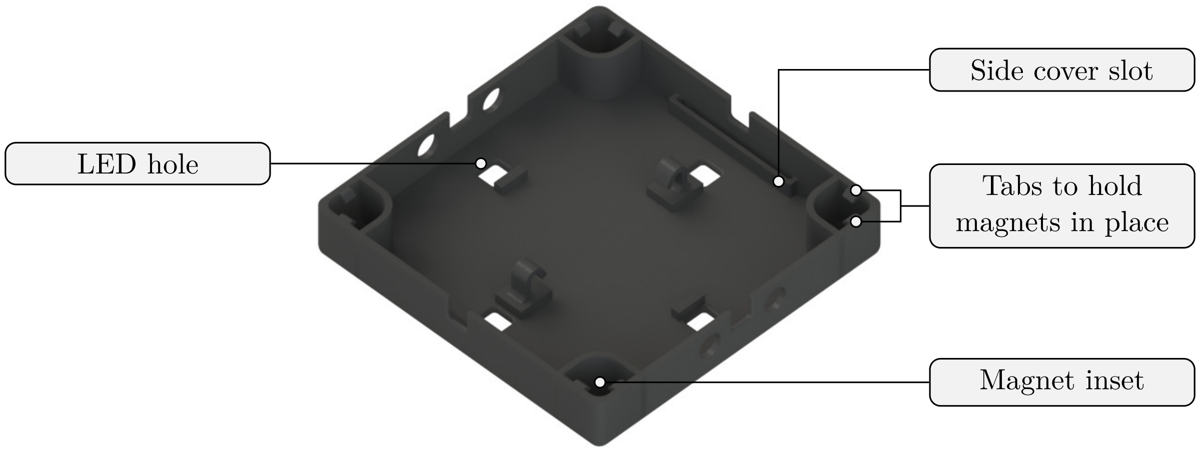
Figure 4.3 shows the mechanical design of the different covers. We have designed the side covers to be used in an alternating up and down configuration. The top and bottom covers have their side cover slots and receptacles at different orientations to accommodate and align with the up and down configuration of the side covers. The side covers contain printed springs, which have been orientated so that the adjacent side covers are held in place using tension. This configuration provides the stigmergic block with structural integrity while allowing the top and bottom covers to be easily removed.

The top and bottom covers both contain four small insets for the spherical magnets. These magnets are held in place using small tabs on the sides of each inset, which allow a magnet to be inserted into position while remaining free to rotate. In addition, the top cover contains a small hole located above the power and reset switch on the central circuit board. This hole provides access to the switch using a small screwdriver.

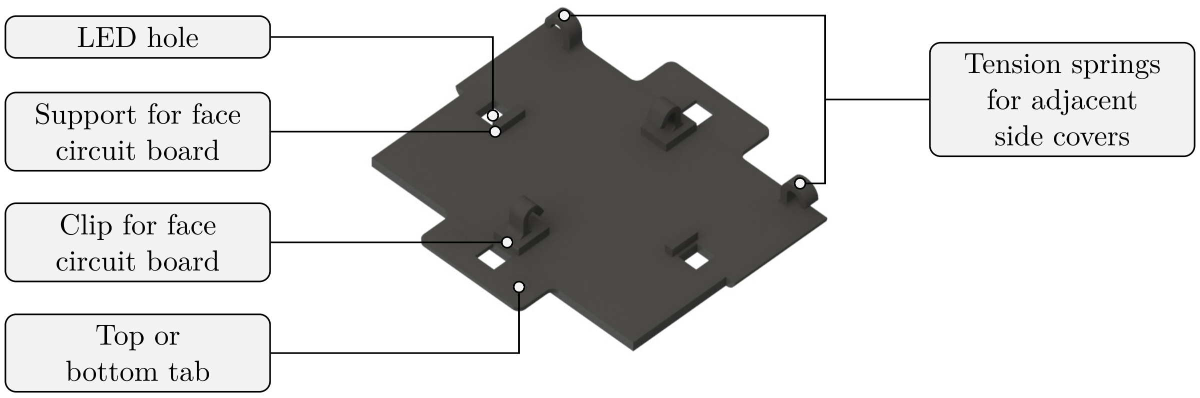
A block consists of four side covers, a top cover, and a bottom cover. To assemble


LED hole
Support for face circuit board
Clip for face circuit board
Receptacle for side cover
Access to power switch

(a)



(b)


LED hole
Support for face circuit board
Clip for face circuit board
Top or bottom tab
Tension springs for adjacent side covers

(c)

Figure 4.3: Mechanical design of the (a) top, (b) bottom, and (c) side covers.

a block, a face circuit board is attached to each of these covers using the two small clips. Each side cover and face circuit board assembly is then attached to a central circuit board by connecting the headers on the face circuit board to the receptacles on the central circuit board. A small lithium-ion battery is connected to and placed underneath the central circuit board.

The top and bottom covers both require four spherical magnets to be inserted into the magnet insets. The face circuit boards in the top and bottom covers connect to the central circuit board using a cable. After the spherical magnets have been inserted and the face circuit boards have been connected, the top and bottom covers slide over the side covers to complete the assembly of a stigmergic block as visually summarized in Figure 4.4. Once a stigmergic block is fully assembled, it has a weight of 110 grams. A localizable tag is placed on each cover of a block so that it can be seen by an autonomous robot.

4.1.3 Software

The microcontroller on the central circuit board contains 32KB of flash memory, 2KB of SRAM, and 1KB of EEPROM. The flash memory is partitioned to include the Optiboot bootloader, which enables reprogramming the microcontroller over USB. The bootloader has been configured with a baud rate of 57600 and runs following a reset of the microcontroller. The microcontroller is programmed using AVRDUDE[4], which automatically resets the microcontroller by pulsing the DTR signal, assuming the Arduino profile is selected.

The firmware for the stigmergic block is written in C++. After the C++ runtime completes its initialization, the microcontroller enters the *main* function, where a singleton instance of a firmware class is created. The constructor for this class initializes several peripherals such as a timer, controllers for the serial ports, and a controller for the I^2C bus. Following initialization of the peripherals, the microcontroller probes each of its ports to determine whether a face circuit board is connected.

For each connected face circuit board, the firmware initializes the NFC transceiver and the LED driver. Once the firmware has initialized each face circuit board, it enters and remains in a loop, until an interrupt is received from one of the faces circuit boards. This interrupt indicates that a robot is trying to communicate with a block using its NFC interface on one of the block's face circuit boards. Upon receiving a message from the robot, the block uses the first byte of this message to configure the color of its LEDs.

This software is relatively simple with respect to the potential functionality of a stigmergic block. For instance, we are currently investigating an implementation of a light-weight pre-emptive operating system for the block. This would enable messages to be sent and received from different faces of a block concurrently, allowing for block-to-block communication inside a structure. Further work based on this block-to-block communication may include adding routing protocols, eventually leading to research into the autonomous construction and maintenance of smart structures.

[4]AVRDUDE: `http://www.nongnu.org/avrdude/`

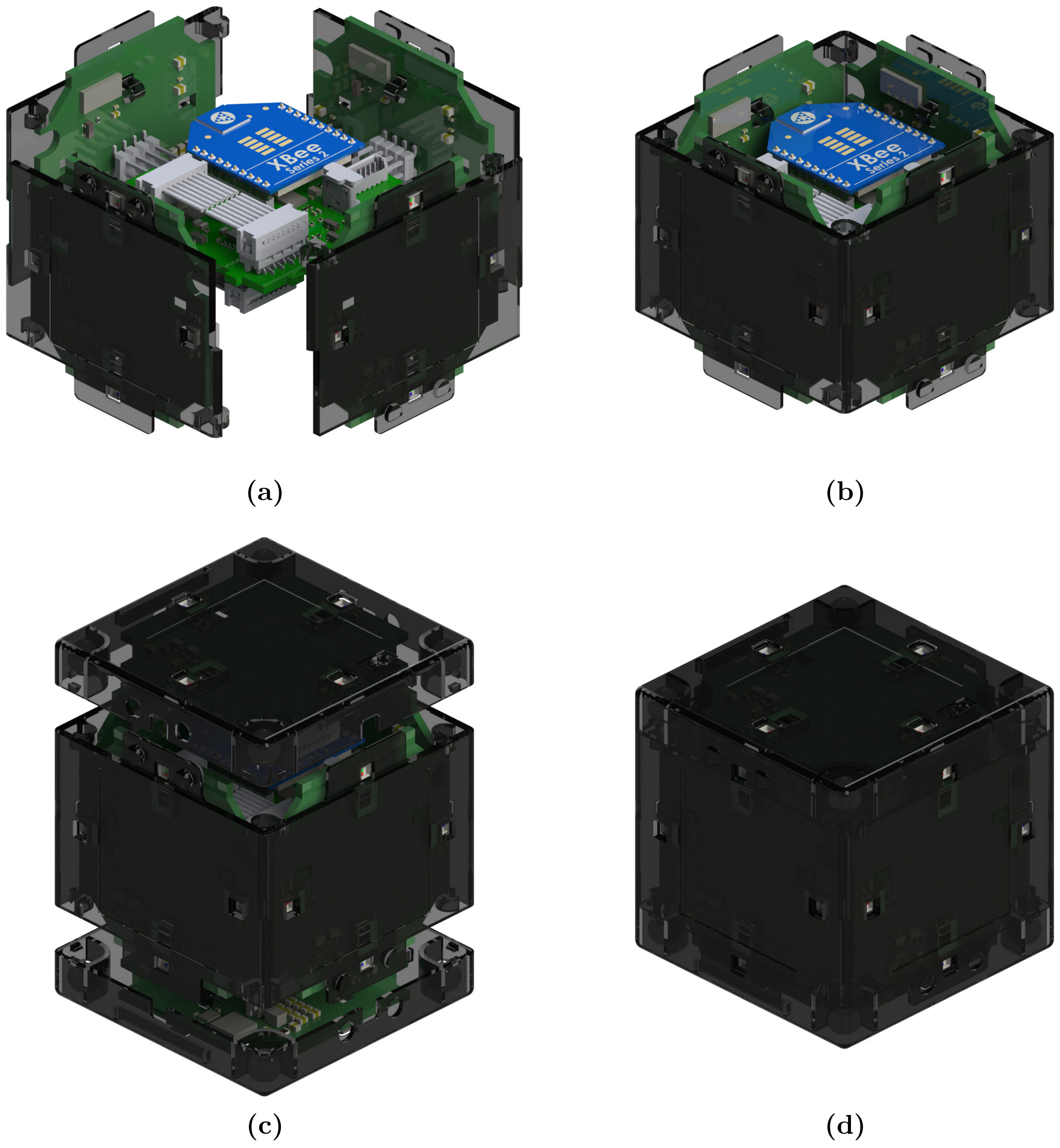


Figure 4.4: The assembly of a stigmergic block. (a) Four side covers attach to four face circuit boards following an alternating up-down configuration, (b) each side cover and face circuit board assembly connects to the central circuit board, (c) the top and bottom covers attach to two face circuit boards, (d) the top cover and bottom cover, including the attached face circuit boards, slide over the side covers of a block to complete its assembly.

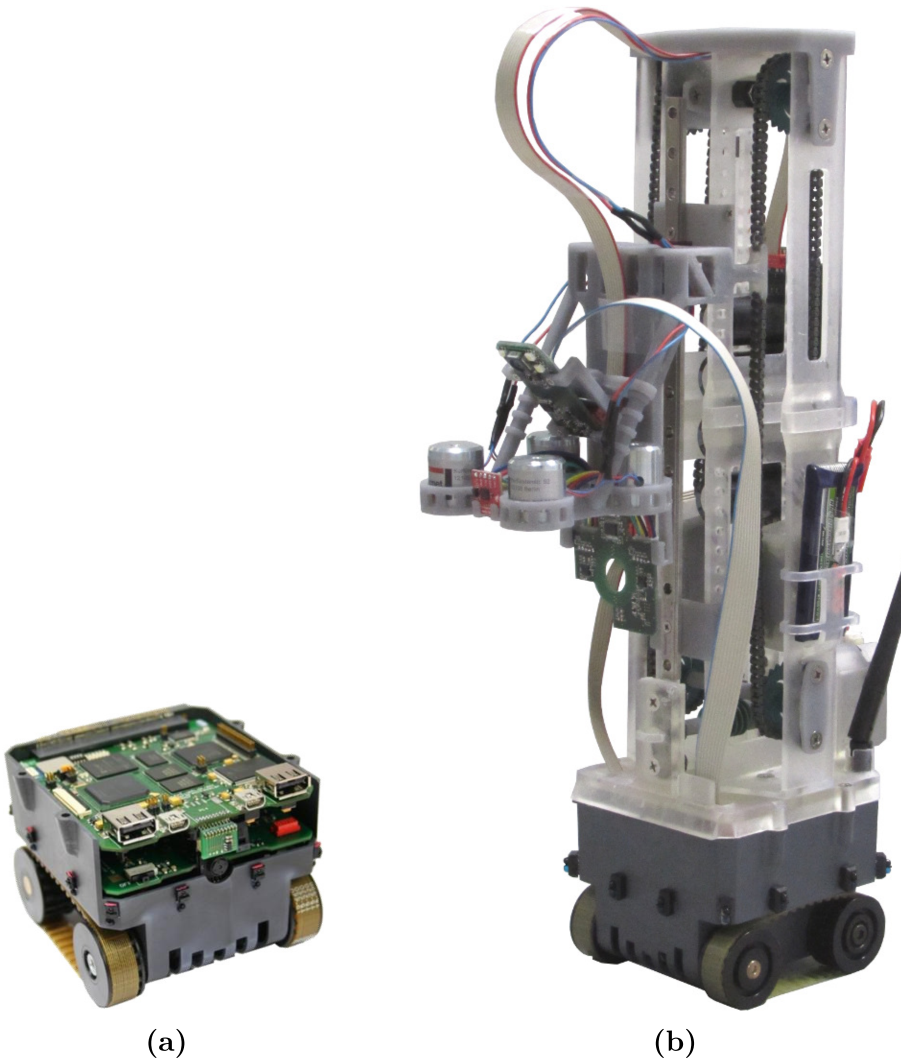

(a) (b)

Figure 4.5: (a) The BeBot mobile robotics platform (reprinted from [24] with permission, © 2011, IEEE), (b) the upgraded mobile robotics platform with the manipulator.

4.2 Autonomous robots

The autonomous robots consist of a mobile robotics platform (an upgraded version of the BeBot [26]) and a manipulator for working with the stigmergic blocks (Figure 4.5). The mobile robotics platform consists of twelve equally-spaced range finders mounted to a molded interconnect device (MID) chassis. The range finders connect to a microcontroller on the chassis, which samples the sensors and provides access to their readings over a serial interface. Two motors mounted to the chassis form a differential drive, allowing the mobile robotics platform to move around its environment. Two circuit boards, which slot into the chassis, are responsible for routing the power, expansion port signals, and the sensor and actuator signals to a central microprocessor.

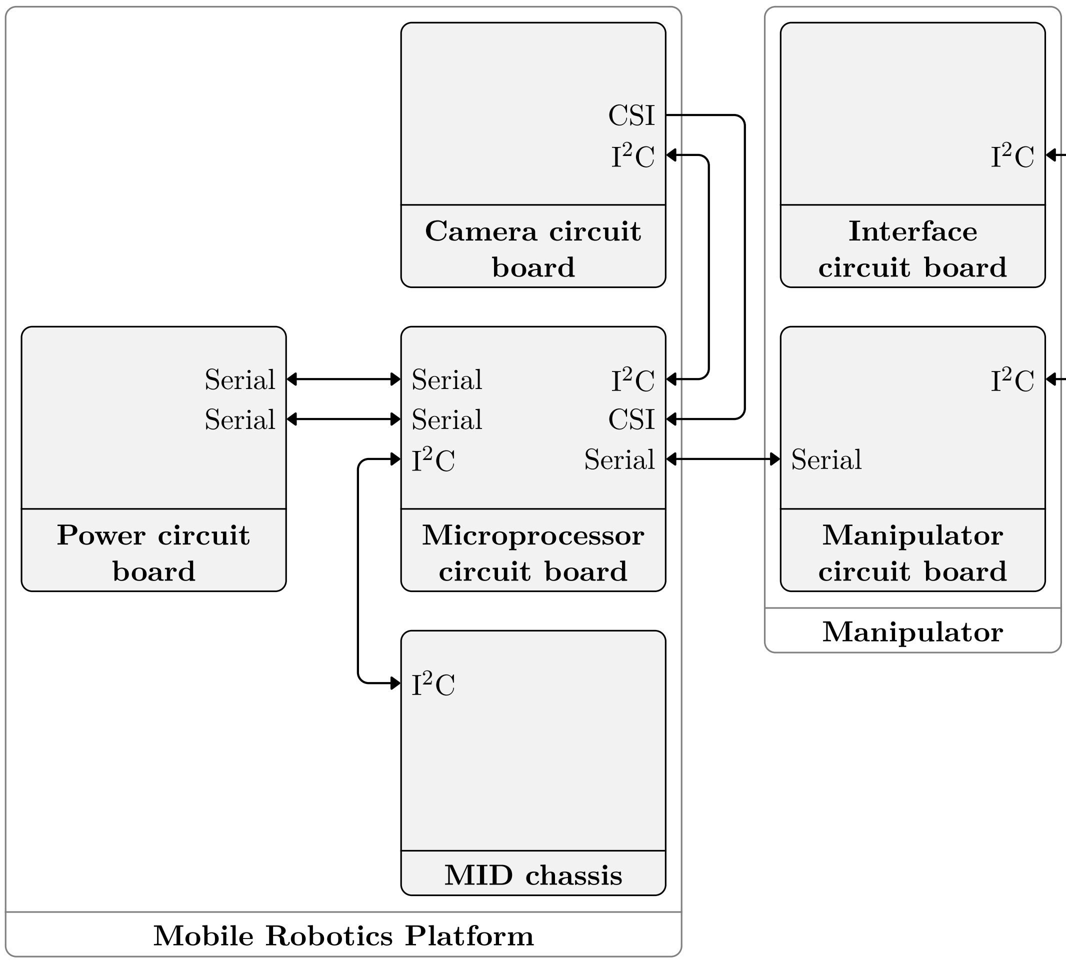


Figure 4.6: Connectivity diagram for the autonomous robot.

Since the microprocessor used in the original mobile robotics platform was inadequate with respect to our computer vision requirements, we have redesigned the two circuit boards around a later generation microprocessor. To reduce development time and manufacturing costs, we use a Duovero Computer-on-Module (COM) from Gumstix, which contains a suitable microprocessor.

To enable an autonomous robot to assemble stigmergic blocks into a structure, we have designed a manipulator, which attaches to the top of the mobile robotics platform. The manipulator controls the vertical position of an end-effector, consisting of four semi-permanent electromagnets. These electromagnets couple with the freely-rotating, spherical magnets inside a stigmergic block, holding it in place during transport.

To locate the stigmergic blocks in an environment, the end-effector is equipped with four range finders and a camera. We have mounted the camera at an angle of 45 degrees from the horizontal. This angle provides a compromise between allowing an autonomous robot to detect a stigmergic block when it is at a distance and when it is nearby. When the end-effector is positioned at its maximum height from the ground (3.5 blocks, or

19.25 centimeters), the camera can see blocks on the ground up to approximately 35 centimeters away from the center of the robot. When the end-effector is positioned at its minimum height from the ground (1 block, or 5.5 centimeters), the blocks can be tracked until they disappear underneath the end-effector.

4.2.1 Electronics

As shown in Figure 4.6, an autonomous robot consists of six interconnected circuit boards. Two of these boards belong to the manipulator, while the other four are part of the mobile robotics platform. Although the camera circuit board is physically connected to the manipulator, we consider it as part of the mobile robotics platform as this is where its power, data, and control signals are routed to and from.

There are two microcontrollers on the power circuit board and one on the manipulator circuit board, all of which communicate with a main microprocessor using a serial interface. This microprocessor runs Linux and is located on the microprocessor circuit board. The microprocessor is capable of reprogramming the microcontrollers on the other circuit boards. We have designed this capability to enable the reprogramming of the attached microcontrollers wirelessly via the microprocessor. This circumvents the need for physical access to the hardware during an upgrade of the software on a large group of robots. The camera and interface circuit boards and the MID chassis provide access to their sensors and actuators over an I^2C interface. These circuit boards have not been designed to be reprogrammed.

Camera circuit board Computer vision on the autonomous robot is provided using a dedicated circuit board to support an image sensor module from Leopard Imaging (Figure 4.7). This image sensor module is based on the OV5640 image sensor from OmniVision.

The OV5640 image sensor requires a clock signal as well as a digital and an analog power supply. We satisfy these requirements using a 24 MHz oscillator, and two low-dropout (LDO) regulators. We have selected LDO regulators to ensure a noise-free power supply for the image sensor. We have placed four white LEDs around the image sensor to optionally enhance the illumination of the captured scene.

The pixel data from the image sensor is routed from the camera circuit board to the microprocessor circuit board using a cable. This cable also provides the camera circuit board with power and the required control signals over an I^2C bus. The I^2C bus connects to the image sensor to control image acquisition, to an LED driver to control the brightness of the four white LEDs, and to a general purpose input/output (GPIO) expander to provide the enable and reset signals for the image sensor and its oscillator and regulators. This level of control is required to correctly power up the image sensor.

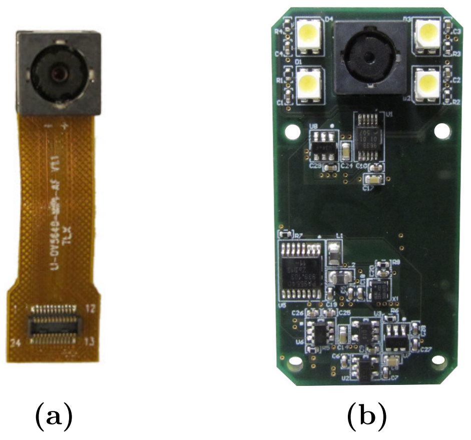


Figure 4.7: Computer vision hardware for the autonomous robot. (a) A module from Leopard Imaging containing the OmniVision OV5640 image sensor. (b) A camera circuit board with an installed module.

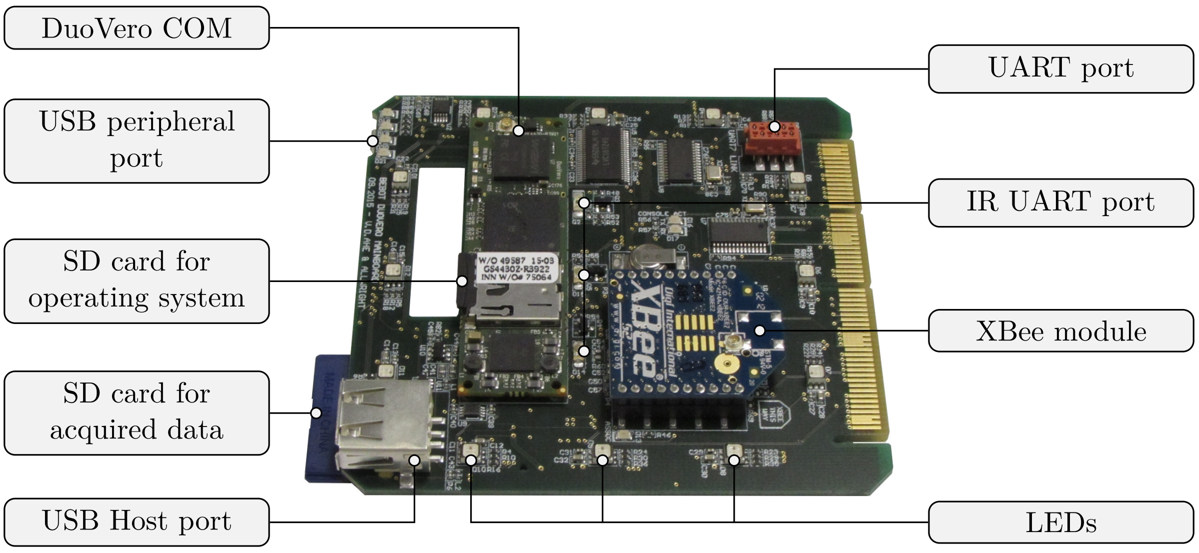


Figure 4.8: The microprocessor circuit board for the mobile robotics platform.

Microprocessor circuit board Figure 4.8 shows the DuoVero COM attached to the microprocessor circuit board. The DuoVero COM provides the main microprocessor for an autonomous robot. This microprocessor runs Linux and has two cores, which are clocked at 1 GHz and share 1 GB of memory. The DuoVero COM also provides Bluetooth and facilitates access to a standard wireless network.

The microprocessor on the DuoVero COM provides two camera serial interface (CSI) ports, which can simultaneously capture video. We have routed both of these ports to two custom connectors, which can connect to two camera circuit boards. These connectors are located on the bottom of the microprocessor circuit board near the cut out on the left-hand side (see Figure 4.8). For a use case involving the capture of video

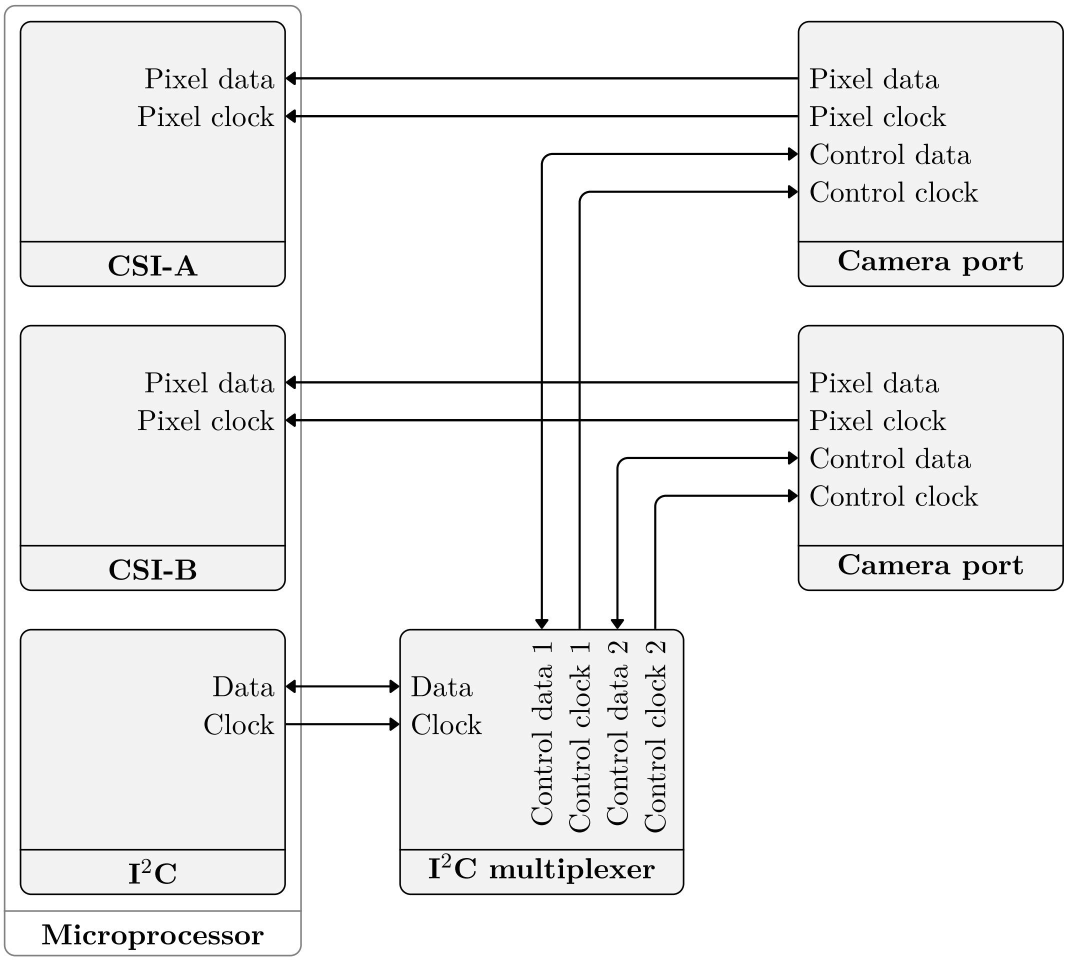


Figure 4.9: Image acquisition connectivity diagram.

from two identical camera circuit boards, an issue arises due to address conflicts on the I^{2}C bus. Figure 4.9 shows how we have solved this issue by adding a multiplexer to the microprocessor circuit board, segmenting the I^{2}C bus. Capturing video over CSI enables the use of the microprocessor's dedicated image processing hardware. This hardware can capture, scale, and compress the video stream from a connected camera. In contrast, a USB camera requires that most of these operations be performed on the CPU, which consumes resources that an autonomous robot could otherwise use for computer vision.

To store the captured images from an autonomous robot, we have added an SD card reader to the microprocessor circuit board. In addition, a standard USB host port is provided, to which any standard USB device can be connected.

To ease development, an autonomous robot can be connected to a PC via its micro-USB port. This port is routed to an integrated USB hub, which provides a developer with low-level access to a robot's bootloader (via an onboard USB-to-serial converter) and high-level access to the robot's operating system by emulating an Ethernet connection over USB On-The-Go (OTG). The integrated USB hub is compliant with the USB

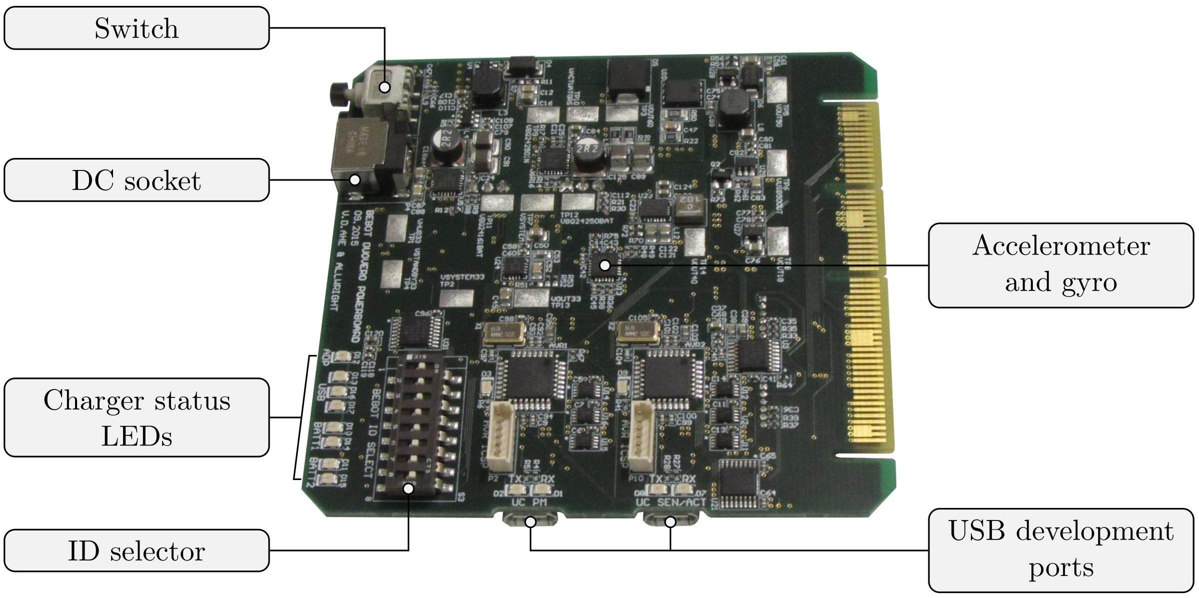


Figure 4.10: The power circuit board for the mobile robotics platform.

battery charging specification, enabling the robot to safely draw current from the USB connection in order to run its system, charge its batteries, or both.

The DuoVero COM on the microprocessor circuit board provides two serial ports. The first serial port is used to access the robot's bootloader and the second serial port is connected to a socket for a low-power wireless module[5]. In our design, we require four further serial connections to communicate with the two microcontrollers on the power circuit board, the microcontroller on the manipulator circuit board, and one additional serial connection for an infrared interface, which is used to maintain backward compatibility with the modules described in [24]. To satisfy this requirement, we include two I^{2}C to serial bridges in our design, which provide the four serial connections.

For debugging and inter-robot communication, twelve multi-color LEDs are evenly spaced around the perimeter of the microprocessor circuit board.

Power circuit board The power circuit board (Figure 4.10) hosts two systems: the sensor-actuator system and the power management system. The sensor-actuator system provides a differential drive for the mobile robotics platform. This system contains a microcontroller, which implements a closed-loop controller for the left and right wheels. Embedded shaft encoders in the motors enable the microcontroller to measure changes in the position of the wheels. The target velocity for the closed-loop controller is set by the microprocessor using a serial interface.

The closed-loop controller for the wheels has an update period of 16.3 milliseconds. During this period, the rotation of the two wheels is measured using an interrupt routine, which is triggered on the rising and falling edges of the shaft encoder signals. A timer

[5]Xbee Wireless Modules: `https://www.digi.com/lp/xbee`

overflow interrupt fires at the end of this period, triggering an update of the closed-loop controller. The closed-loop controller then calculates two output duty cycles to be applied to the motors over the next period. These duty cycles are used to configure the first timer on the microcontroller to generate two pulse width modulation (PWM) waveforms. These waveforms are routed to a motor driver, which is connected to the left and right motors.

The closed-loop controller consists of two PID controllers for the left and right wheels. The tuning of these controllers is predominantly integral, due to issues with the mechanical design of the original hardware. As the wheels are directly attached to motor shafts, the entire weight of the autonomous robot creates a bending moment along the shafts of the motors, probably interfering with the operation of the internal planetary gearboxes in the motors. This interference results in unpredictable friction and occasional jamming. We were only able to partially mitigate these issues by tuning the PID controllers.

In addition to the differential drive for the mobile robotics platform, the sensor-actuator system includes a digital gyroscope-accelerometer sensor. The readings of this sensor are made available to the microprocessor over a serial interface.

The power management system is responsible for routing power and for recharging the batteries in the mobile robotics platform. The power management is broken down into two domains: the system power domain and the actuator power domain. Both of these power domains have their own battery and power management IC (PMIC). External power can be applied to the mobile robotics platform using either the standard 5.5/2.1 millimeter power jack on the power circuit board or the micro-USB connector on the microprocessor circuit board. As shown in Figure 4.11, the external power inputs are connected to the system power management IC, which routes power to the actuator power management IC.

The software controlling the power management system is implemented on a microcontroller. This software configures the integrated USB hub on the microprocessor circuit board and reads the result from the USB hub's battery charger detection circuitry. The software on the microcontroller allocates the power to the mobile robotics platform according to the following prioritized list:

1. system power (if switched on)
2. actuator power (if switched on)
3. system battery (if battery is low)
4. actuator battery (if battery is low)

For each above use of power, the microcontroller subtracts the required amount of power for that use from the remaining available power, which is initially calculated from the input power to the system. As the batteries can be recharged at different rates, the software sets the recharge rate with respect to the remaining available power.

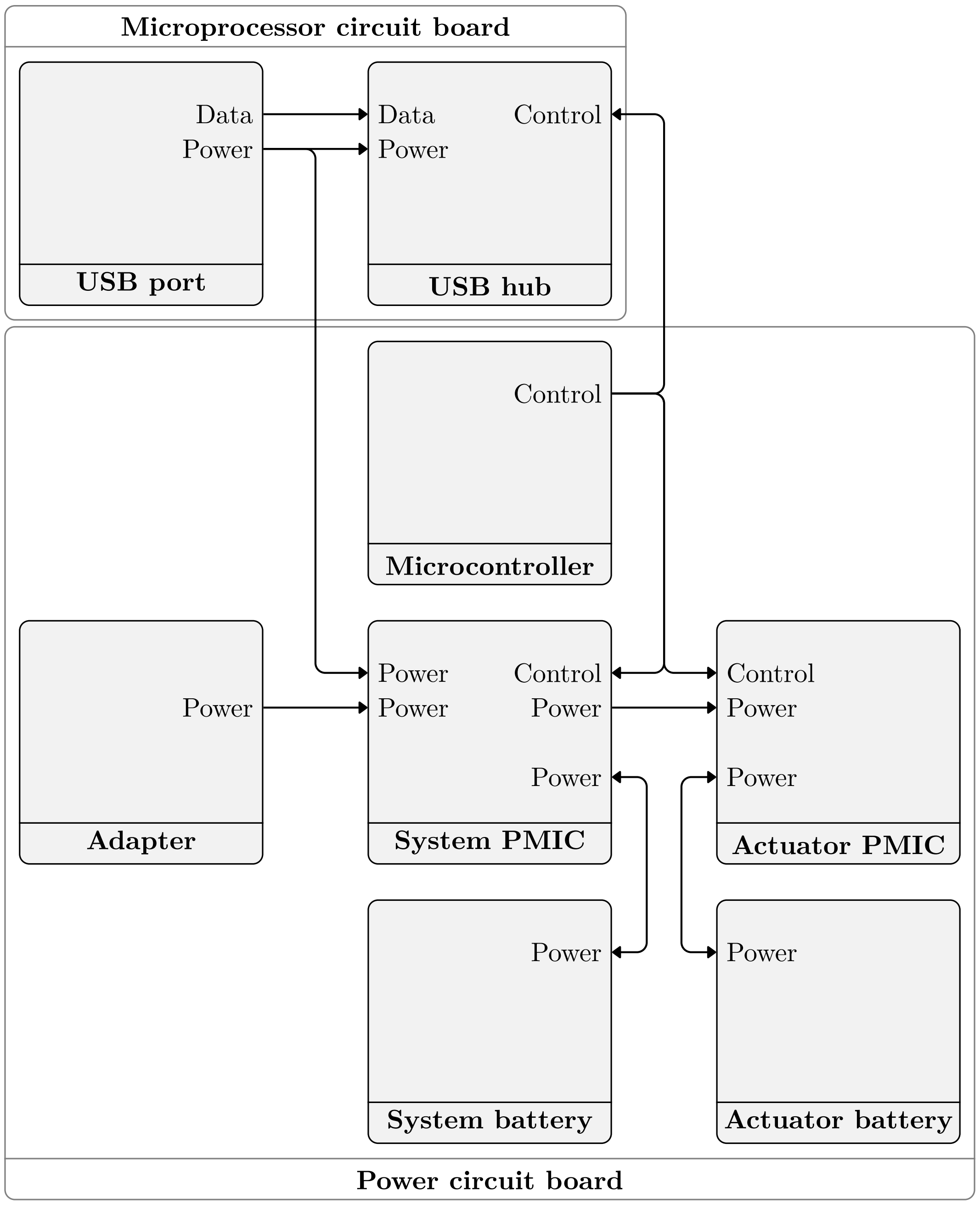


Figure 4.11: Power management system for the mobile robotics platform.

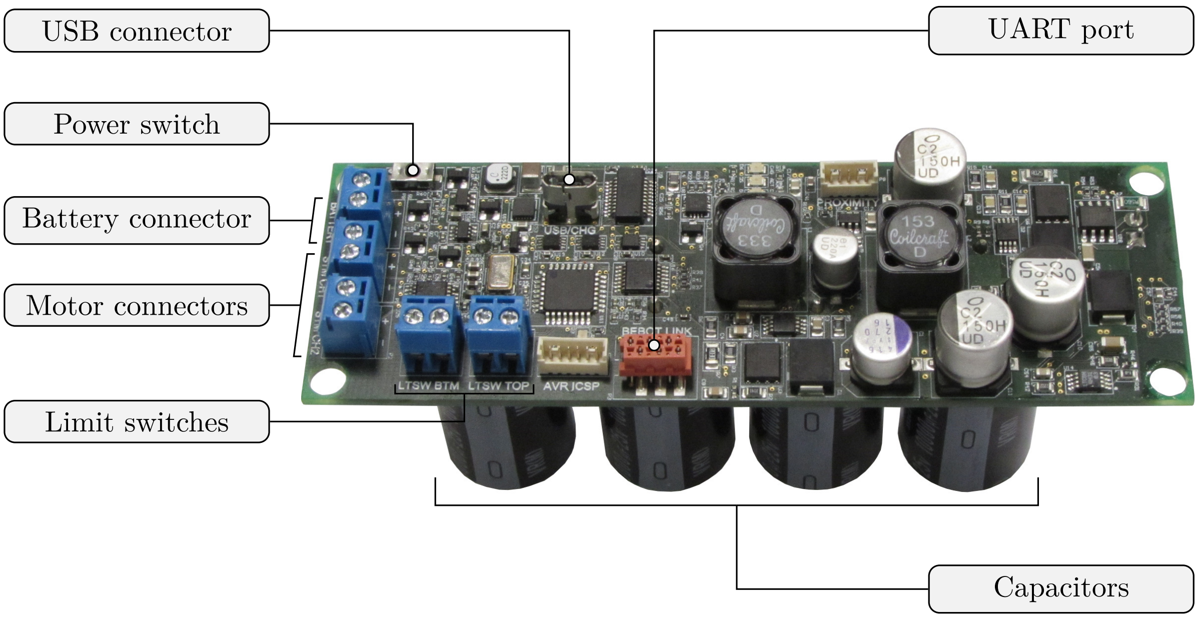


Figure 4.12: Manipulator circuit board.

Manipulator The manipulator adjusts the height of the stigmergic blocks by raising and lowering its end-effector. The height of the end-effector is controlled using a stepper motor and is constrained by two limit switches at the top and bottom of the manipulator. These limit switches trigger as the end-effector starts to move out of range. Four semi-permanent electromagnets are located in the end-effector, which couple with the spherical magnets in a stigmergic block. Depending on the direction of a current applied to the semi-permanent electromagnets, the magnet field can be either strengthened or weakened. This current is generated by precharging four 6.8 millifarad capacitors to 25 volts. The direction of the current is controlled using an H-bridge. An autonomous robot strengthens the magnetic field during block attachment. This strengthening of the field improves the alignment of the stigmergic block with the end-effector prior to the attachment. An autonomous robot weakens the magnetic field in order to detach a block and assemble it into a structure.

The electronics for the manipulator also consists of two circuit boards: a main circuit board and an interface circuit board. The main circuit board (Figure 4.12) contains a microcontroller, which executes the manipulator's software and communicates with the microprocessor on the mobile robotics platform using a serial connection. This serial connection is multiplexed with a USB-to-serial converter. When a USB cable is attached, the serial connection is rerouted over the USB connection to a development PC, which can be used for debugging and upgrading the manipulator's software. The USB-to-serial converter also implements battery charger detection, which configures a power management IC. The manipulator contains its own battery and is charged over the USB connection.

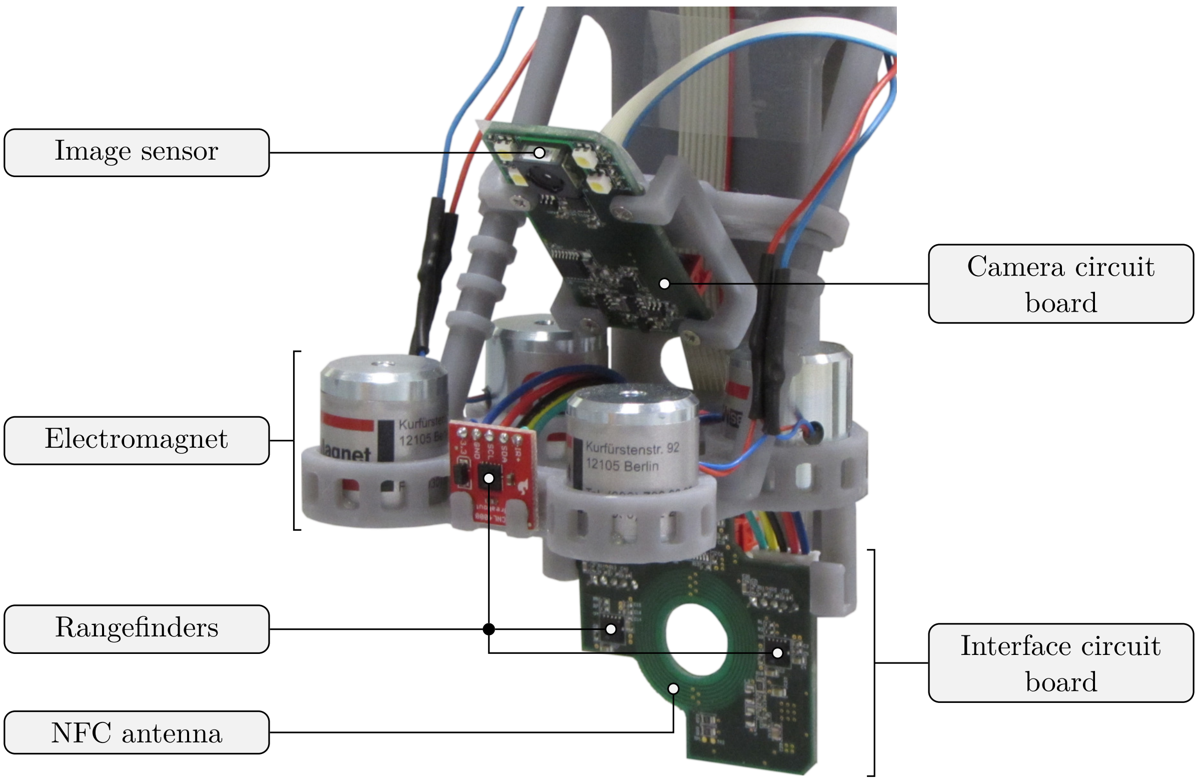


Figure 4.13: Interface circuit board of the manipulator attached to the end-effector.

The interface circuit board (Figure 4.13) contains an NFC transceiver for communicating with a stigmergic block. Two rangefinders are mounted directly to the interface circuit board, which an autonomous robot uses during alignment with a block or with a structure. The interface circuit board also provides two connectors for two additional range finders which are connected to the end-effector.

The software on the microcontroller samples the rangefinders, implements a controller for the NFC transceiver, and controls the precharging of the capacitors and the configuration of the semi-permanent electromagnet H-bridge. The software on the microcontroller also implements an open-loop controller for the height of the end-effector. This controller performs self-calibration of the end-effector, regulates the position of the end-effector, and monitors the state of the limit switches.

4.2.2 Mechanical design

The autonomous robot consists of the mobile robotics platform and a manipulator, which is mounted on top of the platform. Two motors in the mobile robotics platform constitute a differential drive, allowing the robot to move around its environment. The footprint of the mobile robotics platform is a square with a side length of 9 centimeters. The height of the platform is 7 centimeters.

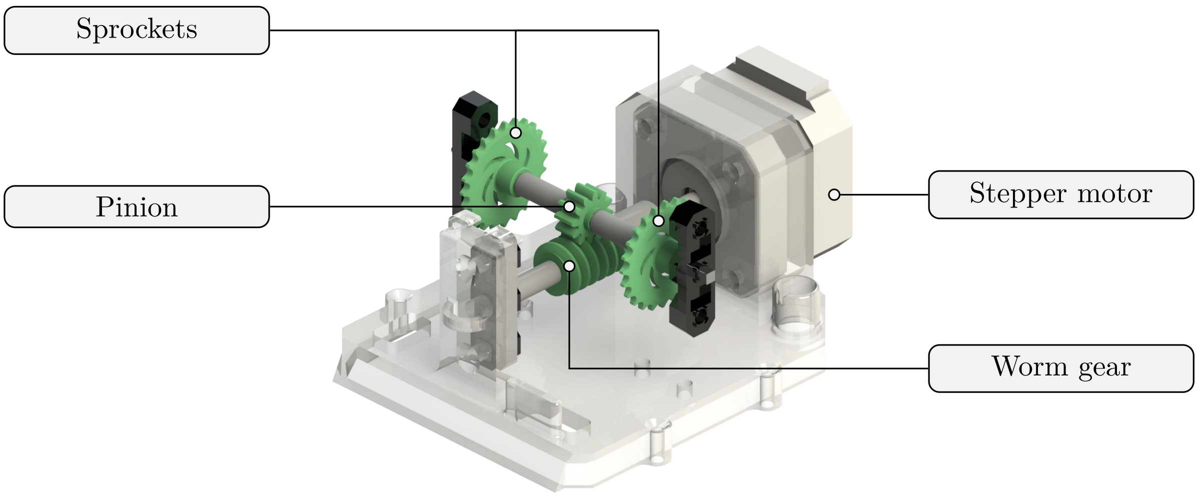


Figure 4.14: Component diagram for the manipulator base.

The manipulator is 30 centimeters tall, and once mounted on top of the mobile robotics platform, gives the autonomous robot an overall height of 37 centimeters. Figure 4.14 shows the base of the manipulator, which is printed from a clear photopolymer resin using stereolithography. A stepper motor is attached to the base using four screws. The motor's shaft is supported by a bearing and drives a worm gear. This worm gear interfaces a pinion, which rotates the lower shaft, driving two sprockets. These sprockets are connected to chains which provide the upwards and downwards motion required by the end-effector. The shafts, bearings, worm gear, pinion, and sprockets are all off-the-shelf components, which have been sourced from Vex Robotics[6].

Figure 4.15 shows two chains running from the bottom sprockets to two upper sprockets, which are suspended by two upper shafts and four bearings. These chains attach directly to the end-effector to change the end-effector's height. To balance the load on the chain, two lead counterweights attach to the chain, opposite the end-effector. Together with the weight from the stepper motor, the weight of the counterweights balances the weight of the electromagnets at the front of the autonomous robot.

The electromagnets and counterweights are off-the-shelf components. The remaining structural components have been printed using either a gray or a clear photopolymer resin using stereolithography. Figure 4.16 shows a part, which we have printed using this technique. This part is the top of the manipulator structure, which aligns the manipulator's columns and supports the bearings for the upper shafts. To prepare this part for use, the support material must be removed. To improve functionality and aesthetics, it is necessary to finish the parts with sandpaper and a polish for plastic surfaces.

[6]Vex Robotics: `http://www.vexrobotics.com/`

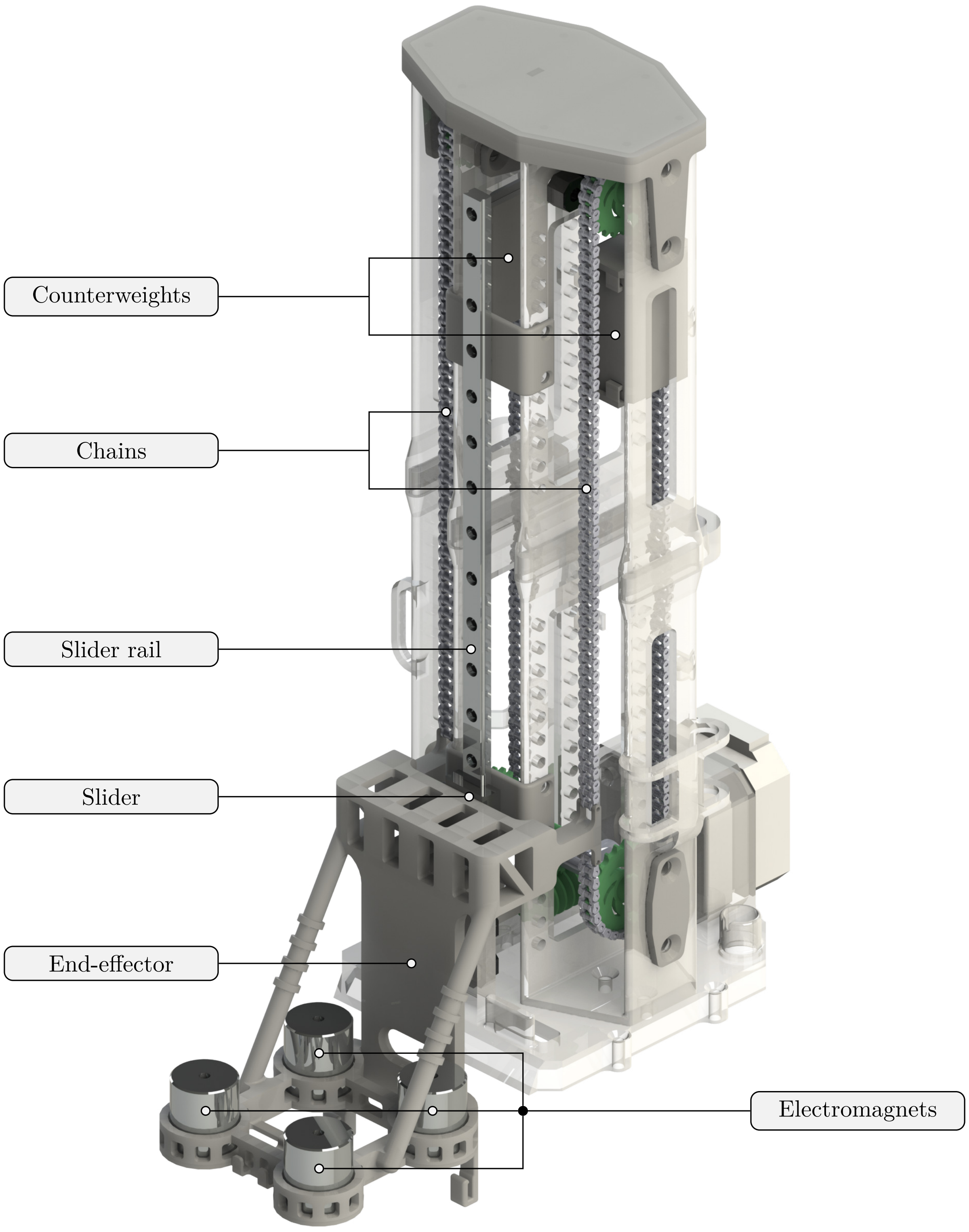


Figure 4.15: Component diagram for the manipulator.

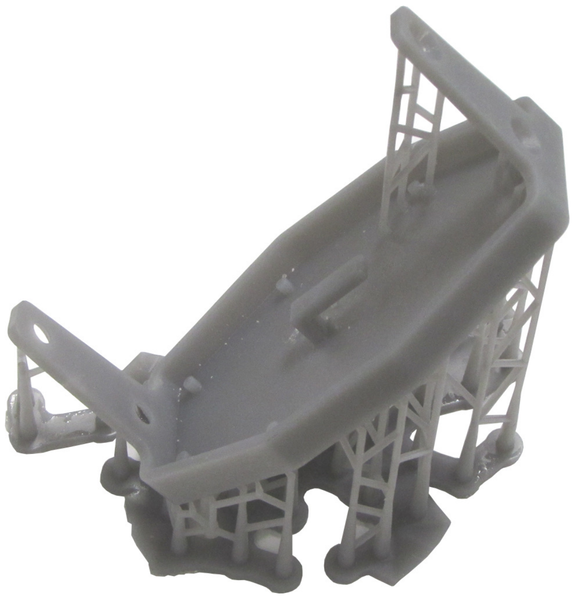


Figure 4.16: An example of a component for the manipulator printed using stereolithography.

4.2.3 Software

Operating system The microprocessor in the mobile robotics platform runs a custom variant of the Linux operating system. This custom variant of Linux is downloaded, configured, and compiled by the Yocto build system[7]. The build system uses *recipes*, which describe the numerous tasks required to prepare a bootable *image* for an embedded system. These tasks may include fetching software from a version control system, applying local patches, and compiling and installing software into the target root file system.

The image for the mobile robotics platform is based on the Gumstix console image[8], which provides a basic configuration of the Linux operating system for the microprocessor. This configuration includes a shell, utilities, and tools for networking and system configuration. We have enhanced this image to support the hardware on the mobile robotics platform by configuring the Linux kernel and adding additional software packages.

We have selected the hardware for the mobile robotics platform with respect to the availability of drivers in the Linux kernel mainline. These drivers are considered to be stable and are well maintained by the Linux community. Furthermore, we have selected hardware for which device tree bindings already exist[9]. This choice of hardware significantly simplifies the process of configuring Linux for the mobile robotics platform.

We have, however, encountered a bug in the Linux kernel, which prevents the clocks of peripheral devices from being detected in the device tree. This bug is fixed in a

[7]Yocto Project: `https://www.yoctoproject.org/`

[8]Gumstix Developer Center: `http://gumstix.org/`

[9]The Devicetree Specification: `https://www.devicetree.org/`

later version of the Linux kernel (4.1), which, however, had an issue with the power management of the microprocessor's imaging subsystem. During the configuration of the Linux kernel for the mobile robotics platform, we also found an issue with enable signals for peripheral clocks connected to I^{2}C GPIO expanders. This configuration generates kernel warnings due to the latencies involved with enabling a clock over an I^{2}C bus. To resolve these issues in a timely manner, we have used an older version of the Linux kernel (3.17) that supports the imaging subsystem. To work around the clock issues, we have patched the drivers to enable their clocks using the Linux GPIO interface and hard coded the respective clock frequencies into the drivers.

Although the driver used for the microprocessor's imaging subsystem is in the Linux kernel mainline, it is part of the staging directory and its quality is not guaranteed. Furthermore, the driver for the robot's camera is out-of-tree and is unmaintained by the Linux community. These drivers required patching before they would work on the mobile robotics platform.

We enhanced the Gumstix console image with additional software packages to support our application. For example, we have included tools for capturing and working with images from the robot camera such as media-ctl[10], yavta[11], and OpenCV[12]. We have added a recipe to compile and install the detector from the AprilTags visual fiducial system as a shared library [71]. We have also created a test application called *blocktracker*, an executable which configures the autonomous robot and runs test routines. We have extended the blocktracker test application to implement the hardware experiments presented in this thesis.

Blocktracker The behavior of the autonomous robot is implemented by the blocktracker executable, which is written in C++ and built using CMake[13]. The executable has four main components: a packet control interface, an image processing pipeline, a finite state machine, and a control loop.

The microcontrollers on the manipulator and power circuit boards communicate with the microprocessor using a packet control interface. We have designed this interface to support multi-byte commands. The packet control interface also checksums the commands and validates their length. Figure 4.17 shows an example command, which queries the battery voltage on a remote microcontroller. A valid command always starts with a two-byte preamble and ends with a two-byte postamble. We have selected the values of these bytes due to their visibility on an oscilloscope. As the example command has no arguments, its length field is zero. For commands which do have arguments, the length field is nonzero and the bytes for the arguments are inserted between the length and checksum fields. In this case, the checksum field represents the summation of the bytes of the arguments. Incoming bytes are stored in a buffer and searched for valid commands using a specialized state machine. If the buffer overflows or an invalid

[10]Media-ctl: `https://git.linuxtv.org/v4l-utils.git/tree/utils/media-ctl`

[11]Yavta: `http://git.ideasonboard.org/yavta.git/tree`

[12]OpenCV: `http://opencv.org/`

[13]CMake: `https://cmake.org/`

F0	CA	01	00	00	53	0F
Preamble		Type	Length	Checksum	Postamble	

Figure 4.17: An example command for the packet control interface.

command is detected, the state machine searches for the next preamble, flushing out any data found before it.

The image processing pipeline is broken down into an asynchronous component and a synchronous component. The implementation of the asynchronous component was motivated due to an earlier implementation, where the microprocessor wasted significant processing time, waiting for the microcontrollers on the manipulator and power circuit boards to respond to commands. The asynchronous component of the pipeline enables the microprocessor to capture and process images from the camera while waiting for the microcontrollers to respond. The asynchronous component of the pipeline is built on top of the concurrency extensions to the C++ standard library[14]. The implementation of the asynchronous component defines an operation class, which contains an operation method, a management thread, a queue of image buffers to be processed, and a pointer to the next operation. When an operation is enabled, its management thread inspects the queue of image buffers every five milliseconds. If an image buffer is found in the queue, the queue is temporarily locked and the image buffer is extracted for processing. Once processing is complete, the operation attempts to lock the queue of the next operation as defined by its next pointer. Once this queue is locked, the operation moves the processed buffer into the next operations queue.

The operation class is specialized to perform the following functions: (i) to capture a frame from the camera, (ii) to stream a frame over a wireless network, (iii) to save a frame to local memory, (iv) to detect the stigmergic blocks in a frame, and (v) to annotate a frame with the output from the detection operation. Upon initialization of the pipeline, we create four image buffers and enqueue them inside the capture operation. The operations are connected in a loop so that the image buffers are recycled and dynamic memory allocation is not required while the pipeline is running.

The stigmergic block detection operation uses the detector from the AprilTags visual fiducial system to find the tags on the stigmergic blocks. This operation also samples the colors of the LEDs on a stigmergic block, which are used for block-to-robot communication. As the images from the microprocessor's imaging subsystem are in the YUV format, we have selected four LED colors with respect to the four quadrants of the UV color space (Figure 4.18). This selection of colors from the UV color space avoids the requirement of performing a color space conversion.

The stigmergic block detection operation maintains a queue of its detections from the processed frames. This queue is lockable and accessible from the synchronous component

[14]Concurrency extensions in C++: `https://isocpp.org/wiki/faq/cpp11-library-concurrency`

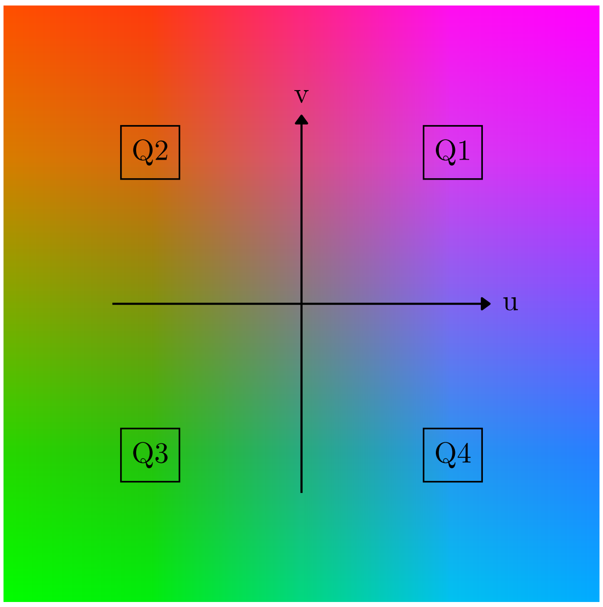


Figure 4.18: The YUV color space with Y = 0.5, showing the four colors used to represent different types of blocks.

of the image processing pipeline, which is executed in the control loop. The synchronous component of the image processing pipeline tracks stigmergic blocks using the Hungarian algorithm with a modified cost matrix, which accommodates new and lost blocks [48]. The tracked blocks are clustered into structures by comparing the distance between any two blocks with a threshold. The results of the image processing pipeline are then available to the finite state machine.

We have developed a state machine library for implementing the behavior of an autonomous robot. The motivation for developing this library was to take advantage of the features in C++11 to create a state machine library that supported reusable sub-states while being compact and easy to read. An example instantiation of a state machine using this library is shown in Listing 4.1. This example demonstrates how classes inherited from the `CState` class can be composed and customized to implement arbitrary behavior. For instance, the class `CStatePrint` inherits from `CState` to define a state that writes the contents of a `std::string` to a `std::ostream`. Line 25 of the example shows how the class `CStatePrint` is configured using its constructor via the templated `AddState` method. The `AddState` method uses the perfect forwarding and the variadic template mechanisms of C++11 to automatically insert the parent pointer as the second argument to the `CStatePrint` constructor. The class `CStateFooBar` demonstrates how the `CStatePrint` class is composed using the initialization list and how the transitions are defined in the constructor body to create the finite state machine shown in Figure 4.19, which writes the string “foobar” to `std::cout` and exits.

The blocktracker executable starts with an initialization routine, which connects to the remote microcontrollers on the power and manipulator circuit boards and checks if they are responding to commands using the packet control interface. The initialization

```cpp
#include <iostream>

#include "state.h"

class CStatePrint : public CState {
public:
   CStatePrint(const std::string& str_id,
               CState* pc_parent,
               std::ostream& c_device,
               const std::string& str_data) :
      /* init the base class id and function with a lambda */
      CState(str_id, pc_parent, [this, &c_device] {
         c_device << m_strData;
      }),
      m_strData(str_data) {
   }
   std::string m_strData;
};

class CStateFooBar : public CState {
public:
   CStateFooBar(const std::string& str_id, CState* pc_parent = nullptr) :
      CState(str_id, pc_parent, nullptr, CState::TVector {
         /* add states */
         AddState<CStatePrint>("print_foo", std::cout, "foo"),
         AddState<CState>("print_bar", nullptr, CState::TVector {
            /* add sub-states */
            AddState<CStatePrint>("print_b", std::cout, "b"),
            AddState<CStatePrint>("print_a", std::cout, "a"),
            AddState<CStatePrint>("print_r", std::cout, "r"),
         }),
      }) {
      /* declare state transitions */
      AddTransition("print_foo","print_bar");
      AddExitTransition("print_bar");
      /* declare sub-state transitions */
      GetState("print_bar").AddTransition("print_b","print_a");
      GetState("print_bar").AddTransition("print_a","print_r");
      GetState("print_bar").AddExitTransition("print_r");
   }
};

int main(int argc, char* args[]) {
   /* instansiate state machine */
   CStateFooBar cStateFooBar("fsm");
   /* run until exit transition */
   for(;;) {
      if(cStateFooBar.Step() != false) {
         break;
      }
   }
   return 0;
}
```

Listing 4.1: Example instantiation of a finite state machine that writes the string "foobar" to `std::cout` and exits.

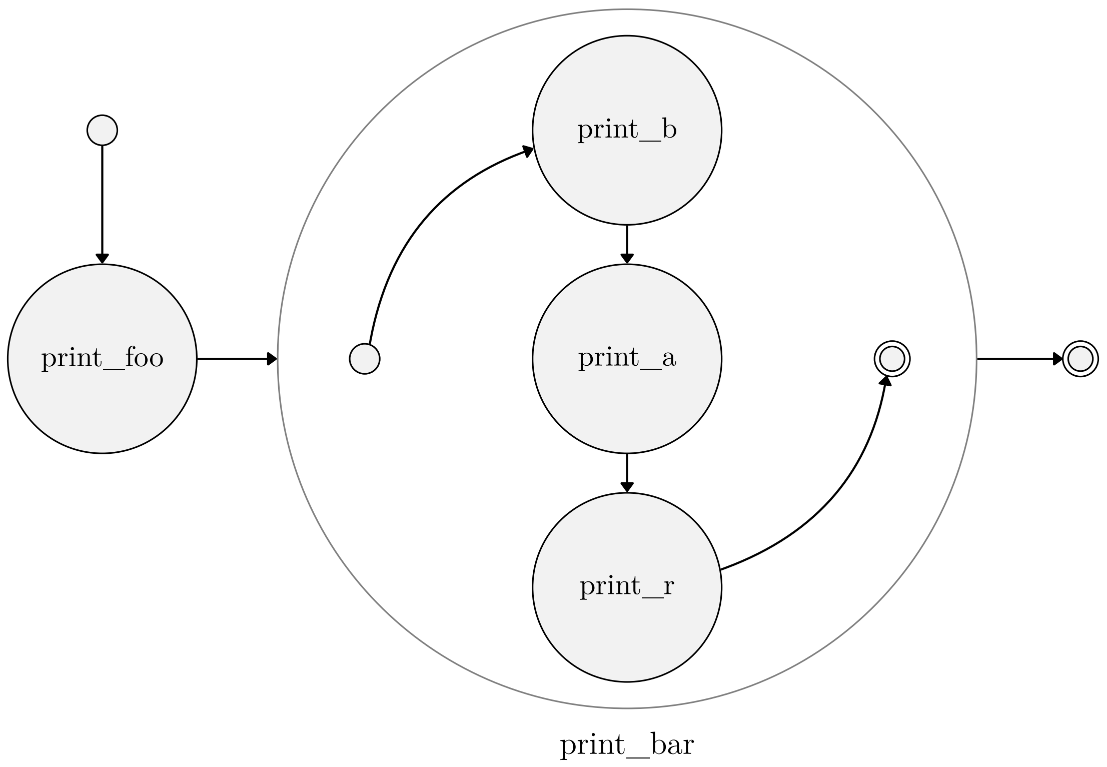


Figure 4.19: The finite state machine `CStateFooBar` as produced by the code in Listing 4.1.

routine then configures the microprocessors imaging subsystem and the camera, before initializing the image processing pipeline. The executable then enters a method called `exec`. This method starts by enabling the actuator power domain and the differential drive system. The method then requests the manipulator to perform its self-calibration routine. After the calibration is complete, the `exec` method enters the control loop, which samples an autonomous robot's sensors, steps its behavioral state machine, and updates its actuators. The control loop continues until the behavioral state machine exits. The image processing pipeline is the largest bottleneck in the control loop and limits the update period to approximately 160 milliseconds in good lighting conditions. To keep the update period for the high-level closed-loop controllers constant, we lengthen the update period so that it always lasts 200 milliseconds.

Figure 4.20 shows the base behavioral state machine for an autonomous robot. The state machine starts with an autonomous robot searching its environment for unused blocks. Once an unused block is found, the robot picks it up and begins to search the environment for a partially-built structure. When a structure is found, the robot inspects it to determine if a pattern of blocks with an associated construction action can be found. If such a pattern is found, the robot performs the construction action. Otherwise, the robot continues searching for another partially-built structure.

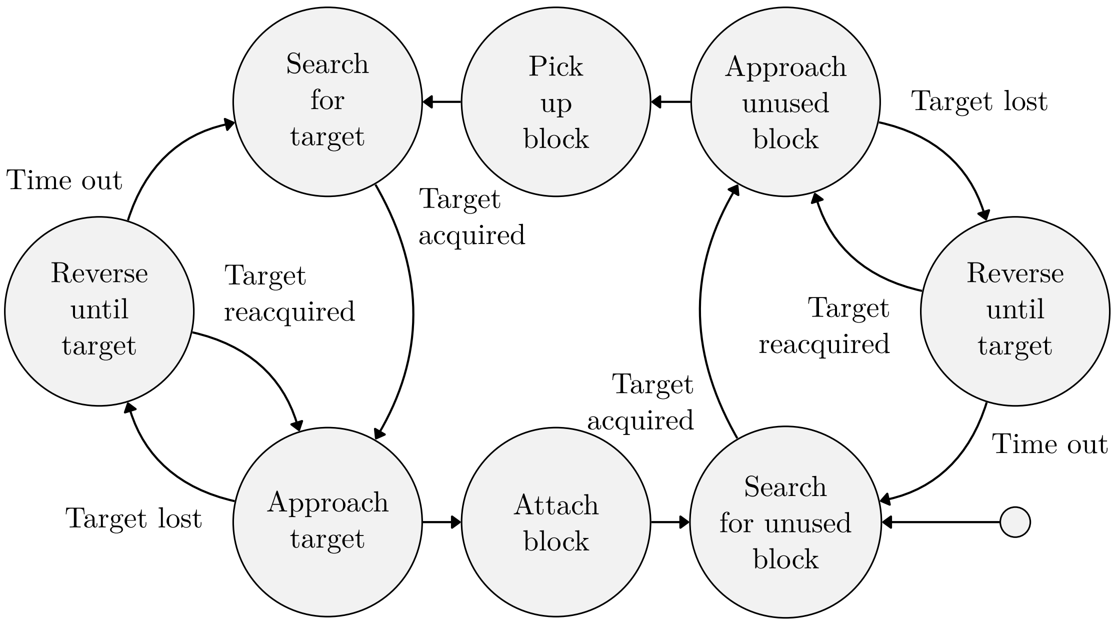


Figure 4.20: The base behavioral state machine for an autonomous robot.

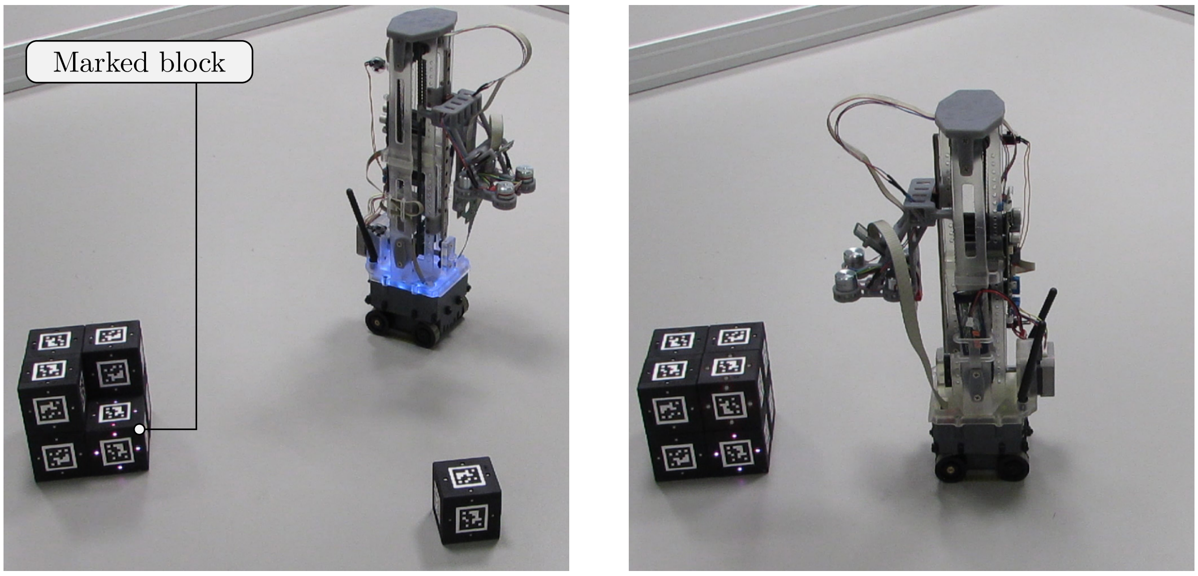
4.3 Verification of the hardware

We have designed two tasks to verify the functionality of the hardware. These tasks test an autonomous robot's ability to place stigmergic blocks at different locations in the structure and its ability to respond to patterns in an environment. These patterns are expressed in terms of the structural arrangement of the stigmergic blocks in the environment and their LED markings.

For an autonomous robot to complete either of these tasks successfully, there must not be any issues in the electronics, mechanical design, or software for a stigmergic block or an autonomous robot. For the following tasks, a failed trial occurs if an autonomous robot does not pick up an unused stigmergic block or fails to attach an unused stigmergic block to the location specified by the task.

4.3.1 Markings-based task

The markings-based task is designed to verify that an autonomous robot can respond to a markings-based stimulus. In this task, an autonomous robot must locate an unused stigmergic block and place it on top of an illuminated block that forms part of a structure. A structure is defined as at least two contiguous stigmergic blocks. The experiment is set up such that an autonomous robot can find both an unused block and the illuminated block by turning on the spot. The illuminated block is placed into a partially-built structure, which once complete is a $2 \times 2 \times 2$ cube. Figure 4.21 shows the setup of the markings-based task and the task after it has been completed successfully.



Initial setup Task complete

Figure 4.21: Markings-based verification task.

A run of the markings-based task proceeds as follows: (i) a robot turns on the spot, searching its environment for an unused block, (ii) upon locating an unused block, the robot approaches it and picks it up, (iii) the robot continues to turn on the spot, searching for an illuminated block that forms part of a structure, (iv) upon locating an illuminated block that forms part of a structure, the robot approaches the illuminated block and places the unused block on top of it.

4.3.2 Structure-based task

The structure-based task is designed to verify that an autonomous robot can respond to a structure-based stimulus. In this task, an autonomous robot must locate an unused stigmergic block and place it against the larger of two detected structures. Upon locating the larger structure, the autonomous robot adds the unused block to the structure by attaching it to the front of the leftmost block.

The size of a structure is estimated using a heuristic: If a block in a structure has more adjacent blocks than the currently selected block, the autonomous robot selects the block with the highest number of adjacent blocks; otherwise, the robot uses the number of adjacent blocks of the currently selected block as an estimate of the structure size.

The experiment is set up so that the autonomous robot can find an unused block and two structures of different sizes by turning on the spot. Figure 4.22 shows the setup of this task and the task after it has been completed successfully by an autonomous robot.

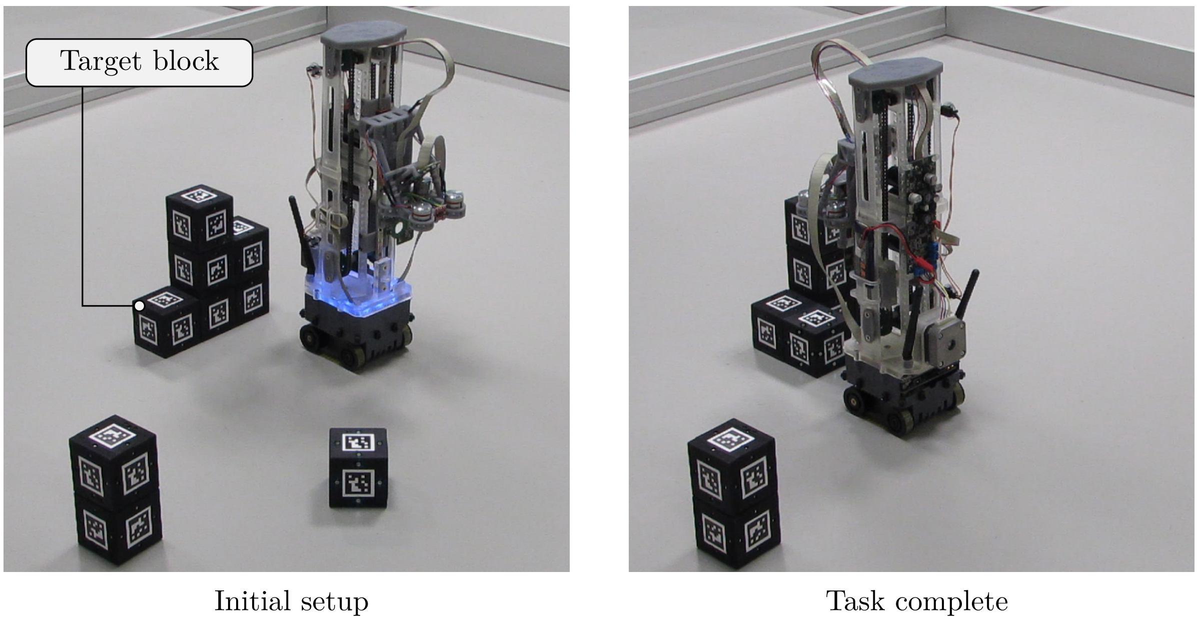


Figure 4.22: Structure-based verification task.

A run of the structure-based task proceeds as follows: (i) the robot turns on the spot, searching its environment for an unused block, (ii) upon locating an unused block, the robot approaches it and picks it up, (iii) the robot continues to turn on the spot, searching for a structure, (iv) upon locating a structure, the robot estimates its size and continues searching its environment for a second structure, (v) upon locating the second structure, the robot estimates its size, (vi) if the first structure was larger, the robot turns on the spot in the opposite direction until it locates the larger structure again, (vii) the robot places the unused block against the leftmost block in the larger structure.

4.3.3 Results

We ran 15 trials for each task and obtained a success rate of 73.3% for the markings-based task and 80% for the structure-based task. The raw data for these tasks is shown in Table 4.1. Figure 4.23 contains a box plot of the task run time for the successful trials. The larger variance in run time for the structure-based task is due to the experimental setup. In half of the cases, an autonomous robot encountered the larger structure before the smaller structure and required additional time to find the larger structure again before attaching the unused block.

In all trials, the unused block was successfully located, picked up, and configured by an autonomous robot. The failed trials occurred during block placement due to misalignment between the autonomous robot and a partially-built structure. This misalignment was caused by issues with the differential drive system. As discussed in Section 4.2.1, these issues result in uneven friction and occasional jamming. These issues aside, the two tasks have verified that the rest of the hardware is functioning correctly.

Trial	Run time	Result
1	96	Pass
2	91	Pass
3	95	Pass
4	95	Pass
5	96	Pass
6	-	Fail
7	101	Pass
8	-	Fail
9	105	Pass
10	-	Fail
11	110	Pass
12	100	Pass
13	100	Pass
14	-	Fail
15	100	Pass

Markings-based task

Trial	Run time	Result
1	98	Pass
2	119	Pass
3	-	Fail
4	115	Pass
5	103	Pass
6	123	Pass
7	-	Fail
8	111	Pass
9	-	Fail
10	101	Pass
11	117	Pass
12	109	Pass
13	108	Pass
14	112	Pass
15	127	Pass

Structure-based task

Table 4.1: Raw data from the two verification tasks. Run time is in seconds.

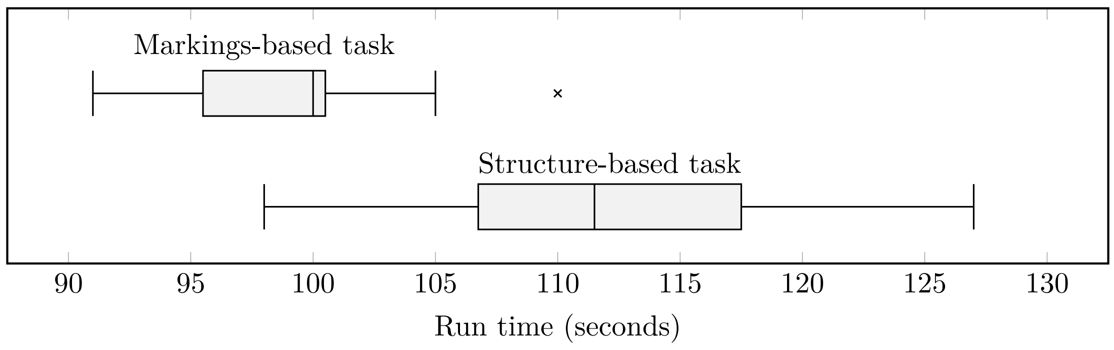


Figure 4.23: Box plot of the run time for the two verification tasks.

CHAPTER 5 Implementation of the Simulation

To support the development and debugging of decentralized control strategies for our multi-robot ACS, we also provide an implementation of our system in simulation. This simulation must be realistic in order to replicate the results from the hardware. This requirement, in turn, requires the autonomous robots, the stigmergic blocks, and their interactional dynamics to be modeled accurately.

In addition to the development and debugging of decentralized control strategies, we intend to use simulations to gather data from experiments involving large numbers of robots and blocks. To run such experiments, the simulation of the multi-robot ACS must be efficient. Furthermore, as we intend to extend an existing robot simulator, we require a simulator that is modular. This modularity eases the implementation of any missing aspects of the simulation.

5.1 The ARGoS simulator

ARGoS is a modular, multi-robot simulator designed for running experiments with large numbers of robots [78]. Benchmarks for ARGoS using the ODE physics engine[1] have demonstrated the simulation of 10,000 robots running faster than real-time. We have selected ARGoS as it is both modular and efficient. In the rest of this section, we provide an overview of the components of the simulator from the perspective of their configuration. Further information, such as how to download, install, and run experiments in ARGoS, can be found on the simulator's website[2].

A simulation in ARGoS is configured using an XML configuration file, which is provided by a user. This configuration file is split into seven subtrees, which are elements of the `<argos-configuration>` root element. These subtrees are listed below and are used to configure the following aspects of a simulation.

`<framework>` configures the number of threads used by the simulator, the length of a simulation, and the length of an individual time step.

[1]Open Dynamics Engine: `http://www.ode.org/`

[2]ARGoS: `http://www.argos-sim.info/`

`<arena>` specifies the layout of a simulated environment. For example, the number of instances of a given type of object and their distribution.

`<loop_functions>` enables the loading of a library called a *loop function*. This library can be used to monitor and modify a simulation by providing functions for the pre-tick and post-tick event hooks.

`<controllers>` enables the loading of libraries called controllers, which are written in C++ or in Lua. These libraries implement the behavior of objects in a simulation. This subtree also describes the configuration of the sensors and actuators attached to an object.

`<media>` defines one or more *mediums*. A medium is a spatially-hashed index, which contains objects that can be detected by a sensor.

`<physics_engines>` configures the physics engine plugins, which provide either kinematics or dynamics for objects in the simulation.

`<visualization>` selects a visualization for a simulation, which may be interactive or non-interactive, graphical or text-based.

ARGoS uses the `<arena>` subtree to initialize the *space*, a data structure representing the current state of a simulation. This data structure contains groupings of properties and states called *entities*, which are arranged through inheritance and composition to represent objects in a simulation. In addition to the representation of an object in the space, a user must provide a physics model. A physics model is used by a physics engine plugin to simulate the kinematics or dynamics of an object. The default visualization plugin used in ARGoS is based on Qt[3] and OpenGL[4]. It provides an interactive environment where objects rendered in OpenGL can be selected and moved around. This visualization provides an additional plugin layer, which supports customization of the user interface and the appearance of the simulated environment. Plugins for this layer are called *user functions*.

5.2 Extensions to the ARGoS simulator

To run experiments in simulation, we have created several plugins for the ARGoS simulator. These plugins enable us to model the autonomous robot and the stigmergic block inside ARGoS. The plugins provide three-dimensional dynamics and magnetism, a prototyping entity for rapid development and testing, a multi-camera framework for computer vision, and radios for wireless communication. In the following sections, we describe each of these plugins in detail.

[3]Qt: `https://www.qt.io/`

[4]OpenGL: `https://www.opengl.org/`

```xml
<physics_engines>
  <dynamics3d id="dyn3d" iterations="20">
    <floor/>
    <plugins>
      <gravity g="9.8"/>
    </plugins>
  </dynamics3d>
</physics_engines>
```

Listing 5.1: Example configuration of the three-dimensional dynamics plugin.

5.2.1 The three-dimensional dynamics plugin

To simulate the physical interactions between the autonomous robots and stigmergic blocks, we require the simulation of three-dimensional dynamics. To this end, we have developed a physics engine plugin, which is a wrapper around the Bullet physics engine[5]. We have selected the Bullet physics engine as it is open source and actively developed. In this plugin, the physics model of a robot is represented by a class, which maintains a collection of bodies and joints.

The three-dimensional dynamics plugin by default only supports collision-based interactions between bodies. To enable the simulation of gravity and magnetism, we have made the three-dimensional dynamics plugin extendable by implementing an additional plugin layer. This layer enables plugins to apply forces and torques to bodies in the simulation.

Listing 5.1 shows a configuration of the three-dimensional dynamics plugin in the `<physics_engines>` subtree. This configuration instructs the three-dimensional dynamics plugin to advance the physics simulation by 20 steps for every time step of the simulation. In addition, the physics simulation is configured to include a planar floor. Gravity is provided using the plugin layer and is configured with a downwards acceleration of 9.8 m/s^2.

5.2.2 The prototyping plugin

The prototyping plugin was originally developed to test the three-dimensional dynamics plugin but has since developed into a solution to rapidly describe a new object to the simulator, without the need to manually code and compile new classes.

The prototyping plugin defines a prototype entity, which is a collection of body and joint entities. The prototype entity is instantiated and configured by adding a `<prototype>` element to the `<arena>` subtree. The `<prototype>` element is populated with further elements, which in turn configure the controller, the bodies, the joints, and the devices of a prototype entity.

The geometry of a body in a prototype entity can be either a box, a cylinder, or a sphere. The dimensions, positions, and orientations of a body are fully specified in the XML. The prototyping plugin allows for joints between any two bodies and supports

[5]Bullet Physics: `http://bulletphysics.org/`

```xml
<prototype id="magnetic_box">
  <body position="-0.05,-0.1,0.0"  orientation="60,0,0"/>
  <bodies reference_body="base">
    <body id="base" geometry="box" size="0.1,0.1,0.1" mass="0.01">
      <coordinates/>
      <offset/>
    </body>
  </bodies>
  <devices>
    <electromagnets>
      <electromagnet body="base" passive_field="5000,0,0"/>
    </electromagnets>
  </devices>
</prototype>
```

Listing 5.2: Instantiation of a magnetic box in the `<arena>` subtree for the testing of the magnetism plugin.

any combination of linear and angular degrees of freedom about any axis. To enable actuation and sensing in the prototyping plugin, we have implemented generic joint sensors and actuators. A joint sensor enables a controller to measure the translation or rotation of a joint from its initial position and an actuator enables a controller to apply a translational force or a torque to a joint.

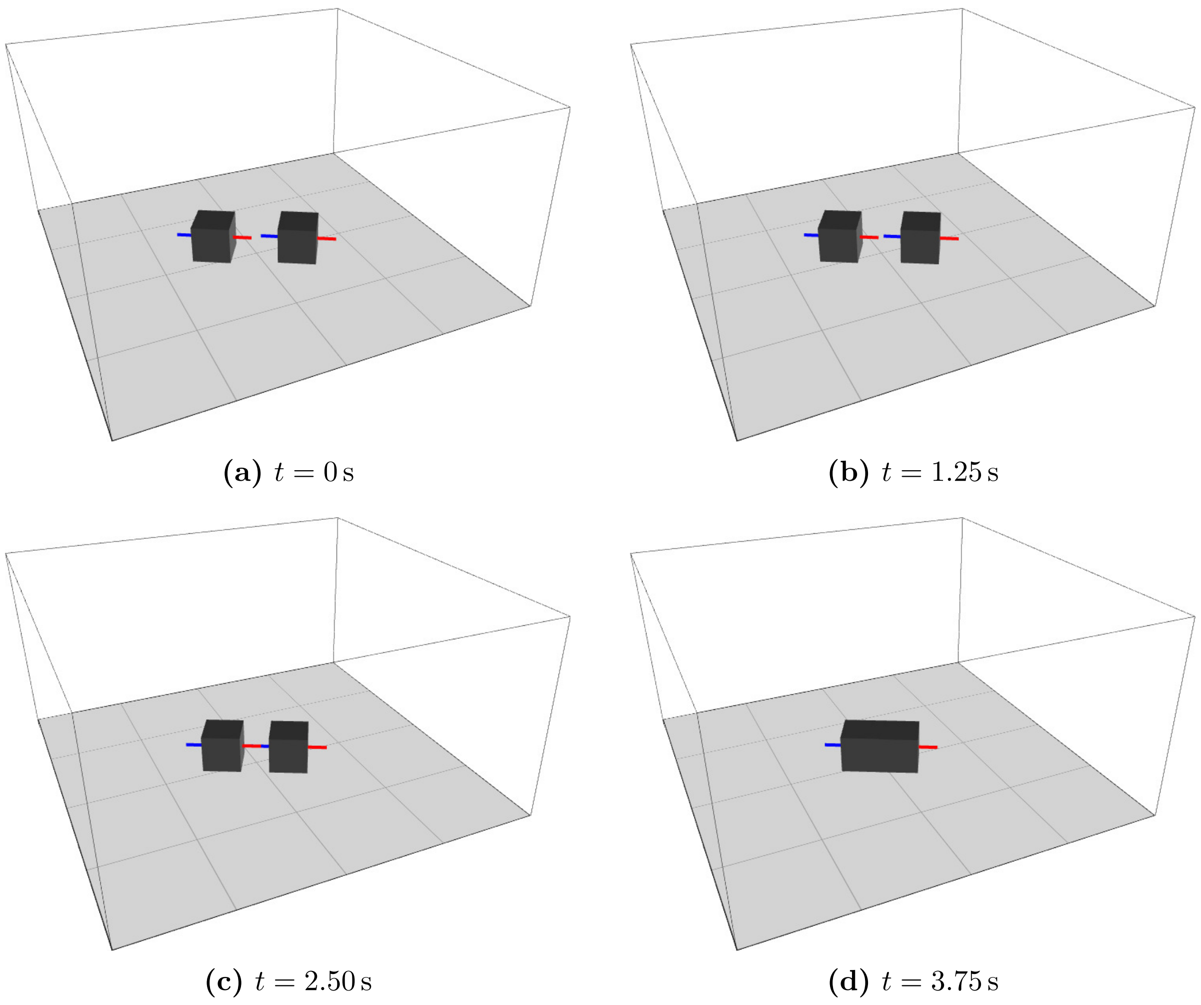
The prototyping plugin provides a generic physics model for the three-dimensional dynamics plugin and a generic visualization model for the Qt-OpenGL visualization plugin. The generic visualization model is useful for debugging as it accurately shows the size, geometry, position, and orientation of the bodies in the three-dimensional physics engine.

5.2.3 The magnetism plugin

An autonomous robot uses four semi-permanent electromagnets to pick up a stigmergic block by coupling with the four freely-rotating spherical magnets in the top face of a block. The spherical magnets inside a block also enable self-alignment and reduce the cumulative misalignment between blocks during construction.

To implement magnetism in ARGoS, we start by defining an entity that represents an electromagnetic object. This entity consists of an electrical current, a passive magnetic field vector and an active magnetic field vector, which is multiplied by the electrical current. This configuration enables the simulation of permanent magnets, electromagnets, and semi-permanent electromagnets. Listing 5.2 shows how a passive magnetic box is modeled using the prototyping plugin.

Listing 5.3 shows the magnetism plugin being loaded via the `<plugins>` subtree of the three-dimensional dynamics plugin. The current implementation of this plugin does not export any configuration options at the time of writing. Figure 5.1 shows two of the magnetic blocks (described in Listing 5.2) being pulled together as a result of the simulated magnetism.



(a) $t = 0\,\mathrm{s}$ **(b)** $t = 1.25\,\mathrm{s}$ **(c)** $t = 2.50\,\mathrm{s}$ **(d)** $t = 3.75\,\mathrm{s}$

Figure 5.1: Demonstration of two of the magnetic blocks (described in Listing 5.2) being pulled together as a result of the simulated magnetism. The red lines represent the north of a dipole and the blue lines represent the south of a dipole.

```xml
<physics_engines>
  <dynamics3d id="dyn3d" iterations="25">
    <plugins>
      <magnetism/>
    </plugins>
  </dynamics3d>
</physics_engines>
```

Listing 5.3: Configuration of the `<physics_engines>` subtree for the testing of the magnetism plugin.

To calculate and apply the magnetic forces, we have implemented the algorithm presented in [100] as a plugin for the three-dimensional dynamics plugin. This algorithm uses finite element analysis (FEA) to calculate the forces and torques on all magnetic elements in the simulation. The total force and torque on a magnetic body are the vector summations of the forces and torques on each of its elements. The current implementation uses a single dipole approximation where each magnetic body is represented by a single FEA element. This approximation, however, is sufficient for the magnetism used in our multi-robot ACS, as the spherical magnets in the stigmergic blocks are similar to the FEA elements used in the algorithm. As a further optimization, we limit the effective distance of the magnetic field to be just short of the distance between two spherical magnets in the corners of a stigmergic block. This optimization is effective as the magnetic forces between the magnetic spheres within the same block are negligible.

5.2.4 The multi-camera framework

In contrast to the existing camera sensors available in ARGoS, the multi-camera framework supports attaching multiple cameras to an object in a simulation. We have designed the multi-camera framework to be compatible with OpenCV[6], the most widely used computer vision library.

To simulate the imperfections of real camera systems, the framework allows a user to specify the intrinsic parameters of a camera (i.e. the principal point and the focal lengths) and its distortion parameters. These parameters are identical to those used in the camera calibration module of OpenCV.

For each simulated camera, it is possible to load a number of simulated computer vision algorithms. These algorithms imitate the output of a computer vision algorithm by simulating feature extraction. Feature extraction is simulated by directly calculating if an object is visible by a camera. This approach is more efficient and uses fewer resources than simulating a real computer vision algorithm, which requires a scene from the perspective of a camera to be rendered to a virtual frame buffer with both accurate lighting and models.

Cameras can be attached to objects in a simulation using the prototyping plugin. The multi-camera framework adds a camera for each `<camera>` element found in the `<cameras>` subtree. Listing 5.4 shows the possible configuration options for each camera. Simulated algorithms are loaded with respect to the configuration in the `<controllers>` subtree (Listing 5.5).

An update of a camera sensor starts by calculating the bounding planes of a frustum, which represents a camera's field of view. This frustum is then used to calculate its enclosing axis-aligned bounding box (AABB). After the camera sensor has precomputed the frustum and the frustum's enclosing AABB, it passes control to each of the simulated computer vision algorithms to update their readings. Figure 5.2 shows an object, which is equipped with a camera. The figure also shows a test pattern, consisting of a tag and eight LEDs.

[6] OpenCV: `http://opencv.org/`

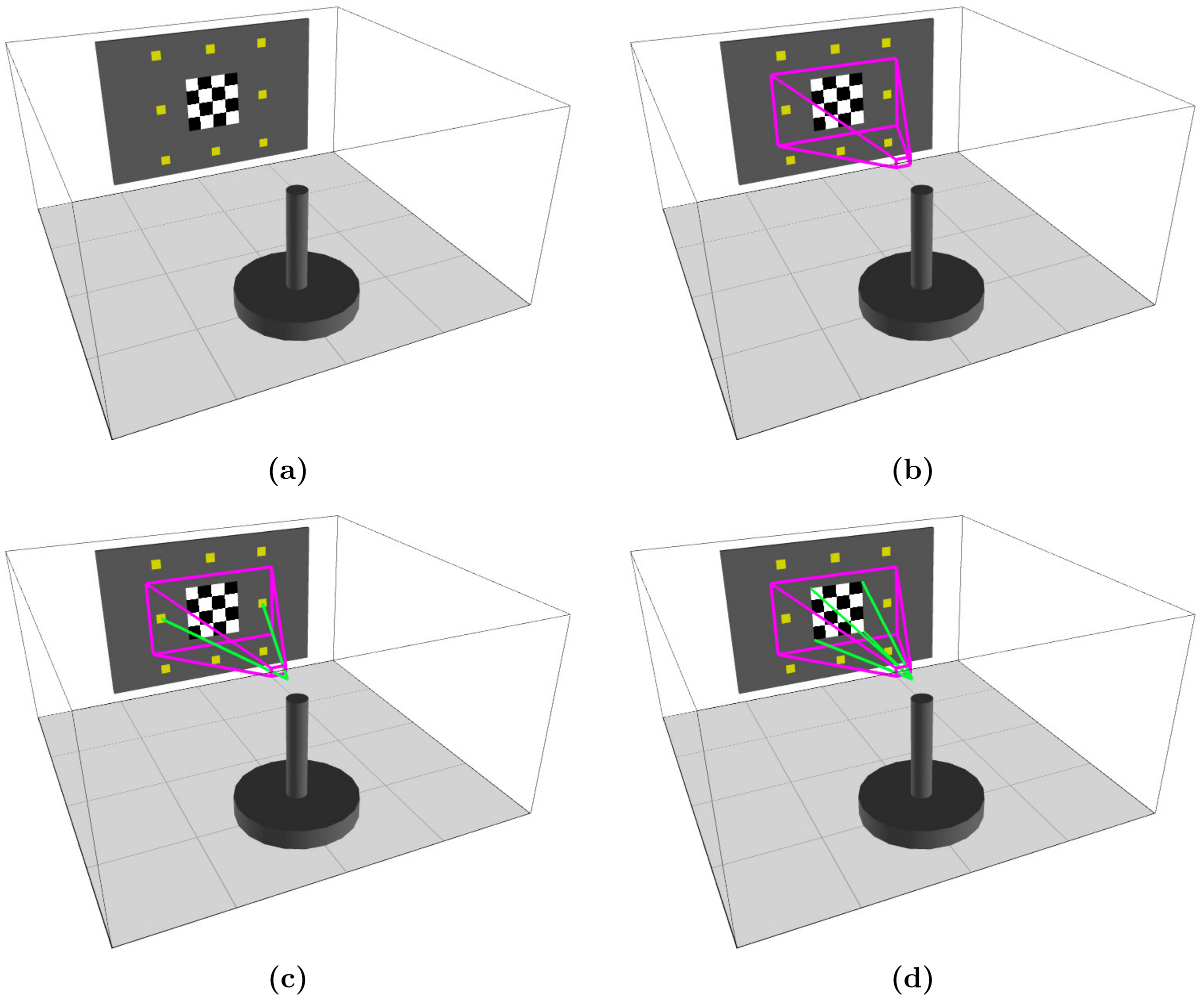


Figure 5.2: Testing the multi-camera framework. (a) The setup of an object with a camera and a test pattern, consisting of a tag and eight LEDs. (b) Rendering the frustum of the camera sensor. (c) Detecting the two LEDs that are inside the frustum. (d) Detecting the four corners of the tag.

```
<devices>
  <cameras>
    <camera id="front_camera" resolution="640,360" enabled="true" body="link_0"
            range="0.05:0.5" position="-0.05,0,0.275" orientation="90,-90,0"
            principle_point="319.5,179.5" focal_length="883.961,883.961"
            distortion_parameters="0,0,0"/>
  </cameras>
</devices>
```

Listing 5.4: Configuration of the prototype entity's `<devices>` subtree for the testing of the multi-camera framework setup in Figure 5.2.

```
<lua_controller id="robot_prototype_lua">
  <actuators/>
  <sensors>
    <cameras implementation="default" show_frustums="true">
      <algorithms>
        <led_detector camera="front_camera" medium="leds" show_rays="true"/>
        <tag_detector camera="front_camera" medium="tags" show_rays="false"/>
      </algorithms>
    </cameras>
  </sensors>
  <params script="src/testing/camera_calibration.lua"/>
</lua_controller>
```

Listing 5.5: Configuration in the prototype entity's `<controllers>` subtree for the testing of the multi-camera framework setup in Figure 5.2.

```
<media>
  <led id="leds" index="grid" grid_size="2,2,2" />
  <tag id="tags" index="grid" grid_size="2,2,2" />
</media>
```

Listing 5.6: Configuration of the prototype entity's `<media>` subtree for the testing of the multi-camera framework setup in Figure 5.2.

A typical algorithm detects a feature by passing the precomputed AABB and a check operation to a *medium*. A medium is an index of spatially-hashed entities and is declared to the simulator using the `<media>` subtree. Listing 5.6 shows the required configuration for the example in Figure 5.2. A medium runs the check operation against every entity selected by the AABB volume. The check operation determines whether an entity is visible to the camera and can include (i) checking if the entity is completely inside the frustum, (ii) checking if the angle of observation between the camera and the entity is within a given range, and (iii) checking if the object is occluded by other objects in the simulation.

Once an algorithm determines that a feature is visible to a camera, it stores a reading, which can be accessed by the controller of an object. The contents of this reading depend on the algorithm and may include the pixel coordinates of an entity in the virtual image, its colors, and other types of data. For example, the tag detection algorithm contains the pixel coordinates of the corners of a tag in the virtual image and a string representing a tag's data payload.

```
<devices>
  <leds medium="leds">
    <led id="led" body="base" position="0,0,0.0205" orientation="0,0,0"
         observable_angle="75" color="black"/>
  </leds>
  <radios>
    <radio id="radio+x" medium="nfc" duplex_mode="half" body="base"
           position="0.05,0,0.0205" orientation="0,0,0" range="0.075"/>
    <radio id="radio-x" medium="nfc" duplex_mode="half" body="base"
           position="-0.05,0,0.0205" orientation="0,0,0" range="0.075"/>
  </radios>
</devices>
```

Listing 5.7: Configuration in the prototype entity's `<devices>` subtree for the testing of the radio plugin.

```
<lua_controller id="radiopad-ctrl">
  <actuators>
    <leds implementation="default" medium="leds"/>
    <radios implementation="default" medium="nfc"/>
  </actuators>
  <sensors>
    <radios implementation="default" medium="nfc"/>
  </sensors>
  <params script="src/testing/radiopad-ctrl.lua"/>
</lua_controller>
```

Listing 5.8: Configuration for the controller of a radio pad in the `<controllers>` subtree. The radio pad uses a Lua controller and contains an LED actuator, a radio actuator, and a radio sensor.

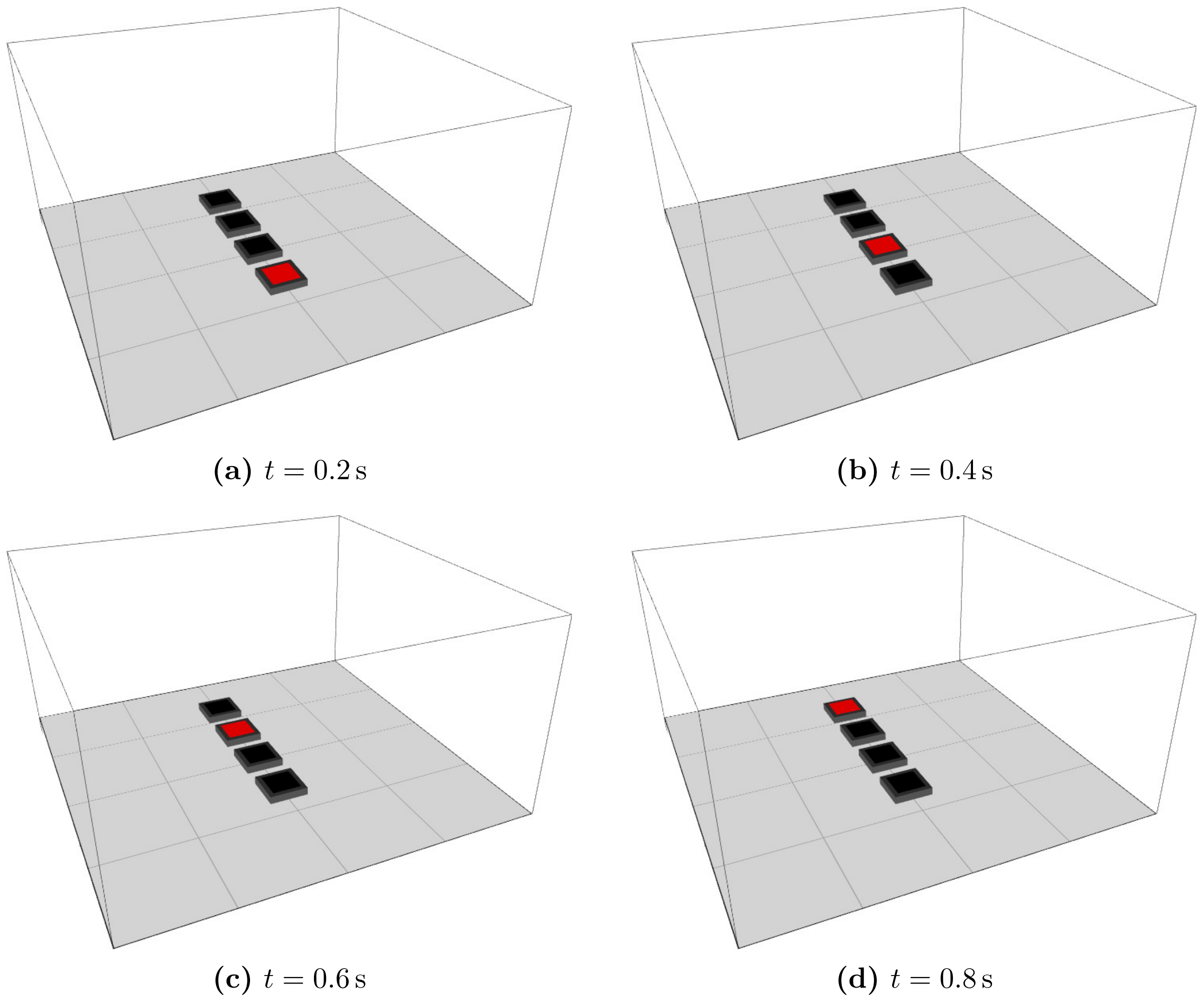
5.2.5 The radio plugin

To realize the NFC communication between a stigmergic block and an autonomous robot, we have implemented a generic radio plugin. This plugin supports attaching multiple radios to an object in the simulation. For example, Listing 5.7 describes the configuration of two radios, which have been attached to an example *radio pad* object using the prototyping plugin. In this example, the radios are set up in half-duplex mode, meaning that they can not receive messages while transmitting.

Listing 5.8 shows the configuration of a Lua-based controller for a radio pad. A radio actuator is used to transmit messages to nearby radios, while a radio sensor is used to receive messages. Listing 5.9 shows the source code of a Lua controller for a radio pad. This controller waits for a message to be received on either of its two radios. Upon receiving a message, it sets its LED to red for the current time step of the simulation and sends a message out of its `radio-x` radio.

Figure 5.3 shows several radio pads, which are arranged to form a line. By inserting a fake message at the beginning of the simulation, the line of radio pads illuminates as the message is passed along from radio pad to radio pad.

A transmission between two objects occurs as follows. The controller on the transmitting object writes one or more messages using its radio actuator, each message containing



(a) $t = 0.2\,\mathrm{s}$ **(b)** $t = 0.4\,\mathrm{s}$ **(c)** $t = 0.6\,\mathrm{s}$ **(d)** $t = 0.8\,\mathrm{s}$

Figure 5.3: Test setup for the radio plugin. Four radio pads, each with two radios, are arranged into a line. A message is passed down the line of radio pads as indicated by the red LED.

an arbitrary number of bytes. During an update of the simulation, the radio medium transfers the messages from all transmitting radios to all receiving radios within a configurable transmission range. This medium must be declared to the simulator using the `<media>` subtree. Listing 5.10 shows the required configuration of this subtree for the example in Figure 5.3. A receiving radio is defined as a radio that is not transmitting or that is in full-duplex mode. Following a transmission, the transmitted messages are available to the controller on a receiving object via its radio sensor.

5.2.6 User and loop functions

To ease running experiments in the simulator, we have implemented a loop function for the simulation and a user function for the Qt-OpenGL visualization plugin.

```lua
function init()
end

function step()
   local rx = false;
   for radio,data in pairs(robot.radio_rx) do
      if (#data >= 1) then
         rx = true;
         robot.leds.set_all_colors("red");
         robot.radio_tx.send("radio-x", {1});
      end
   end
   if(rx == false) then
      robot.leds.set_all_colors("black");
   end
end

function reset()
end

function destroy()
end
```

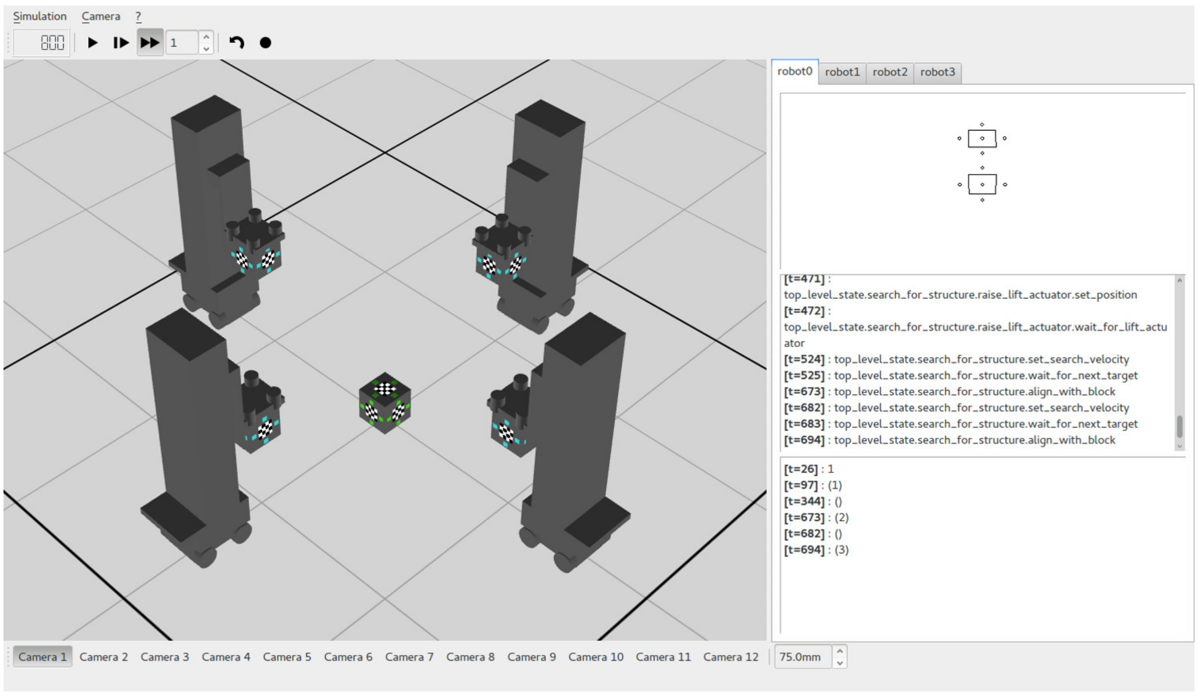
Listing 5.9: The Lua controller for a radio pad. This controller sends a message using the `radio-x` radio and sets its LED to red if a message is received. Otherwise, the LED is switched off.

```xml
<media>
  <radio id="nfc" index="grid" grid_size="2,2,2" />
</media>
```

Listing 5.10: Configuration in the `<media>` subtree for the testing of the radio plugin. A single radio medium with an identifier `nfc` is declared.

Instead of using the `<arena>` subtree to set up the experiment, the definitions for the stigmergic block and autonomous robot are placed in the `<loop_functions>` subtree. These definitions are then parsed by the implemented loop function and stored in a map. This approach allows blocks and robots to be added and removed from the simulation programmatically. For example, the loop function defines a set of block cache locations from where the robots can retrieve blocks during an experiment. These locations are automatically monitored and populated with a block when no other blocks are in the vicinity of the location.

The user function modifies the Qt-OpenGL visualization plugin by adding a `QTabWidget` to the user interface, consisting of a tabbed pane for each of the robots in the simulation (Figure 5.4). These panes include a preview window, which displays the extracted features from the simulated computer vision algorithms. In addition, the logs from a robot's controller can be displayed to assist with debugging and development. When a robot is selected in the simulation, a Qt signal is emitted that is captured by a slot in the `QTabWidget`. The function associated with this slot automatically selects and displays the pane associated with the selected robot.



```
[t=471] :
top_level_state.search_for_structure.raise_lift_actuator.set_position
[t=472] :
top_level_state.search_for_structure.raise_lift_actuator.wait_for_lift_actu
ator
[t=524] : top_level_state.search_for_structure.set_search_velocity
[t=525] : top_level_state.search_for_structure.wait_for_next_target
[t=673] : top_level_state.search_for_structure.align_with_block
[t=682] : top_level_state.search_for_structure.set_search_velocity
[t=683] : top_level_state.search_for_structure.wait_for_next_target
[t=694] : top_level_state.search_for_structure.align_with_block
```

```
[t=26] : 1
[t=97] : (1)
[t=344] : ()
[t=673] : (2)
[t=682] : ()
[t=694] : (3)
```

Figure 5.4: The Qt-OpenGL visualization plugin customized with a user function.

5.3 Modeling hardware in simulation

5.3.1 The stigmergic block

A stigmergic block is modeled using the prototyping plugin (Section 5.2.2). It consists of nine bodies: the main body of the block and its eight spherical magnets. We have added a joint between the main body of the block and each of its spherical magnets to enable unconstrained rotation around the X, Y, and Z axes. Figure 5.5 shows the completed model of a block, which is equipped with four rectangular LEDs, a tag, and an NFC transceiver (provided by the radio plugin) on each of its faces. The complete XML description of the stigmergic block for the prototyping plugin is provided in Appendix A. The software on a stigmergic block is simulated using the Lua script in Listing 5.11, which configures the colors of the LEDs on a block in response to receiving an NFC message from an autonomous robot.

The dimensions and weight of the simulated model for a stigmergic block are equal to those of the hardware. As we use a single dipole approximation for magnetism, we have tuned the strength of the simulated spherical magnets empirically. We observed from the real hardware that placing two blocks approximately one centimeter apart on a low-friction surface was sufficient for the spherical magnets inside the two blocks to turn, align, and pull the blocks together. We set up a similar configuration in simulation and adjusted the magnetic field strength of the magnetic spheres until we observed a similar response.

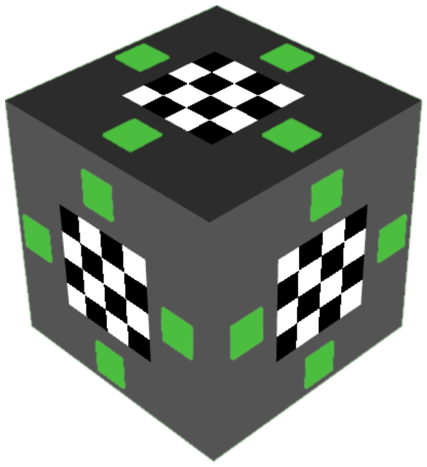


Figure 5.5: The visualization of a stigmergic block. A checkerboard pattern represents a tag and a green square represents an LED. This visualization is provided automatically by the Qt-OpenGL model for the prototyping plugin (Section 5.2.2).

```lua
local lookup_table = {
   ["0"] = {r = 0, g = 0, b = 0},
   ["1"] = {r = 255, g = 120, b = 255},
   ["2"] = {r = 255, g = 208, b = 27},
   ["3"] = {r = 74, g = 255, b = 27},
   ["4"] = {r = 74, g = 255, b = 255},
}

function init()
   reset();
end

function reset()
   local setting = lookup_table["0"];
   robot.leds.set_all_colors(setting.r,setting.g,setting.b);
end

function step()
   -- for each radio on the block
   for radio,data in pairs(robot.radio_rx) do
      -- if data was received
      if (#data >= 1) then
         -- configure the LEDs
         local setting = lookup_table[string.char(data[1][1])];
         robot.leds.set_all_colors(setting.r,setting.g,setting.b);
      end
   end
end
```

Listing 5.11: The Lua controller for a simulated stigmergic block. The color values in the `lookup_table` correspond to the OFF, Q1, Q2, Q3 and Q4 colors used on the hardware.

One difference between the hardware of the stigmergic block and its simulated model is that the hardware has curved edges, while the simulated model is completely cubic. While it is possible to simulate the geometry of a stigmergic block more accurately using meshes, the collision detection for meshes requires more CPU time. Furthermore, we have found from empirical testing that the differences in edge geometry do not play a significant role in the dynamics of the system. In addition, the simulated model of a stigmergic block assumes that the center of mass is at the center of a block. To be precise, the location of the battery and the central circuit board offset the center of mass slightly away from the center of a block. Due to the nature of our experiments, however, we assume that this difference between the hardware and its simulated model is negligible.

5.3.2 The autonomous robot

Similar to the stigmergic block, we model the autonomous robot using the prototyping plugin (Figure 5.6). The model of the robot contains 15 bodies and 15 joints. Four of these joints have a single rotational degree of freedom and allow the robot's wheels to turn. An additional joint has a single translational degree of freedom, allowing the end-effector of the robot to move up and down. The remaining joints have no degrees of freedom and are used to fix the bodies of an autonomous robot in place. Geometrically, the model is very similar to the hardware. We have, however, merged some of the simulated bodies to improve collision detection performance in the three-dimensional dynamics plugin.

To simulate the computer vision on the autonomous robot, we attach a camera sensor to the body of the end-effector in the model. This camera sensor is provided by the multi-camera framework (Section 5.2.4). To configure the camera sensor, we use the intrinsic and distortion parameters from the camera used on the real robot, which we obtained from the camera calibration module of OpenCV. This calibration process involves using the real camera on an autonomous robot to take a number of photos of an asymmetrical circle pattern (Figure 5.7).

We have added twelve proximity sensors around the body of the mobile robotics platform and four proximity sensors to the body of the manipulator. The simulated proximity sensors are calibrated based on the datasheets for the hardware sensors with some minor adjustments based on empirical measurements. As with the stigmergic block, the radio plugin (Section 5.2.5) is used to simulate the NFC transceiver on an autonomous robot. The plugin is configured to operate in half-duplex mode with a range of two centimeters, which matches the performance of the NFC transceiver on the hardware. The complete XML description of an autonomous robot for the prototyping plugin is provided in Appendix B.

An autonomous robot in simulation is controlled using a slightly modified version of the state machine used by the hardware. These modifications are required due to the use of global data in the state machine used by the hardware, for example, the `std::cout` and `std::cerr` output streams. Since the ARGoS executable and its libraries

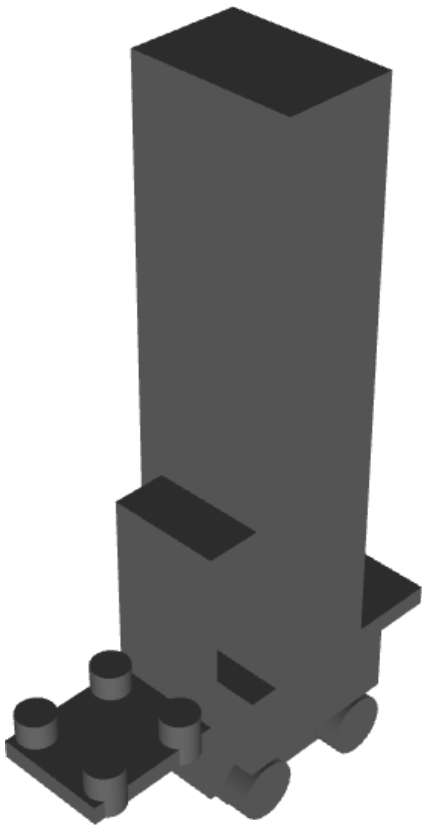


Figure 5.6: The visualization of an autonomous robot. This visualization is provided automatically by the Qt-OpenGL model for the prototyping plugin (Section 5.2.2).

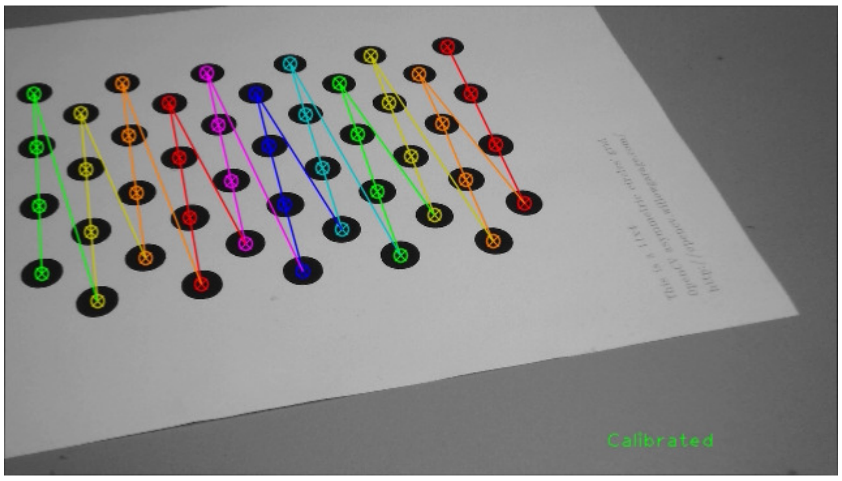


Figure 5.7: A photo of the OpenCV asymmetrical circle pattern taken by the robot. The image has been annotated by the calibration process, which combines several perspectives of this pattern to solve for the camera's intrinsic and distortion parameters.

create an instance of the state machine for each autonomous robot, this global data is shared among each of those instances. In the case of the `std::cout` and `std::cerr` output streams, we resolve this issue by creating separate output streams based on `std::stringstream` for each autonomous robot.

The timeout transitions in the state machine used by the hardware also require modification. This modification is required due to the use the C++11 Chrono library, which provides facilities for working with wall time. The use of wall time in simulation is problematic as a simulation may be paused or may run at different speeds depending on the complexity of the simulation. To this end, adjustments were made to the state machine to use a time point instead of the *now* function from one of the C++11 Chrono library's clock classes. This time point is initialized to the default epoch and is incremented at each time step of the simulation.

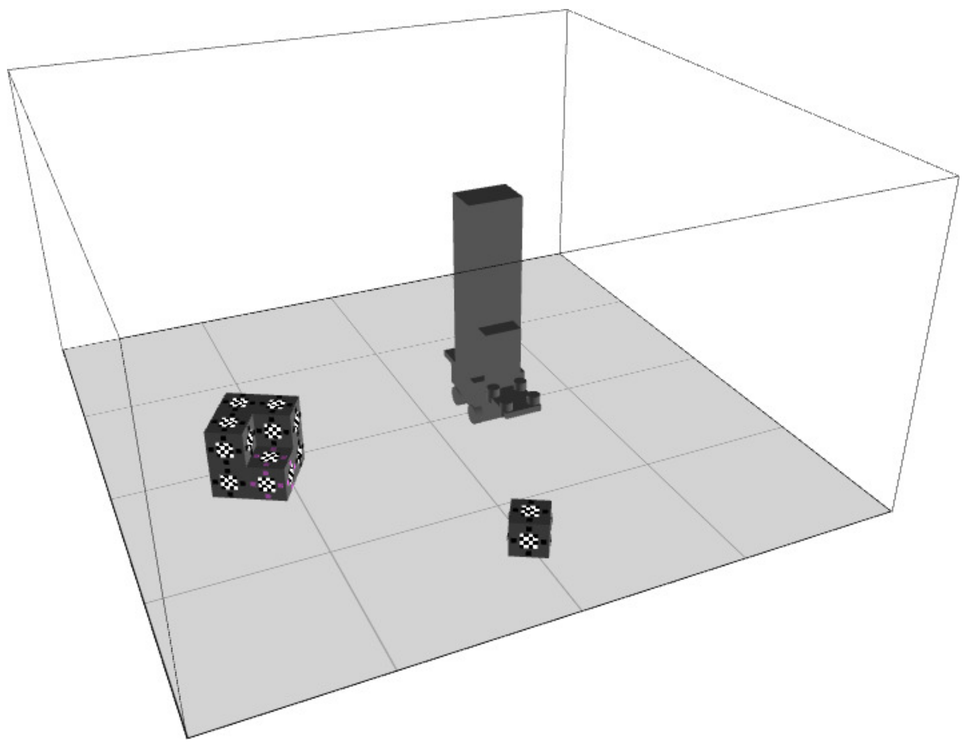
5.4 Verification of the simulation

To verify the plugins and the models of the stigmergic block and the autonomous robot, we have set up two experiments. These experiments are identical to those presented in Section 4.3, which demonstrated how an autonomous robot was able to place stigmergic blocks into a structure at different locations with respect to patterns in the immediate environment. These patterns were expressed in terms of the structural arrangement of stigmergic blocks and their LED markings.

As discussed in Section 4.3.1, the first task is designed to demonstrate that an autonomous robot can perform an action in response to a markings-based stimulus. In this task, an autonomous robot must pick up an unused block and place it on top of the block that is located inside a structure and has its LEDs illuminated. Figure 5.8 shows two snapshots of this experiment in simulation. The first snapshot shows the setup of the task and the second snapshot shows the task after it has been completed.

As described in Section 4.3.2, the second task requires the robot to estimate the size of two structures using a heuristic. Upon identifying the larger of the two structures, the robot extends the larger structure by attaching an unused block to the front of the leftmost block in the structure. The snapshots in Figure 5.9 shows the task setup and the task after it has been completed.

Due to the straightforward nature of the tasks, the outcome of these two experiments in simulation is practically deterministic. To this end, we have not run trials or undertaken any statistical analysis of the performance of these tasks in simulation.


(a) Initial setup

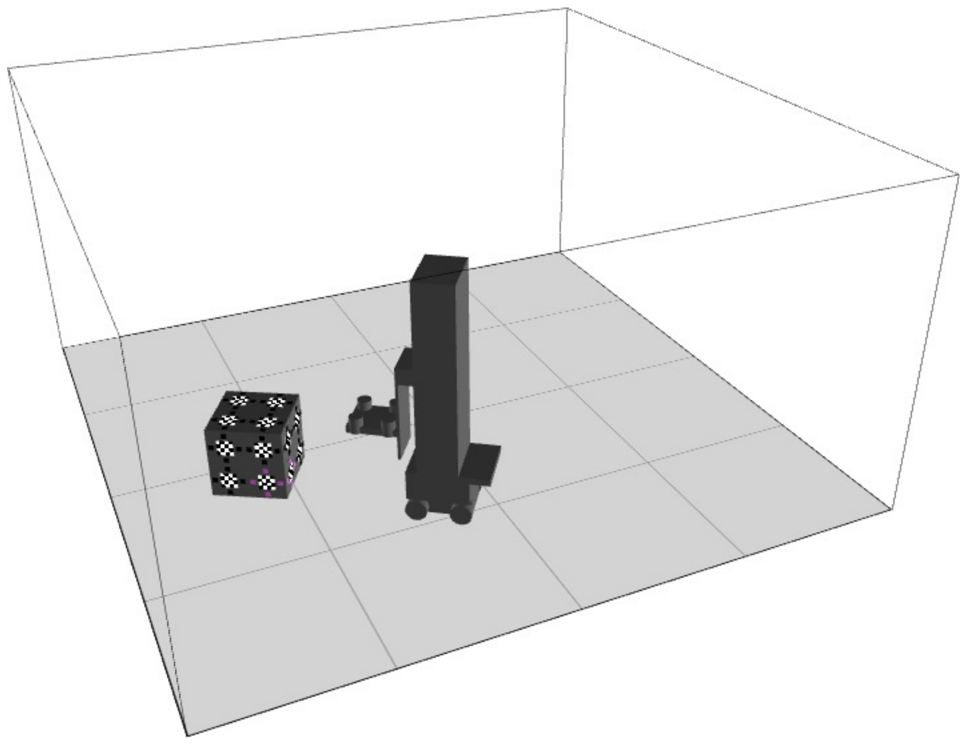
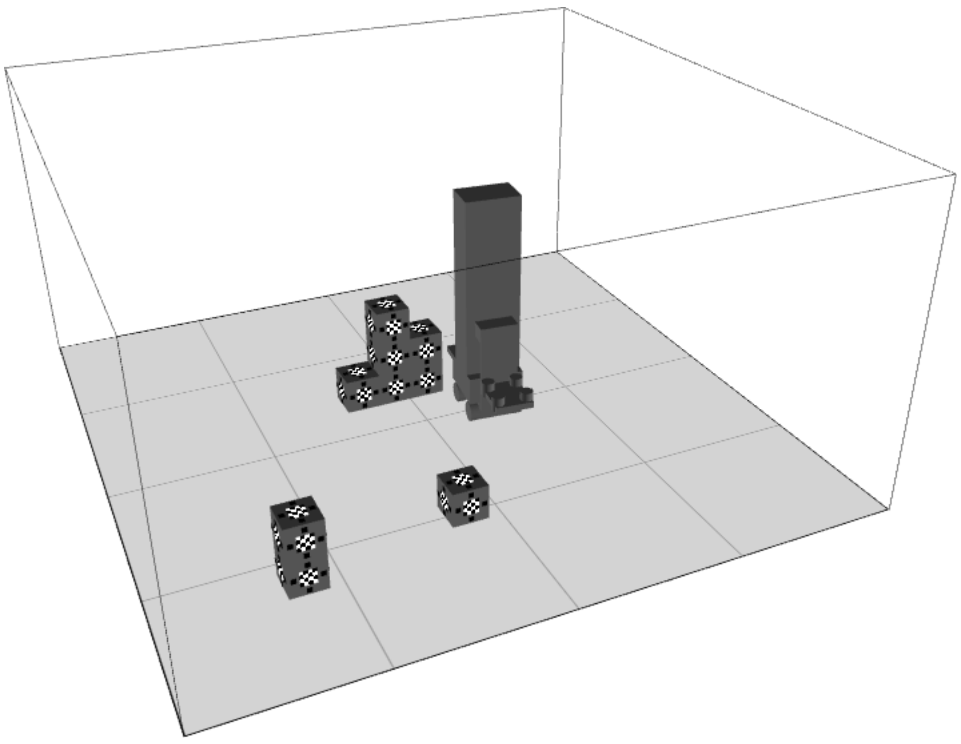

(b) Task complete

Figure 5.8: Markings-based verification task.


(a) Initial setup

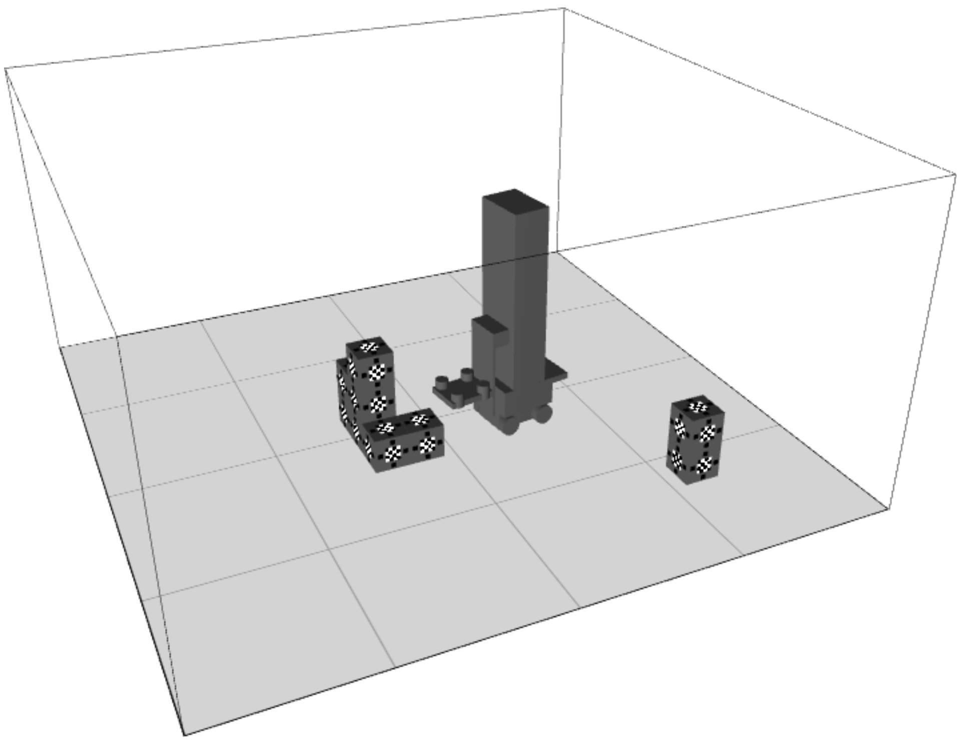

(b) Task complete (rotated view)

Figure 5.9: Structure-based verification task.

CHAPTER 6

Results and Demonstrations

In this chapter, we demonstrate that our decentralized control strategy can be used to coordinate construction. Using the hardware implementation of our multi-robot ACS, we show how a single autonomous robot can assemble a staircase from stigmergic blocks and how two autonomous robots can cooperate to build a column.

We then repeat the staircase demonstration in simulation to show that this implementation of our multi-robot ACS is capable of replicating the results from the hardware implementation. In addition, we present a multi-robot construction scenario involving four autonomous robots that cooperate to build a stepped pyramid in simulation.

In each of these construction scenarios, the autonomous robots do not maintain an internal representation of the environment nor do they have a notion of the structure being built. Instead, the robots perform construction in response to the structural arrangement of the stigmergic blocks in the environment and to their LED markings.

6.1 Demonstrations using hardware

6.1.1 Setup and configuration

Demonstrations of the hardware implementation of our multi-robot ACS and its decentralized control strategy were performed at the Heinz Nixdorf Institute using the Teleworkbench (Figure 6.1). The Teleworkbench is a testing and validation tool for single-robot and multi-robot systems, which can run experiments remotely over the Internet [96]. The work presented in this section, however, does not utilize most of the features of the Teleworkbench other than the raised surface and the illuminated work area. This illumination provides consistent lighting, which improves the performance of the computer vision algorithms running on the autonomous robots.

To communicate with the autonomous robots during an experiment, we configure the robots to connect automatically to a TP-Link WR1043ND wireless router after booting. We have replaced the default operating system on this router with OpenWRT[1], a distribution of Linux for embedded devices. Running Linux on the wireless router allows us to enhance the network of the autonomous robots through the use of scripts and standard POSIX tools.

[1]OpenWRT: `https://openwrt.org/`

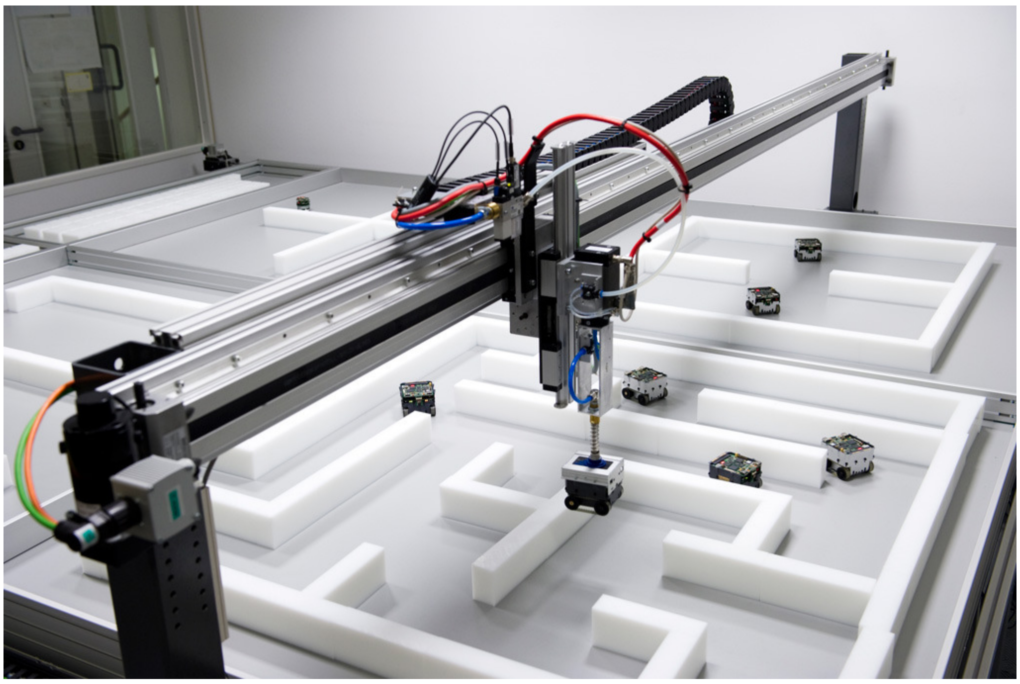


Figure 6.1: The teleworkbench repositioning a BeBot (reprinted from [97] with permission from Christoph Scheytt, © 2017, Heinz Nixdorf Institute).

To start an experiment using one or more autonomous robots in this setup, we launch a Bourne-again shell (Bash) script to search the wireless network for and to connect to the autonomous robots. This script functions as follows.

1. A workstation on the same network as the autonomous robots launches the script from a Bash shell.
2. The script uses SSH to connect to the wireless router running OpenWRT and to read the file `/tmp/dhcp.leases`. This file contains, among other data, the mappings of media access control (MAC) addresses to internet protocol (IP) addresses.
3. Since the WiFi chips on the autonomous robot are manufactured by the same vendor (Wi2Wi), we can determine which devices on the network are robots by checking the first three bytes of each device's MAC address. For each MAC address matching the vendor's identifier, we store the associated IP address.
4. For each of these IP addresses, we then determine if the robot associated with the given address is online by sending internet control message protocol (ICMP) packets using the `ping` utility.
5. The script then starts `tmux`, a terminal multiplexer that creates virtual terminals that can be arranged, displayed, and managed inside the existing terminal.
6. For each online robot, the script requests `tmux` to create a new virtual terminal and to start an SSH connection to the robot inside of it.

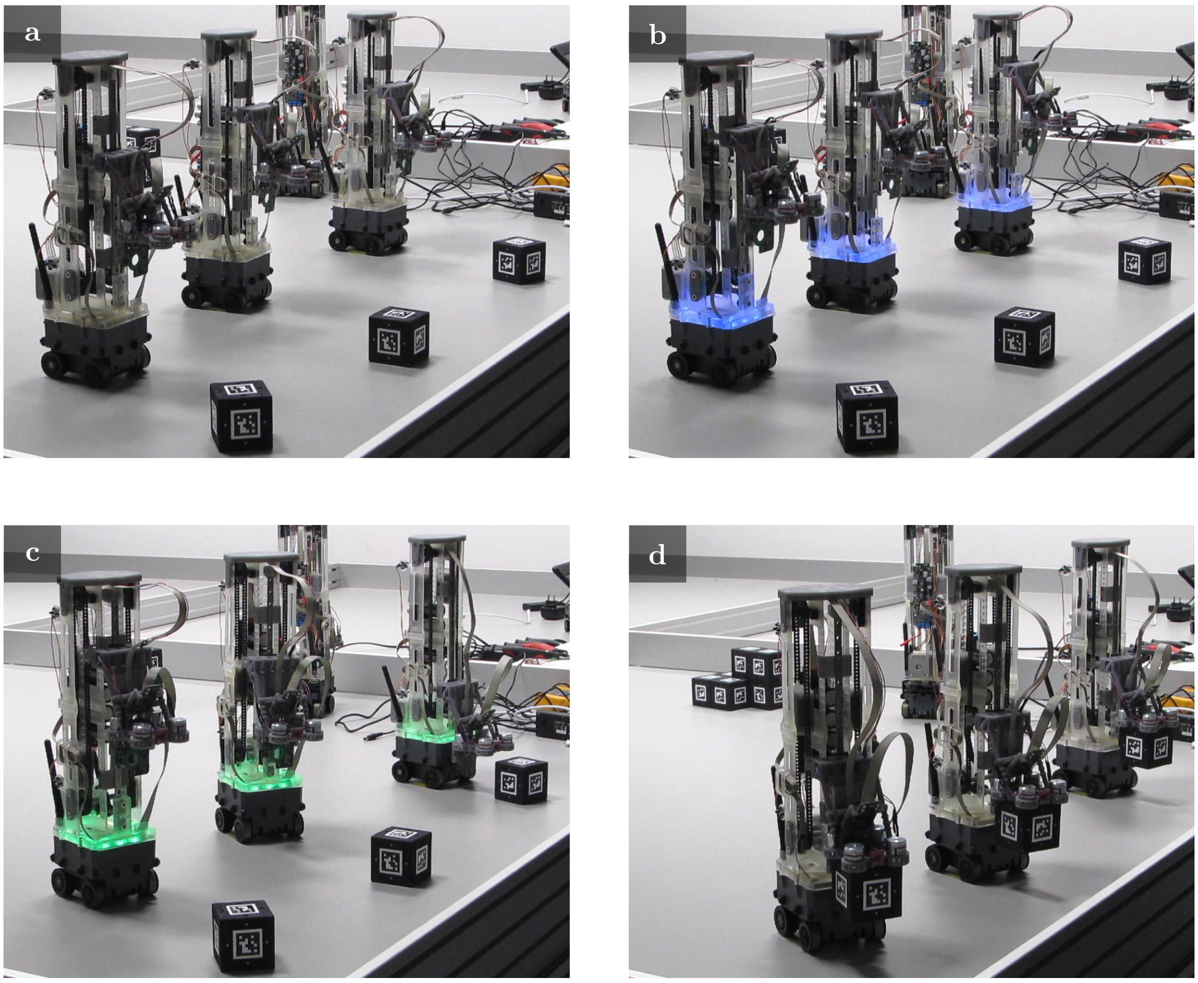


Figure 6.2: Three robots performing an unused block search and pick-up test. (a) Initialization, (b) searching for an unused block, (c) approaching an unused block, (d) picking up an unused block.

7. The script configures several key bindings for `tmux` to automate tasks such as updating the blocktracker executable or checking the status of the power management circuitry.
8. The script enters the interactive interface of `tmux` where all the terminals for the connected robots are displayed as panes.

For our setup, the most important key binding enables pane synchronization. In pane synchronization mode, all keystrokes are sent to all panes, which in our case are the SSH sessions for the connected robots. With pane synchronization mode enabled, we can quickly send an interrupt signal to all connected robots. If the blocktracker executable is running on a robot, this interrupt signal is caught by a signal handler that powers down the controllers for the differential drive system, the stepper motor, and the electromagnet precharge circuitry of an autonomous robot. This mechanism enables an emergency stop, which can prevent or at least mitigate damage to the hardware. Other key bindings include single-line commands, which can download the latest version of the blocktracker executable from an HTTP server or can start a program such as the test routine running in Figure 6.2.

An experiment is started by loading the blocktracker executable. As discussed in Section 4.2.3, the blocktracker executable starts by initializing the robot's hardware. Following initialization, the robot searches for an unused block by turning on the spot. After locating an unused block, the robot approaches the unused block and picks it up using the semi-permanent electromagnets in its end-effector. The robot then continues to turn on the spot, searching for a partially-built structure. Once a structure is found, the robot approaches it before configuring the unused block's LEDs and attaching the unused block to the structure. The configuration of the unused block's LEDs and its attachment are performed with respect to the structural arrangement and LED markings of the partially-built structure.

During this sequence of operations, a robot indicates its internal state using its LEDs. For example, when a robot is searching for a target (an unused block or a structure), the LEDs are blue. Furthermore, when the robot is approaching a target using its computer vision, the LEDs are green, and when it is approaching or aligning with a target without computer vision, the LEDs are red. The LEDs on an autonomous robot are switched off during initialization or when it is shutdown.

6.1.2 Construction of a staircase

Objective In this demonstration, we aim to show how our decentralized control strategy can be used to construct a staircase using a single autonomous robot. The staircase consists of three columns descending in height, with each column using a different type of block. The highest column contains three Q3 (green) blocks, the middle column contains two Q2 (red) blocks, and the last column contains a single Q1 (violet) block.

The high-level state chart in Figure 6.3 shows the target behavior for the robot as it cycles between retrieving an unused block and attaching it to the partially-built staircase. This state chart also shows the error handling transitions for timeouts and for when the tracking of a target block or structure is lost.

The logic in Algorithm 6.1 demonstrates how an autonomous robot performs construction without maintaining an internal representation of the environment or the state of the partially-built staircase. Instead, the robot's construction behavior is coordinated using a feedback loop. In this feedback loop, previous construction by the robot modifies the partially-built staircase, and a modification to the partially-built staircase causes the robot to perform further construction.

Setup and procedure The demonstration is set up with a seed block (the bottom block in the highest column), an autonomous robot, and an unused block. On the first run, the robot autonomously searches, retrieves, and attaches the unused block to the

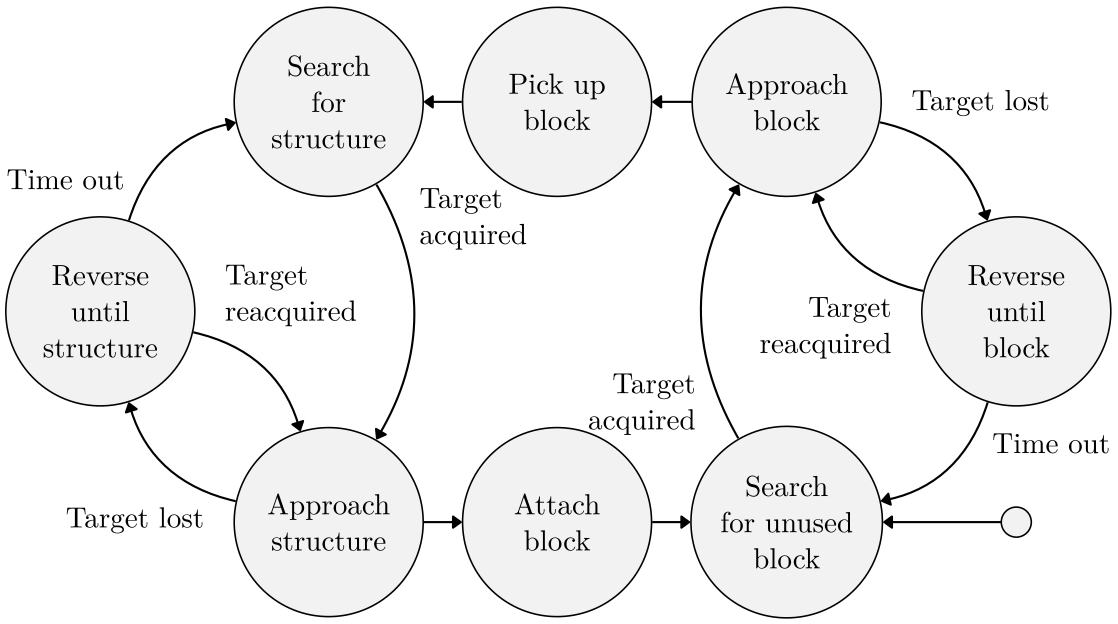


Figure 6.3: High-level state chart for the staircase demonstration.

seed block. After this first run, however, we intervene and manually attach the next unused block directly to the robot. This intervention is required to mitigate the issues with the drive system and allows us to complete this demonstration.

Results Figure 6.4 shows six snapshots of the robot building a staircase. Due to the aforementioned issues with the drive system, this demonstration was assembled from multiple runs. In each of these runs, however, the autonomous robot was able to correctly identify the structural arrangement of the stigmergic blocks in the environment and their LED markings. Furthermore, the autonomous robot was able to correctly attach the unused blocks to the partially-built staircase.

6.1.3 Multi-robot construction of a column

Objective In this demonstration, a collision avoidance mechanism is utilized to prevent two robots simultaneously adding blocks to the same construction site. This collision avoidance mechanism works by reserving the Q4 (cyan) color to indicate that an unused block is being transported. That is, as a robot picks up an unused block, it should configure the color of the block's LEDs to Q4. After locating and approaching a structure, the robot should configure the block's LEDs with respect to the structural arrangement and LED markings of the structure, before attaching its block.

If a robot detects a block with Q4 LEDs while approaching a structure, it should abort its approach and track the Q4 block until it has disappeared (i.e. the other robot has reconfigured its block's LEDs). During tracking the Q4 block, the robot should

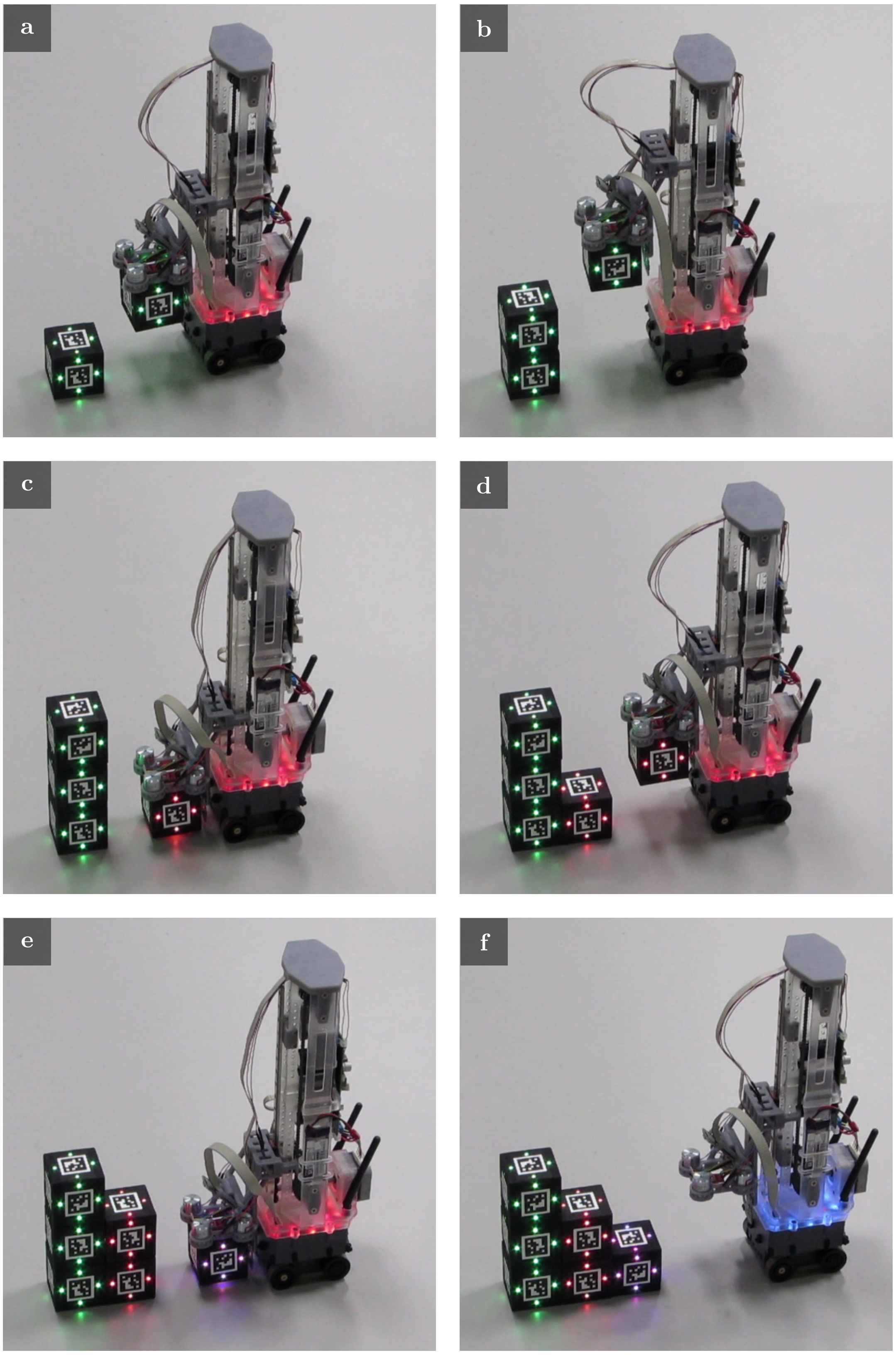


Figure 6.4: Construction of a staircase using a single autonomous robot.

```
set target_block to highest block in frontmost_column;
switch target_block.type do
    case Q3 do
        if target_block.height $< 3$ then
            set unused_block.type to Q3;
            stack unused_block on target_block;
        else
            set unused_block.type to Q2;
            extend frontmost_column with unused_block;
        end
    end
    case Q2 do
        if target_block.height $< 2$ then
            set unused_block.type to Q2;
            stack unused_block on target_block;
        else
            set unused_block.type to Q1;
            extend frontmost_column with unused_block;
        end
    end
    case Q1 do
        shut down;
    end
end
```

Algorithm 6.1: Algorithmic description of the *attach block* state from the high-level state chart in Figure 6.3.

disable its block's LEDs to avoid a deadlock scenario where two robots are waiting for each other indefinitely. Once the tracked Q4 block has disappeared, the robot should restart its approach and attach its own block to the structure.

Setup and procedure This demonstration is set up with two robots, a seed block, and two unused blocks. The target structure is a column, consisting of three Q3 (green) blocks. The robots locate and retrieve the two unused blocks from the environment before approaching the seed block. The robot that arrives at the structure second should give way to the first robot while it is placing its block.

Results Figure 6.5 shows the robots completing this task. While both robots managed to retrieve the unused blocks from the environment and attach them to the structure, successfully avoiding a collision, the second robot (on the right) misinterpreted the structural arrangement of the stigmergic blocks, causing it to extend the partially-built column with a Q2 block, forming a small step instead of a column.

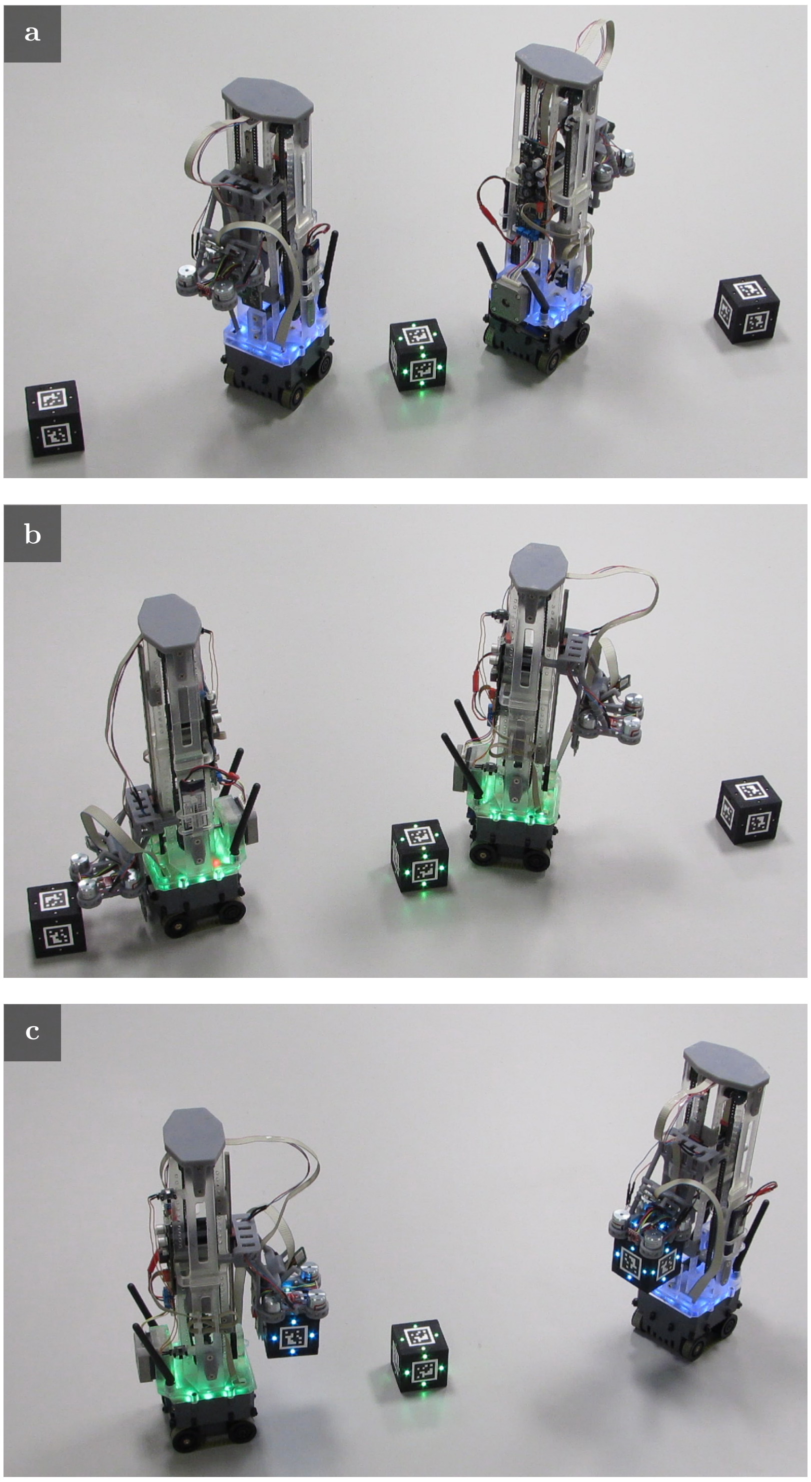


Figure 6.5: Multi-robot construction of a column. (a) Two robots are separated by a seed block. (b) The robots find and retrieve the unused blocks on the left and the right. (c) Both robots approach the seed block attempting to attach their block.

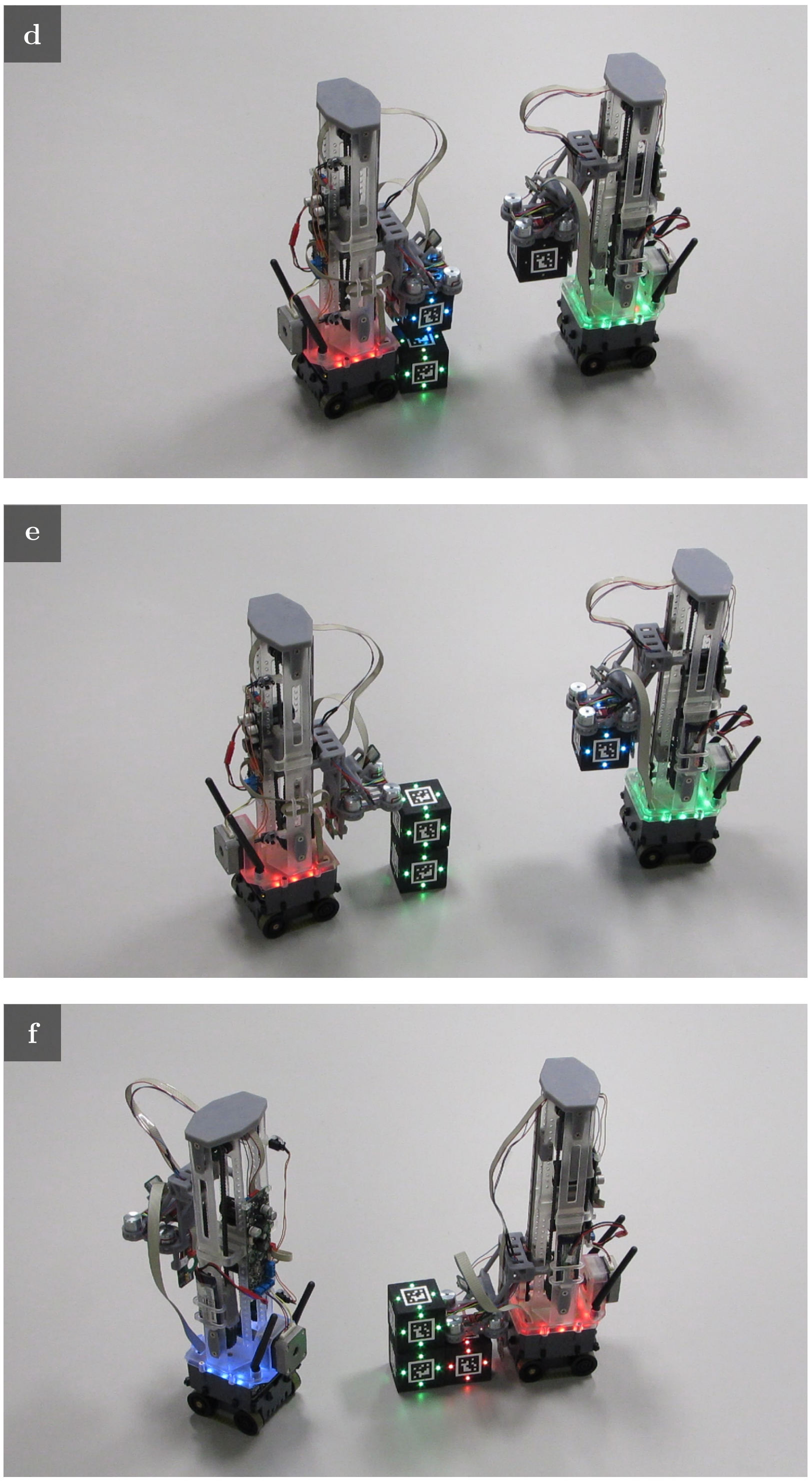


Figure 6.5: (d) The robot on the left reaches the construction site first and the robot on the right gives way, switching its block's LEDs off. (e) The robot on the left finishes attaching its block and the robot on the right restarts its approach, switching its block's LEDs on. (f) The robot on the right attaches its block to the structure.

6.2 Demonstrations in simulation

6.2.1 Setup and configuration

As discussed in Chapter 5, the simulations of our multi-robot ACS are performed using the ARGoS simulator. To run the simulator, we use a Fujitsu Laptop with Intel Core i7-2640M CPU running Ubuntu Linux 16.04 LTS. In addition to the standard ARGoS dependencies [76], the OpenCV computer vision library is installed to provide the `solvePnP` function, which calculates the pose of an object using three-dimensional to two-dimensional point correspondences. An autonomous robot uses this function to calculate the three-dimensional pose of a tag on a stigmergic block.

The ARGoS simulator is configured to use the Qt-OpenGL user function and the loop function described in Section 5.2.6. The controller of a robot is configured to send messages to two logs. The first log displays the current state of the autonomous robot's finite state machine and the second log displays the output of the stigmergic block tracking algorithm. Instead of using the `<arena>` subtree to set up an experiment, the definitions for the stigmergic block and autonomous robot are parsed and stored inside the provided loop function. This configuration enables the adding of a block or a robot to the simulation programmatically. This functionality is used to create block cache locations, which are automatically populated with a stigmergic block after a previous block has been removed from the vicinity of a cache location.

6.2.2 Construction of a staircase

Objective In this demonstration, we aim to show that the simulation of our multi-robot ACS is accurate by replicating the staircase construction demonstration that was performed using the hardware implementation (Section 6.1.2).

Setup and procedure The configuration file for ARGoS provides the definitions of the autonomous robot and the stigmergic block to our loop function, which adds these objects to the arena. We configure the loop function to set this demonstration up identically to the hardware experiment with a seed block, an autonomous robot, and an unused block. We define a single cache location, which adds an unused block to the simulation after the previous unused block has been removed by the robot. There is one significant difference in the setup and procedure of this demonstration. Due to the issues with the drive system not applying to the simulated model of the autonomous robot, invention during experiments is not necessary. To this end, the robot is able to complete the experiment by continuously retrieving blocks from the cache location and transferring them to the partially-built structure.

Results Figure 6.6 shows six snapshots of the autonomous robot successfully constructing the staircase in a simulation without intervention. Since this demonstration is deterministic, we do not provide any statistical analysis.

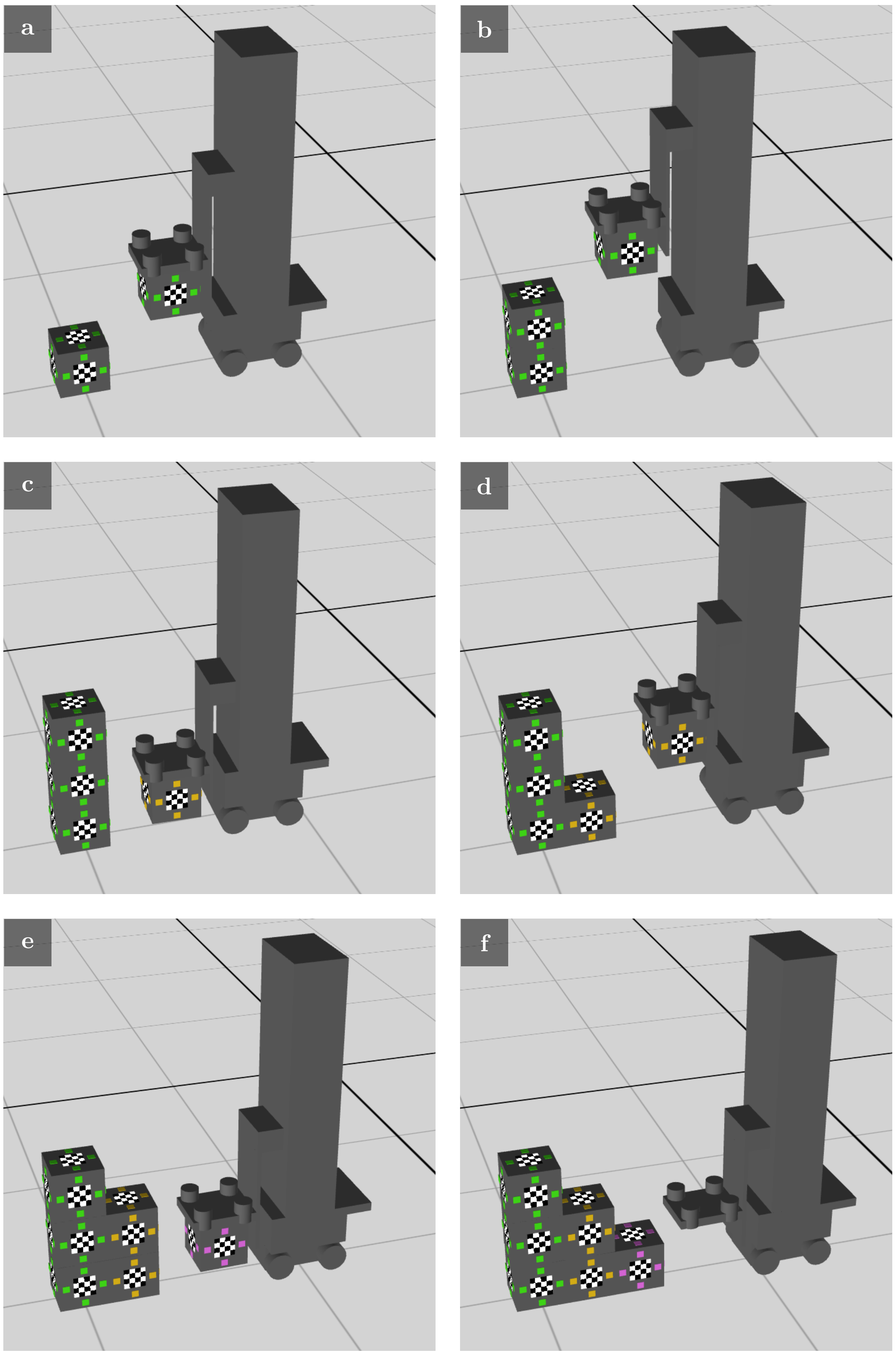


Figure 6.6: Construction of a staircase using a single autonomous robot in simulation.

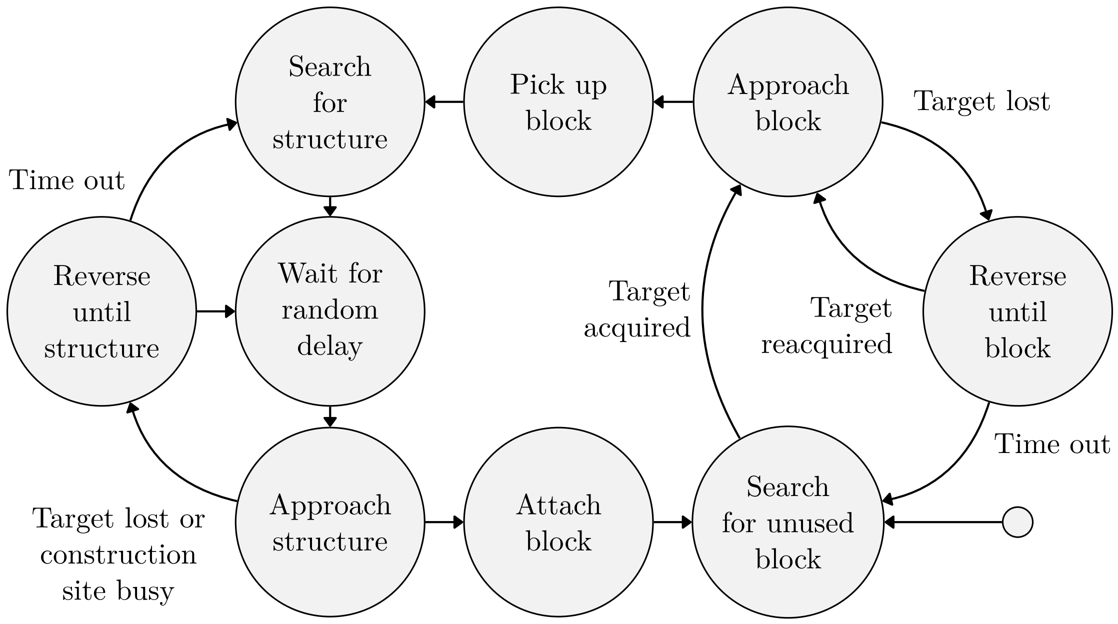


Figure 6.7: High-level state chart for the stepped pyramid demonstration.

6.2.3 Construction of a stepped pyramid

Objective In this demonstration, we aim to show how our decentralized control strategy can scale to a multi-robot construction scenario by using four robots to build a stepped pyramid.

Setup and procedure The four robots in this demonstration are effectively building four of the staircases from the demonstration presented in Section 6.2.2 with a common central column. As the robots do not have an internal representation of the simulated environment or a notion of the structure being built, we only need to make minor modifications to our state machine to include a collision avoidance mechanism. Figure 6.7 shows how we achieve the required collision avoidance by adding an additional transition to the state machine that causes a robot to abort and retry its block placement routine if the construction site is busy. A robot senses whether the construction site is busy through the use of its rangefinders and computer vision. We also add a random delay state, which lowers the probability that all four robots approach the structure at the same time.

Results The plot in Figure 6.8 shows the average construction progress from five runs. Initially, all robots attempt to construct the central column, saturating this construction site. After the robots have built the central column, however, the rate of construction increases as the robots start to build the wings of the stepped pyramid in parallel. The rate of construction decreases towards the end of the demonstration as we have

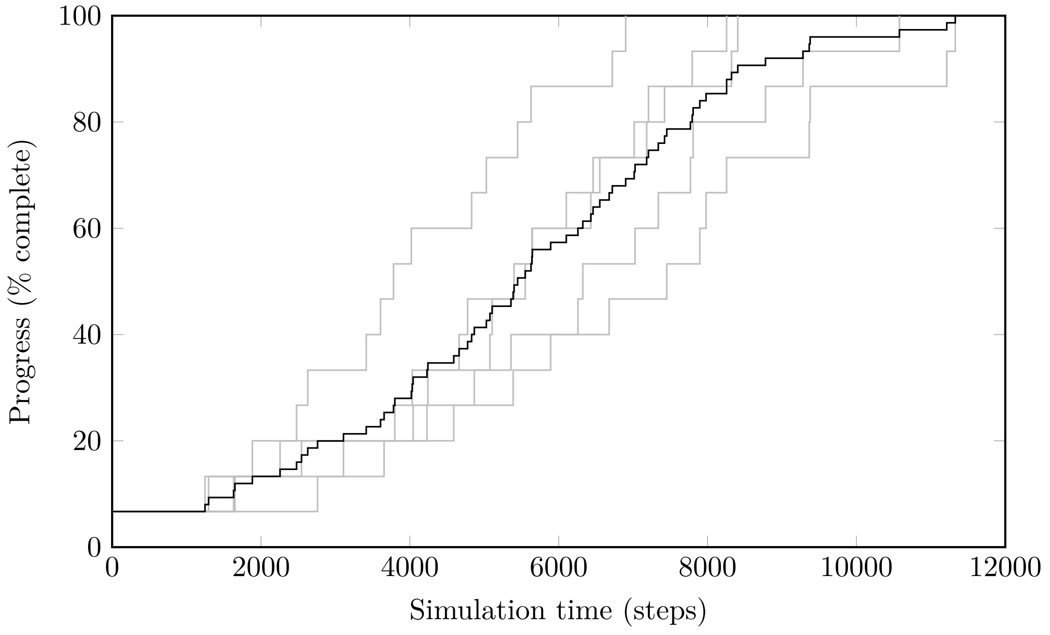


Figure 6.8: Construction progress for the stepped pyramid as depicted in Figure 6.9. The black line shows the average progress, while the grey lines show the progress from each of the five individual runs.

programmed the robots to shut down after they have attached their last block to the structure. Figure 6.9 shows twelve snapshots of the robots building the stepped pyramid.

6.3 Discussion

In this section, we discuss the results from the demonstrations of the two implementations of our multi-robot ACS.

6.3.1 The decentralized control stategy

The staircase demonstration showed how our decentralized control strategy is used to coordinate construction. In this demonstration, an autonomous robot performed construction in response to the structural arrangement and LED markings of the stigmergic blocks in the partially-built staircase. The coordination that led to the construction of the staircase was the result of a feedback loop, where construction actions by the autonomous robot modified the partially-built staircase, and modifications to the partially-built staircase caused the autonomous robot to perform further construction.

As noted in Section 6.2.3, we only needed to make minor adaptations to the staircase demonstration in order to build the stepped pyramid. This ease of adaptation reveals an interesting characteristic of our decentralized control strategy, which arises from two

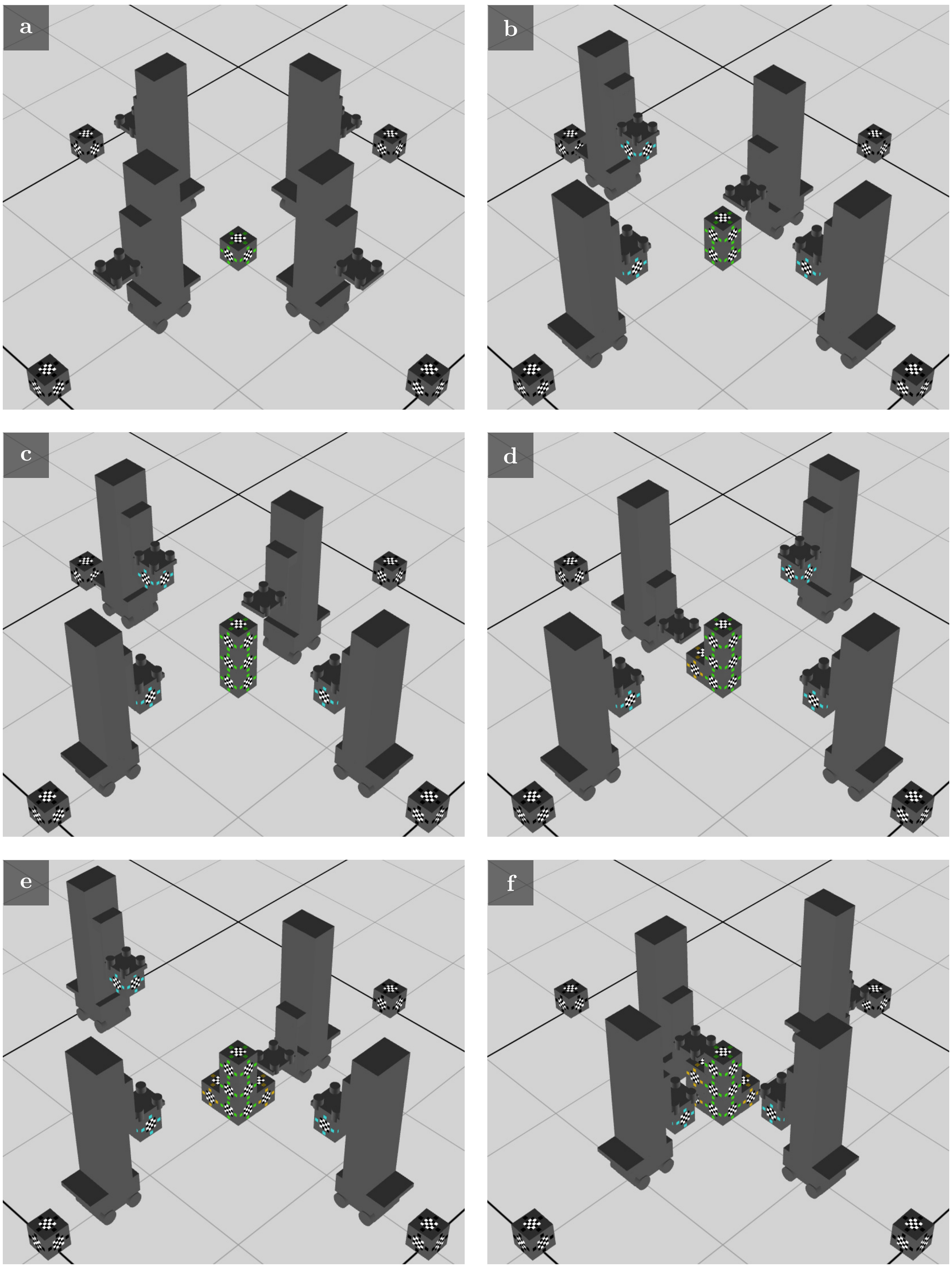


Figure 6.9: Construction of a stepped pyramid using four autonomous robots in simulation.

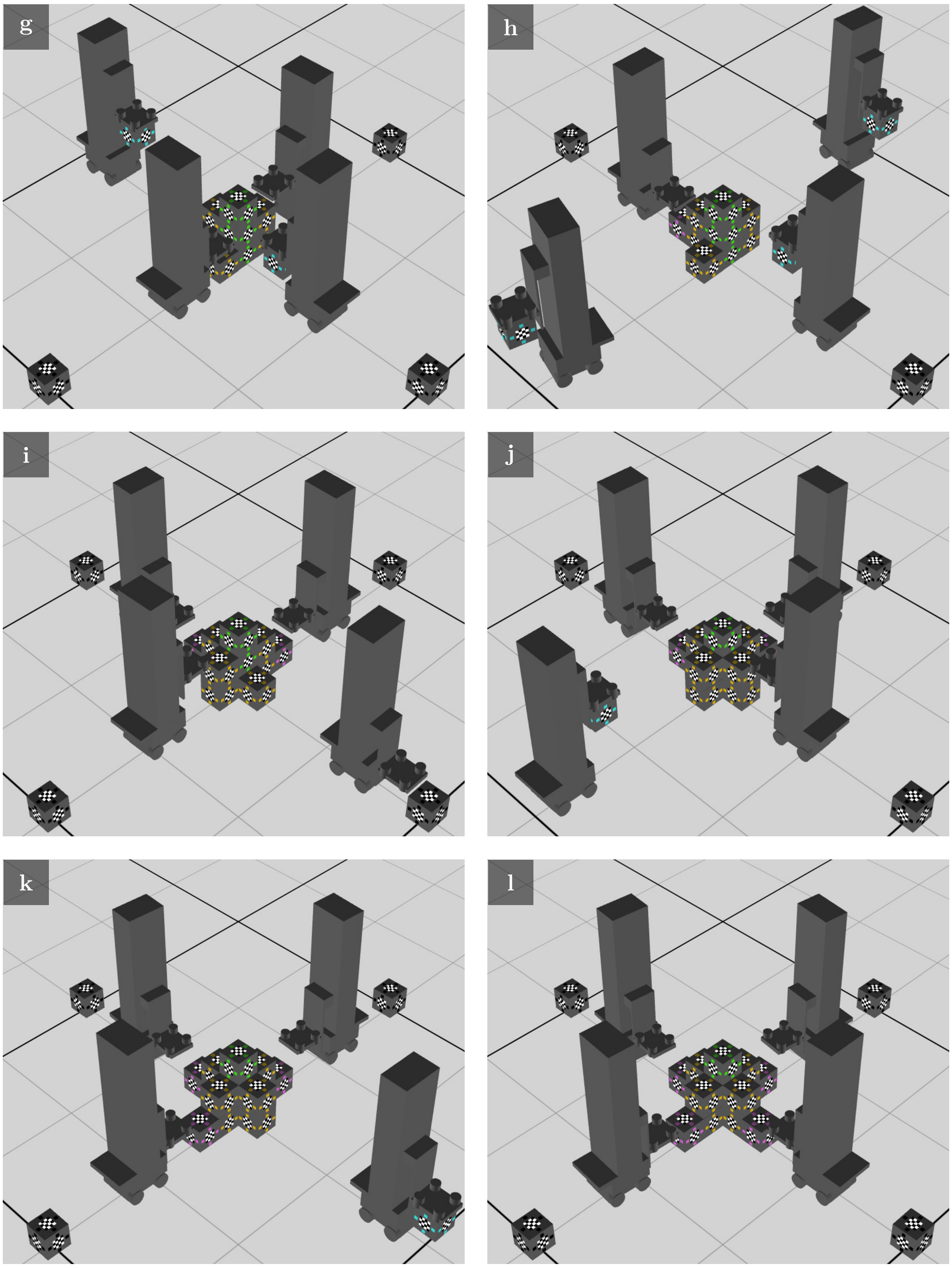


Figure 6.9: Construction of a stepped pyramid using four autonomous robots in simulation.

factors. Firstly, the robots do not maintain nor rely on an internal representation of the environment or of the structures being built. Instead, the robots coordinate their actions using only locally available information such as their observations of a partially-built structure. Secondly, the stepped pyramid is related to the staircase through radial symmetry. This relationship enables the same combinations of structural arrangements and LED markings that regulated the construction of the staircase to also regulate the construction of the stepped pyramid. It is possible that this characteristic of our decentralized control strategy can be exploited to efficiently coordinate the construction of structures with a high degree of symmetry.

As shown in Figure 6.8, the rate of construction for the stepped pyramid started out slowly due to congestion at the central column. This congestion was the result of using a constrained approach to the allocation of the construction tasks. This approach involved statically assigning each robot to work on a single wing of the pyramid. We believe, however, that it may be possible to mitigate this congestion by using a more flexible approach to the allocation of construction tasks, potentially involving multiple robots constructing several structures in parallel.

6.3.2 Portability between hardware and simulation

The construction of the staircase was performed using the hardware implementation and the implementation in simulation. Our decentralized control strategy for this demonstration was implemented as a finite state machine, which controlled the autonomous robot's behavior. This behavior resulted in the autonomous robot cycling back and forth between locating an unused block and attaching that unused block to the partially-built staircase.

With some minor modifications (as discussed in Section 5.3.2), this state machine successfully coordinated the construction of the staircase using the hardware implementation and the implementation in simulation. This portability between the hardware and the simulation is a result of the extensions to the ARGoS simulator, which enable a realistic simulation of our multi-robot ACS.

6.3.3 Collision avoidance

In Section 6.1.3, we demonstrated a construction scenario where two robots were tasked with constructing a column. In this task, we assigned one of the stigmergic block colors to represent a block being transported. When an autonomous robot detects a block being transported, it activates a collision avoidance mechanism and tracks the transported block until it disappears.

This solution to collision avoidance is built on top of the existing computer vision algorithms and incurs a negligible computational overhead. Due to the limited field of view, however, this solution is not comprehensive. Nonetheless, we are able to improve the effectiveness of this solution by combining the output from the computer vision algorithm with other sensory input such as from the rangefinders. We use this combination of computer vision and input from an autonomous robot's rangefinders in the stepped pyramid demonstration to detect whether a construction site is busy. A disadvantage of this approach, however, is that one of the four possible LED colors is no longer available for regulating construction.

6.4 Summary

Instead of maintaining an internal representation of the environment and performing construction with respect to a blueprint, the autonomous robots in our multi-robot ACS perform construction in response to the structural arrangement of the stigmergic blocks in the environment and their LED markings.

Jones and Matarić used a similar approach to coordinate construction in a case study on the automatic synthesis of multi-robot controllers [34]. The robots in this work were limited, however, to building planar structures and relied on a human operator to perform manipulation. This use of a human operator to perform manipulation sidesteps significant challenges in the design and operation of a multi-robot ACS.

In contrast to the work by Jones and Matarić, we have designed a multi-robot ACS that is capable of building three-dimensional structures and that is completely autonomous. We have implemented this multi-robot ACS using hardware and in simulation, verifying that our decentralized control strategy can indeed be used to coordinate multi-robot construction. These results constitute a contribution to the research on multi-robot ACSs by demonstrating that it is possible to use a decentralized control strategy based on structure and markings to coordinate autonomous three-dimensional construction in a physical system.

CHAPTER 7
Conclusion

7.1 Overview

At the start of this thesis, we noted that despite numerous examples of decentralized construction in nature by social insects, there was limited research on how a decentralized control strategy, in particular, one based on the principles of swarm intelligence, could be implemented on a multi-robot ACS. To this end, we proposed a decentralized control strategy based on the work by Theraulaz and Bonabeau, who simulated multi-agent construction in a three-dimensional lattice [98].

In this thesis, we provided a literature review where we analyzed and compared the existing approaches to coordinating construction in a multi-robot ACS. We found that there was only one multi-robot ACS that coordinated construction using a control strategy similar to Theraulaz and Bonabeau's. This multi-robot ACS was presented as a case study for the automatic synthesis of controllers for multi-robot systems [34]. However, this multi-robot ACS was limited to planar structures and required a human operator to perform manipulation on behalf of the robots, side stepping significant challenges in the design and operation of a multi-robot ACS.

We devised a decentralized control strategy by making several modifications to the control strategy from the work of Theraulaz and Bonabeau. These modifications included a practical solution to working with different types of building material and compensating for the inherent discretization in Theraulaz and Bonabeau's simulation. We demonstrated that these modifications enabled us to realize our decentralized control strategy in a physical system by designing a new multi-robot ACS consisting of stigmergic blocks and autonomous robots, which assembled the stigmergic blocks into structures. We implemented these components using hardware and verified the functionality of that hardware using two tasks. These tasks required an autonomous robot to locate an unused block in its environment and to attach this block to a partially-built structure with respect to the structural arrangement of the stigmergic blocks already in the structure and their LED markings.

We also provided a secondary implementation of our multi-robot ACS using simulation. This implementation was built on top of the ARGoS simulator, for which we developed several extensions to support our work. These extensions enabled us to accurately simulate the computer vision and to model the three-dimensional dynamics of a stigmergic block and an autonomous robot. We demonstrated that this implementation in simulation was accurate by running the same two tasks that we used to verify the functionality of the hardware.

To show how our decentralized control strategy could be used to coordinate construction, we provided several demonstrations using the hardware and in simulation. These demonstrations included the construction of a column by two robots using the hardware, the construction of a staircase by a robot using the hardware and in simulation, and the construction of a stepped pyramid by four robots in simulation.

7.2 Future work

In this thesis, we have used both the hardware implementation and the implementation in simulation of our multi-robot ACS to demonstrate the potential of our decentralized control strategy. To enable further research, however, both implementations will require further development.

7.2.1 Enhancements to the hardware implementation

As noted in Section 4.3 and Section 6.1, the drive system in the autonomous robots is currently unreliable and needs to be upgraded. In the short term, this upgrade involves replacing the motor modules, with modules that have a higher gear ratio that reduces speed and increases torque. Furthermore, the new modules would be more effective if a lower resolution encoder was selected, as the current resolution is too sensitive and appears to occasionally overload the microcontroller with interrupts, which causes it to stall. Finally, the mounting of these motors needs to be improved such that the weight of the entire robot does not rest on the shafts of the motors.

In the long term, however, moving to an omnidirectional drive system would be far better suited to our application. This suitability arises from the fact that an omnidirectional drive system can simultaneously adjust its position and orientation, allowing for faster alignment with an unused block or partially-built structure. This solution stands in contrast to the current design of our autonomous robots that use a differential drive system and at times requires multiple approaches before a suitable alignment with a partially-built structure can be achieved.

Although not as consequential as the drive system, the camera module can be upgraded to improve system functionality. In the current design, the camera module is fixed at an angle of 45 degrees. This angle provides a compromise between allowing a robot to observe blocks at a distance and to track a block during its approach. An alternative to this compromise is to add a servo motor to the camera module that facilitates an adjustable angle, enabling the camera to capture the scene from both perspectives.

7.2.2 Enhancements to the simulation implementation

We intend to leverage the implementation of our multi-robot ACS in simulation to investigate how our decentralized control strategy for multi-robot construction scales by running experiments with large numbers of stigmergic blocks and autonomous robots.

We have noted during running experiments that the speed of a simulation decreases as more blocks are added to a structure. This decrease in speed may be related to an optimization in the Bullet physics engines that allows a body to *sleep* when its velocity is near zero. We hypothesize that our magnetism plugin, which applies forces to bodies, may be preventing the spherical magnets inside a block from sleeping. If further analysis provides evidence to support this hypothesis, a significant optimization may be possible by disabling the magnetic forces between sleeping bodies.

7.2.3 Enhancements to the behavior of an autonomous robot

Substructure detection In the demonstrations presented in this thesis, we have set up our controllers to identify specific structural arrangements of the stigmergic blocks. In order to enable construction of other structures and to reduce the required development overhead, we aim to generalize this approach by using substructure detection. This detection would allow an autonomous robot to detect, for example, an L-shaped arrangement of stigmergic blocks with specific LED markings and to use this detection to trigger a construction action. This may lead to a more generic controller, where a set of rules that map substructure detections to construction actions could be described externally using XML.

Random walk and obstacle avoidance The demonstrations presented in this thesis were set up so that an autonomous robot could locate an unused stigmergic block or a partially-built structure by turning on the spot. This approach was chosen to mitigate the issues with the drive system. Once these issues are resolved, however, we aim to allow an autonomous robot to search its environment for unused blocks and partially-built structures by means of a random walk behavior. This random walk behavior would include an obstacle avoidance mechanism to prevent an autonomous robot from colliding with unused blocks, other robots, partially-built structures, and the walls of an arena.

Using ARGoS with the hardware At the time of writing, both the hardware implementation and the implementation in simulation of our multi-robot ACS use a similar state machine to control the behavior of an autonomous robot. The two implementations, however, use different wrapper code to interface the physical and simulated hardware. Maintaining these two wrappers is time-consuming and error-prone, which at times may impact the accuracy of a simulation. To this end, it is desirable to use ARGoS on an autonomous robot instead of the blocktracker executable. Furthermore, by porting the code that controls the behavior of an autonomous robot from C++ to Lua, we would enable both implementations to share an identical Lua script, which is compatible with and faster to deploy on both implementations. Furthermore, it is advantageous to use ARGoS's configuration file to tune parameters for and to provide a set of construction rules to a controller.

7.2.4 Future research directions

In this section, we discuss future research directions by proposing several different construction scenarios involving autonomous robots and stigmergic blocks.

Multi-robot construction Figure 7.1 shows a stepped pyramid, which has been built by two robots. Due to the issues with the drive system on the autonomous robot, however, we have only achieved this in simulation. To this end, realizing the multi-robot construction of the stepped pyramid using the hardware implementation is the next milestone in our research.

The use of multiple robots to perform construction increases construction throughput. There are two ways in which this increase in throughput occurs: (i) while a robot is attaching a block to a partially-built structure, another robot searches an environment for unused blocks, and (ii) through the use of multiple construction sites on a single structure, two robots may attach blocks to a partially-built structure in parallel.

The use of multiple construction sites on a single structure in a multi-robot ACS mitigates congestion at a given construction site due to multiple robots attempting to perform the same construction action. This issue of congestion was present in the work of Wawerla et al. who used multiple robots to construct a wall from Velcro blocks [107].

Recursive construction As per our discussion in Section 6.3.1, the staircase and the stepped pyramid are related to one another through radial symmetry. Specifically, the stepped pyramid is four of the staircases with a common central column.

Looking forward, we believe that our decentralized control strategy is well suited to construction scenarios in which structures exhibit high degrees of symmetry and can be expressed in terms of recursive patterns. Figure 7.2 shows an example of how a recursive pattern may be leveraged to enable construction. In this example, an observation of the L-shaped substructure on either side of the wall initiates the construction of a new L-shaped substructure. If uninterrupted, this process will continue indefinitely until the robots have used up the unused stigmergic blocks.

Adaptive construction Figure 7.3 shows a further construction scenario with two robots attempting to contain two green “radioactive” blocks. In this construction scenario, the robots are placed in an environment with unused stigmergic blocks and a couple of radioactive blocks. The robots must contain the radioactive blocks by building walls around them from unused stigmergic blocks. In effect, these radioactive blocks create a template in the environment to which the autonomous robots perform adaptive construction. A variant of this construction scenario is also possible where the autonomous robots first cluster the radioactive blocks prior to containing them.

Self-organization in a multi-robot ACS In a more advanced construction scenario, we may also apply recent results from research in the self-organization of robot swarms. For example, we may implement a variant of dynamic task allocation where a subset

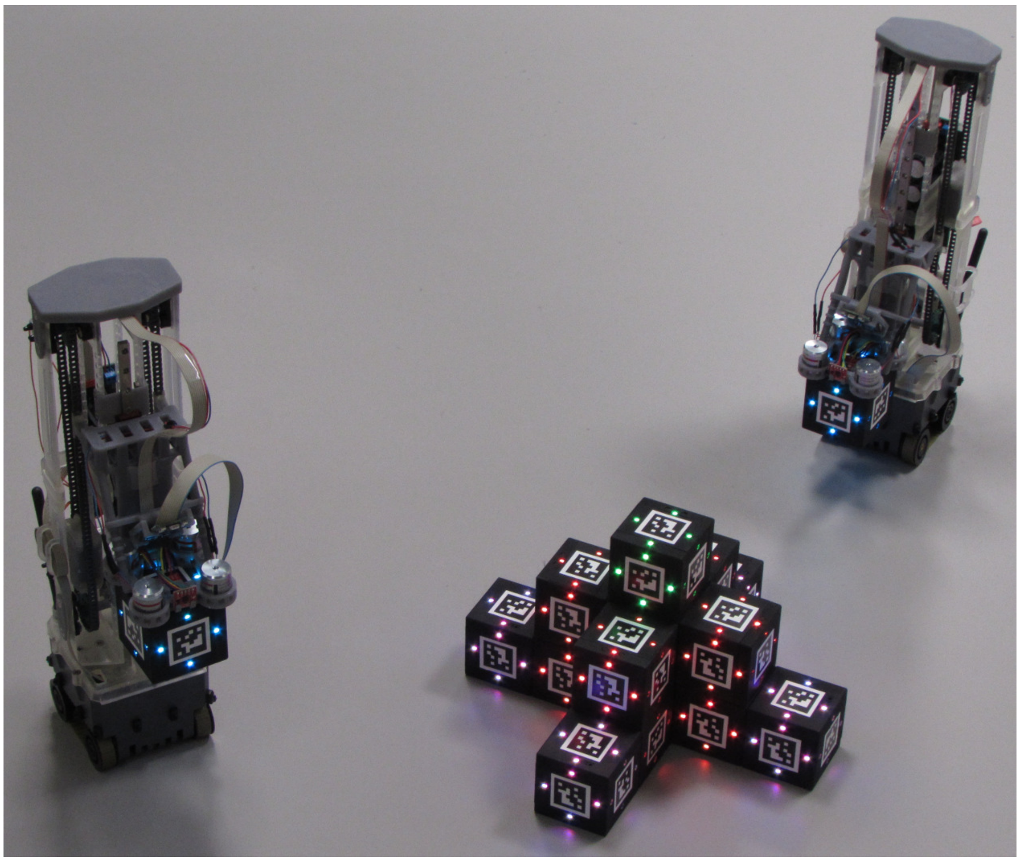


Figure 7.1: Two robots approach and inspect a stepped pyramid.

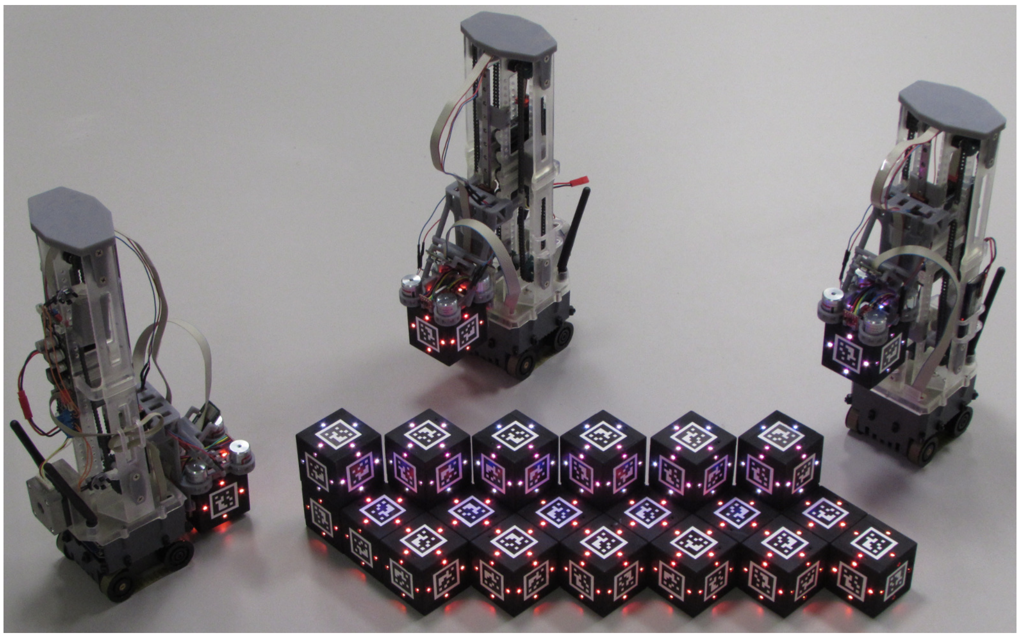


Figure 7.2: Robots constructing a wall using a recursive pattern.

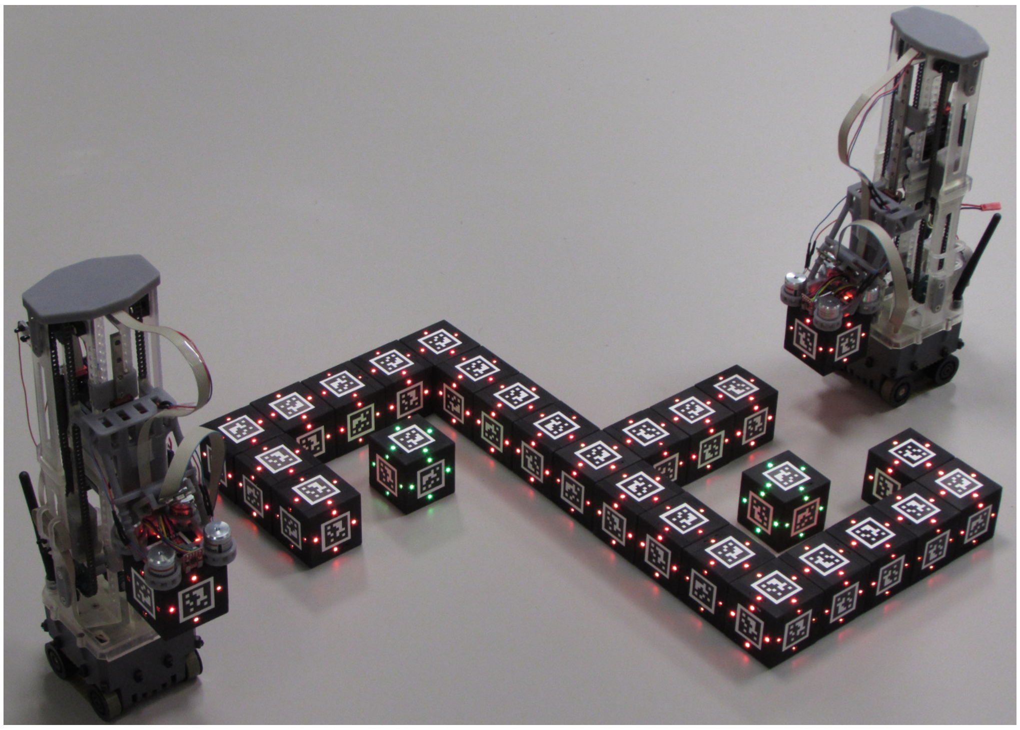


Figure 7.3: Two robots constructing walls from stigmeric blocks (red) to contain the two "radioactive" blocks (green).

of the robots assign themselves in a decentralized fashion to gather and arrange unused blocks into a cache nearby a construction site while the remaining robots perform construction. Furthermore, we may implement a variant of collective exploration and decision making, where a group of robots explores its environment before reaching a consensus on the optimal location for a given structure collectively.

Construction and maintenance of smart structures Since a stigmergic block has an NFC interface embedded in each of its faces, it is possible for two contiguous stigmergic blocks to communicate with each other and to route information to different parts of a structure. Combined with the LEDs on the faces of each block, these capabilities would then allow a structure to communicate with the autonomous robots in an environment to partially control its own construction or maintenance.

7.3 Summary

In this thesis, we have presented a decentralized control strategy that coordinates construction in a multi-robot ACS. This control strategy utilizes the structural arrangement and markings of the blocks already in a partially-built structure to coordinate further construction.

By demonstrating the functionality and basic characteristics of this control strategy, we believe that we have laid the foundations for a significant and new area of research. To this end, we have noted several research directions that have not yet been explored.

Significant work is still required to improve the functionality and reliability of our multi-robot ACS, however, through continued collaboration and by keeping our research open (i.e. publishing our work, keeping our designs and code on public repositories, and where possible, using open source tools), we believe that this area of research has the potential to significantly impact the way in which we think about and approach construction in terrestrial, and potentially, lunar and martian environments.

APPENDIX A
Description of a Stigmergic Block in ARGoS

```
<prototype>
  <controller config="stigmergic_block_controller"/>
  <!-- physical bodies of a stigmergic block -->
  <bodies reference_body="block">
    <body id="block" geometry="box" size="0.055,0.055,0.055" mass="0.102">
      <offset/>
      <coordinates/>
    </body>
    <body id="magnet_tq1" geometry="sphere" radius="0.003" mass="0.001">
      <offset position=" 0.0225, 0.0225, 0.047" orientation="0,0,0"/>
      <coordinates/>
    </body>
    <body id="magnet_tq2" geometry="sphere" radius="0.003" mass="0.001">
      <offset position="-0.0225, 0.0225, 0.047" orientation="0,0,0"/>
      <coordinates/>
    </body>
    <body id="magnet_tq3" geometry="sphere" radius="0.003" mass="0.001">
      <offset position="-0.0225,-0.0225, 0.047" orientation="0,0,0"/>
      <coordinates/>
    </body>
    <body id="magnet_tq4" geometry="sphere" radius="0.003" mass="0.001">
      <offset position=" 0.0225,-0.0225, 0.047" orientation="0,0,0"/>
      <coordinates/>
    </body>
    <body id="magnet_bq1" geometry="sphere" radius="0.003" mass="0.001">
      <offset position=" 0.0225, 0.0225, 0.002" orientation="0,0,0"/>
      <coordinates/>
    </body>
    <body id="magnet_bq2" geometry="sphere" radius="0.003" mass="0.001">
      <offset position="-0.0225, 0.0225, 0.002" orientation="0,0,0"/>
      <coordinates/>
    </body>
    <body id="magnet_bq3" geometry="sphere" radius="0.003" mass="0.001">
      <offset position="-0.0225,-0.0225, 0.002" orientation="0,0,0"/>
      <coordinates/>
    </body>
    <body id="magnet_bq4" geometry="sphere" radius="0.003" mass="0.001">
      <offset position=" 0.0225,-0.0225, 0.002" orientation="0,0,0"/>
      <coordinates/>
    </body>
  </bodies>
  <!-- joints between the physical bodies of a stigmergic block -->
  <joints>
    <joint id="block:magnet_tq1" disable_collisions="true">
      <frames>
        <frame body="block" position="0.0225,0.0225,0.050"
               orientation="0,0,0"/>
        <frame body="magnet_tq1" position="0,0,0.003" orientation="0,0,0"/>
      </frames>
      <axes>
        <axis direction="x" mode="angular" range="unconstrained"/>
        <axis direction="y" mode="angular" range="unconstrained"/>
        <axis direction="z" mode="angular" range="unconstrained"/>
      </axes>
    </joint>
    <joint id="block:magnet_tq2" disable_collisions="true">
      <frames>
        <frame body="block" position="-0.0225,0.0225,0.050"
               orientation="0,0,0"/>
        <frame body="magnet_tq2" position="0,0,0.003" orientation="0,0,0"/>
      </frames>
      <axes>
        <axis direction="x" mode="angular" range="unconstrained"/>
        <axis direction="y" mode="angular" range="unconstrained"/>
        <axis direction="z" mode="angular" range="unconstrained"/>
      </axes>
    </joint>
    <joint id="block:magnet_tq3" disable_collisions="true">
      <frames>
        <frame body="block" position="-0.0225,-0.0225,0.050"
        orientation="0,0,0"/>
        <frame body="magnet_tq3" position="0,0,0.003" orientation="0,0,0"/>
      </frames>
      <axes>
        <axis direction="x" mode="angular" range="unconstrained"/>
        <axis direction="y" mode="angular" range="unconstrained"/>
        <axis direction="z" mode="angular" range="unconstrained"/>
      </axes>
    </joint>
    <joint id="block:magnet_tq4" disable_collisions="true">
      <frames>
        <frame body="block" position="0.0225,-0.0225,0.050"
        orientation="0,0,0"/>
        <frame body="magnet_tq4" position="0,0,0.003" orientation="0,0,0"/>
      </frames>
      <axes>
        <axis direction="x" mode="angular" range="unconstrained"/>
        <axis direction="y" mode="angular" range="unconstrained"/>
        <axis direction="z" mode="angular" range="unconstrained"/>
      </axes>
    </joint>
    <joint id="block:magnet_bq1" disable_collisions="true">
      <frames>
        <frame body="block" position="0.0225,0.0225,0.005"
               orientation="0,0,0"/>
        <frame body="magnet_bq1" position="0,0,0.003" orientation="0,0,0"/>
      </frames>
      <axes>
        <axis direction="x" mode="angular" range="unconstrained"/>
        <axis direction="y" mode="angular" range="unconstrained"/>
        <axis direction="z" mode="angular" range="unconstrained"/>
      </axes>
    </joint>
    <joint id="block:magnet_bq2" disable_collisions="true">
      <frames>
        <frame body="block" position="-0.0225,0.0225,0.005"
               orientation="0,0,0"/>
        <frame body="magnet_bq2" position="0,0,0.003" orientation="0,0,0"/>
      </frames>
      <axes>
```

```
        <axis direction="x" mode="angular" range="unconstrained"/>
        <axis direction="y" mode="angular" range="unconstrained"/>
        <axis direction="z" mode="angular" range="unconstrained"/>
      </axes>
    </joint>
    <joint id="block:magnet_bq3" disable_collisions="true">
      <frames>
        <frame body="block" position="-0.0225,-0.0225,0.005"
               orientation="0,0,0"/>
        <frame body="magnet_bq3" position="0,0,0.003" orientation="0,0,0"/>
      </frames>
      <axes>
        <axis direction="x" mode="angular" range="unconstrained"/>
        <axis direction="y" mode="angular" range="unconstrained"/>
        <axis direction="z" mode="angular" range="unconstrained"/>
      </axes>
    </joint>
    <joint id="block:magnet_bq4" disable_collisions="true">
      <frames>
        <frame body="block" position="0.0225,-0.0225,0.005"
               orientation="0,0,0"/>
        <frame body="magnet_bq4" position="0,0,0.003" orientation="0,0,0"/>
      </frames>
      <axes>
        <axis direction="x" mode="angular" range="unconstrained"/>
        <axis direction="y" mode="angular" range="unconstrained"/>
        <axis direction="z" mode="angular" range="unconstrained"/>
      </axes>
    </joint>
  </joints>
  <!-- devices of a stigmergic block -->
  <devices>
    <electromagnets>
      <electromagnet body="magnet_tq1" passive_field="0,0,75"/>
      <electromagnet body="magnet_tq2" passive_field="0,0,75"/>
      <electromagnet body="magnet_tq3" passive_field="0,0,75"/>
      <electromagnet body="magnet_tq4" passive_field="0,0,75"/>
      <electromagnet body="magnet_bq1" passive_field="0,0,75"/>
      <electromagnet body="magnet_bq2" passive_field="0,0,75"/>
      <electromagnet body="magnet_bq3" passive_field="0,0,75"/>
      <electromagnet body="magnet_bq4" passive_field="0,0,75"/>
    </electromagnets>
    <tags medium="apriltags">
      <tag id="top" body="block" position="0.000,0.000,0.056"
           orientation="0,0,0" observable_angle="75" side_length="0.024"/>
      <tag id="north" body="block" position="0.0285,0.000,0.0275"
           orientation="0,90,0" observable_angle="75" side_length="0.024"/>
      <tag id="east" body="block" position="0.000,-0.0285,0.0275"
           orientation="0,0,90" observable_angle="75" side_length="0.024"/>
      <tag id="south" body="block" position="-0.0285,0.000,0.0275"
           orientation="0,-90,0" observable_angle="75" side_length="0.024"/>
      <tag id="west" body="block" position="0.000,0.0285,0.0275"
           orientation="0,0,-90" observable_angle="75" side_length="0.024"/>
      <tag id="bottom" body="block" position="0.000,0.000,-0.001"
           orientation="0,0,180" observable_angle="75" side_length="0.024"/>
    </tags>
    <leds medium="leds">
      <led id="top_a" color="black" observable_angle="75" body="block"
           position="0.000,0.020,0.056" orientation="0,0,0"/>
      <led id="top_b" color="black" observable_angle="75" body="block"
           position="0.020,0.000,0.056" orientation="0,0,0"/>
      <led id="top_c" color="black" observable_angle="75" body="block"
           position="0.000,-0.020,0.056" orientation="0,0,0"/>
      <led id="top_d" color="black" observable_angle="75" body="block"
           position="-0.020,0.000,0.056" orientation="0,0,0"/>
      <led id="north_a" color="black" observable_angle="75" body="block"
           position="0.0285,0.000,0.0475" orientation="0,90,0"/>
      <led id="north_b" color="black" observable_angle="75" body="block"
           position="0.0285,0.020,0.0275" orientation="0,90,0"/>
      <led id="north_c" color="black" observable_angle="75" body="block"
           position="0.0285,0.000,0.0075" orientation="0,90,0"/>
      <led id="north_d" color="black" observable_angle="75" body="block"
           position="0.0285,-0.020,0.0275" orientation="0,90,0"/>
      <led id="east_a" color="black" observable_angle="75" body="block"
           position="0.000,-0.0285,0.0475" orientation="0,0,90"/>
      <led id="east_b" color="black" observable_angle="75" body="block"
           position="0.020,-0.0285,0.0275" orientation="0,0,90"/>
      <led id="east_c" color="black" observable_angle="75" body="block"
           position="0.000,-0.0285,0.0075" orientation="0,0,90"/>
      <led id="east_d" color="black" observable_angle="75" body="block"
           position="-0.020,-0.0285,0.0275" orientation="0,0,90"/>
      <led id="south_a" color="black" observable_angle="75" body="block"
           position="-0.0285,0.000,0.0475" orientation="0,-90,0"/>
      <led id="south_b" color="black" observable_angle="75" body="block"
           position="-0.0285,-0.020,0.0275" orientation="0,-90,0"/>
      <led id="south_c" color="black" observable_angle="75" body="block"
           position="-0.0285,0.000,0.0075" orientation="0,-90,0"/>
      <led id="south_d" color="black" observable_angle="75" body="block"
           position="-0.0285,0.020,0.0275" orientation="0,-90,0"/>
      <led id="west_a" color="black" observable_angle="75" body="block"
           position="0.000,0.0285,0.0475" orientation="0,0,-90"/>
      <led id="west_b" color="black" observable_angle="75" body="block"
           position="-0.020,0.0285,0.0275" orientation="0,0,-90"/>
      <led id="west_c" color="black" observable_angle="75" body="block"
           position="0.000,0.0285,0.0075" orientation="0,0,-90"/>
      <led id="west_d" color="black" observable_angle="75" body="block"
           position="0.020,0.0285,0.0275" orientation="0,0,-90"/>
      <led id="bottom_a" color="black" observable_angle="75" body="block"
           position="0.000,0.020,-0.001" orientation="0,0,180"/>
      <led id="bottom_b" color="black" observable_angle="75" body="block"
           position="0.020,0.000,-0.001" orientation="0,0,180"/>
      <led id="bottom_c" color="black" observable_angle="75" body="block"
           position="0.000,-0.020,-0.001" orientation="0,0,180"/>
      <led id="bottom_d" color="black" observable_angle="75" body="block"
           position="-0.020,0.000,-0.001" orientation="0,0,180"/>
    </leds>
    <radios>
      <radio id="top" medium="nfc" duplex_mode="half" range="0.020"
             body="block" position="0.000,0.000,0.050" orientation="0,0,0"/>
      <radio id="north" medium="nfc" duplex_mode="half" range="0.020"
             body="block" position="0.0225,0.000,0.0275" orientation="0,90,0"/>
      <radio id="east" medium="nfc" duplex_mode="half" range="0.020"
             body="block" position="0.000,-0.0225,0.0275"
             orientation="0,0,90"/>
      <radio id="south" medium="nfc" duplex_mode="half" range="0.020"
             body="block" position="-0.0225,0.000,0.0275"
             orientation="0,-90,0"/>
      <radio id="west" medium="nfc" duplex_mode="half" range="0.020"
             body="block" position="0.000,0.0225,0.0275"
             orientation="0,0,-90"/>
      <radio id="bottom" medium="nfc" duplex_mode="half" range="0.020"
             body="block" position="0.000,0.000,0.005" orientation="0,0,180"/>
    </radios>
  </devices>
</prototype>
```

APPENDIX B Description of an Autonomous Robot in ARGoS

```
<prototype>
  <controller config="robot_ctrl"/>
  <bodies reference_body="lower-chassis">
    <body id="lower-chassis" geometry="box" size="0.089510,0.057000,0.030000"
          mass="0.250">
      <coordinates/>
      <offset position="0,0,0.005750"/>
    </body>
    <body id="wheel-front-left" geometry="cylinder" radius="0.01575"
          height="0.01400" mass="0.100">
      <coordinates/>
      <offset position="0.031750, 0.02850,0.01575" orientation="0,0,-90"/>
    </body>
    <body id="wheel-front-right" geometry="cylinder" radius="0.01575"
          height="0.01400" mass="0.100">
      <coordinates/>
      <offset position="0.031750,-0.02850,0.01575" orientation="0,0,90"/>
    </body>
    <body id="wheel-rear-left" geometry="cylinder" radius="0.01575"
          height="0.01400" mass="0.100">
      <coordinates/>
      <offset position="-0.031750, 0.02850, 0.01575" orientation="0,0,-90"/>
    </body>
    <body id="wheel-rear-right" geometry="cylinder" radius="0.01575"
          height="0.01400" mass="0.100">
      <coordinates/>
      <offset position="-0.031750, -0.02850, 0.01575" orientation="0,0,90"/>
    </body>
    <body id="upper-chassis" geometry="box" size="0.089510,0.087500,0.034500"
          mass="0.200">
      <coordinates/>
      <offset position="0,0,0.035750"/>
    </body>
    <body id="top-fixture" geometry="box" size="0.118600,0.087500,0.010000"
          mass="0.500">
      <coordinates/>
      <offset position="-0.014545,0,0.07025"/>
    </body>
    <body id="lift-fixture" geometry="box" size="0.0585,0.08750,0.28775"
          mass="0.500">
      <coordinates/>
      <offset position="0,0,0.080250"/>
    </body>
```

```xml
    <body id="vertical-link" geometry="box" size="0.004000,0.060000,0.1185"
          mass="0.250">
      <coordinates/>
      <offset position="0.055755,0,0.01975"/>
    </body>
    <body id="horizontal-link" geometry="box" size="0.072000,0.060000,0.010000"
          mass="0.150">
      <coordinates/>
      <offset position="0.093755,0,0.055750"/>
    </body>
    <!-- for visualization purposes -->
    <body id="visual-link" geometry="box" size="0.028505,0.060000,0.025000"
          mass="0.005">
      <coordinates/>
      <offset position="0.0435025,0,0.13825"/>
    </body>
    <!-- for visualization purposes -->
    <body id="electromagnet-q1" geometry="cylinder" radius="0.010"
          height="0.023500" mass="0.010">
      <coordinates/>
      <offset position="0.070755,0.023000,0.055750"/>
    </body>
    <body id="electromagnet-q2" geometry="cylinder" radius="0.010"
          height="0.023500" mass="0.010">
      <coordinates/>
      <offset position="0.116755,0.023000,0.055750"/>
    </body>
    <body id="electromagnet-q3" geometry="cylinder" radius="0.010"
          height="0.023500" mass="0.010">
      <coordinates/>
      <offset position="0.116755,-0.023000,0.055750"/>
    </body>
    <body id="electromagnet-q4" geometry="cylinder" radius="0.010"
          height="0.023500" mass="0.010">
      <coordinates/>
      <offset position="0.070755,-0.023000,0.055750"/>
    </body>
  </bodies>
  <joints>
    <joint id="wheel-front-left:lower-chassis" disable_collisions="true">
      <frames>
        <frame body="wheel-front-left" position="0,0,0" orientation="0,0,90"/>
        <frame body="lower-chassis" position="0.031750, 0.02850, 0.01000"
               orientation="0,0,0"/>
      </frames>
      <axes>
        <axis direction="y" mode="angular" range="unconstrained">
          <actuator enabled="true" force="15" target_velocity="0"/>
        </axis>
      </axes>
    </joint>
    <joint id="wheel-front-right:lower-chassis" disable_collisions="true">
      <frames>
        <frame body="wheel-front-right" position="0,0,0"
               orientation="0,0,-90"/>
        <frame body="lower-chassis" position="0.031750, -0.02850, 0.01000"
               orientation="0,0,0"/>
      </frames>
      <axes>
        <axis direction="y" mode="angular" range="unconstrained">
          <actuator enabled="true" force="15" target_velocity="0"/>
        </axis>
      </axes>
    </joint>
    <joint id="wheel-rear-left:lower-chassis" disable_collisions="true">
      <frames>
        <frame body="wheel-rear-left" position="0,0,0" orientation="0,0,90"/>
        <frame body="lower-chassis" position="-0.031750, 0.02850, 0.01000"
               orientation="0,0,0"/>
      </frames>
      <axes>
        <axis direction="y" mode="angular" range="unconstrained">
          <actuator enabled="true" force="15" target_velocity="0"/>
        </axis>
      </axes>
    </joint>
    <joint id="wheel-rear-right:lower-chassis" disable_collisions="true">
      <frames>
        <frame body="wheel-rear-right" position="0,0,0" orientation="0,0,-90"/>
        <frame body="lower-chassis" position="-0.031750, -0.02850, 0.01000"
               orientation="0,0,0"/>
      </frames>
      <axes>
        <axis direction="y" mode="angular" range="unconstrained">
          <actuator enabled="true" force="15" target_velocity="0"/>
        </axis>
      </axes>
    </joint>
    <joint id="upper-chassis:lower-chassis" disable_collisions="true">
      <frames>
        <frame body="upper-chassis" position="0,0,0" orientation="0,0,0"/>
        <frame body="lower-chassis" position="0,0,0.030000"
               orientation="0,0,0"/>
      </frames>
    </joint>
    <joint id="top-fixture:upper-chassis" disable_collisions="true">
      <frames>
        <frame body="top-fixture" position="0.014545,0,0" orientation="0,0,0"/>
        <frame body="upper-chassis" position="0,0,0.034500"
               orientation="0,0,0"/>
      </frames>
    </joint>
    <joint id="lift-fixture:top-fixture" disable_collisions="true">
      <frames>
        <frame body="lift-fixture" position="0,0,0" orientation="0,0,0"/>
        <frame body="top-fixture" position="0.014545,0,0.010000"
               orientation="0,0,0"/>
      </frames>
    </joint>
    <joint id="lift-fixture:vertical-link" disable_collisions="true">
      <frames>
        <frame body="lift-fixture" position="0.055755,0,-0.0605"
               orientation="0,0,0"/>
        <frame body="vertical-link" position="0,0,0"
               orientation="0,0,0"/>
      </frames>
      <axes>
        <axis direction="z" mode="linear" range="-0.0005:0.1375">
          <actuator enabled="true" force="20" target_velocity="0"/>
        </axis>
      </axes>
    </joint>
    <!-- for visualization purposes -->
    <joint id="vertical-link:visual-link" disable_collisions="true">
      <frames>
        <frame body="vertical-link" position="-0.0122525,0,0.1185"
```

```
                 orientation="0,0,0"/>
          <frame body="visual-link" position="0,0,0" orientation="0,0,0"/>
        </frames>
      </joint>
      <!-- for visualization purposes -->
      <joint id="vertical-link:horizontal-link" disable_collisions="true">
        <frames>
          <frame body="vertical-link" position="0.038000,0,0.036"
                 orientation="0,0,0"/>
          <frame body="horizontal-link" position="0,0,0" orientation="0,0,0"/>
        </frames>
      </joint>
      <joint id="horizontal-link:electromagnet-q1" disable_collisions="true">
        <frames>
          <frame body="horizontal-link" position="-0.023000,0.023000,0"
                 orientation="0,0,0"/>
          <frame body="electromagnet-q1" position="0,0,0" orientation="0,0,0"/>
        </frames>
      </joint>
      <joint id="horizontal-link:electromagnet-q2" disable_collisions="true">
        <frames>
          <frame body="horizontal-link" position="0.023000,0.023000,0"
                 orientation="0,0,0"/>
          <frame body="electromagnet-q2" position="0,0,0" orientation="0,0,0"/>
        </frames>
      </joint>
      <joint id="horizontal-link:electromagnet-q3" disable_collisions="true">
        <frames>
          <frame body="horizontal-link" position="0.023000,-0.023000,0"
                 orientation="0,0,0"/>
          <frame body="electromagnet-q3" position="0,0,0" orientation="0,0,0"/>
        </frames>
      </joint>
      <joint id="horizontal-link:electromagnet-q4" disable_collisions="true">
        <frames>
          <frame body="horizontal-link" position="-0.023000,-0.023000,0"
                 orientation="0,0,0"/>
          <frame body="electromagnet-q4" position="0,0,0" orientation="0,0,0"/>
        </frames>
      </joint>
    </joints>
    <devices>
      <cameras>
        <camera id="duovero_camera" enabled="true" body="vertical-link"
                position="0.056500,0,0.09528" orientation="-90,135,0"
                principle_point="319.5,179.5" range="0.05:0.50"
                focal_length="883.961,883.961" resolution="640,360"
                distortion_parameters="0,0,0"/>
      </cameras>
      <proximity_sensors>
        <sensor body="upper-chassis" range="0.2" direction="-1,0,0"
                offset="-0.044755,0.015,0.01"/>
        <sensor body="upper-chassis" range="0.2" direction="-0.707,0.707,0"
                offset="-0.044755,0.04375,0.01"/>
        <sensor body="upper-chassis" range="0.2" direction="0,1,0"
                offset="-0.015,0.04375,0.01"/>
        <sensor body="upper-chassis" range="0.2" direction="0,1,0"
                offset="0.015,0.04375,0.01"/>
        <sensor body="upper-chassis" range="0.2" direction="0.707,0.707,0"
                offset="0.044755,0.04375,0.01"/>
        <sensor body="upper-chassis" range="0.2" direction="1,0,0"
                offset="0.044755,0.015,0.01"/>
        <sensor body="upper-chassis" range="0.2" direction="1,0,0"
```

```
                offset="0.044755,-0.015,0.01"/>
        <sensor body="upper-chassis" range="0.2" direction="0.707,-0.707,0"
                offset="0.044755,-0.04375,0.01"/>
        <sensor body="upper-chassis" range="0.2" direction="0,-1,0"
                offset="0.0150,-0.04375,0.01"/>
        <sensor body="upper-chassis" range="0.2" direction="0,-1,0"
                offset="-0.0150,-0.04375,0.01"/>
        <sensor body="upper-chassis" range="0.2" direction="-0.707,-0.707,0"
                offset="-0.044755,-0.04375,0.01"/>
        <sensor body="upper-chassis" range="0.2" direction="-1,0,0"
                offset="-0.044755,-0.015,0.01"/>
        <sensor body="horizontal-link" range="0.2" direction="1,0,0"
                offset="0.036000,0,0.007300"/>
        <sensor body="horizontal-link" range="0.2" direction="0,0,-1"
                offset="0.001000,0,0.001000"/>
        <sensor body="vertical-link" range="0.2" direction="1,0,0"
                offset="0.002000,0.016000,0.0071"/>
        <sensor body="vertical-link" range="0.2"  direction="1,0,0"
                offset="0.002000,-0.016000,0.0071"/>
      </proximity_sensors>
      <electromagnets>
        <electromagnet body="electromagnet-q1" passive_field="0,0,165"
                       active_field="0,0,1"/>
        <electromagnet body="electromagnet-q2" passive_field="0,0,165"
                       active_field="0,0,1"/>
        <electromagnet body="electromagnet-q3" passive_field="0,0,165"
                       active_field="0,0,1"/>
        <electromagnet body="electromagnet-q4" passive_field="0,0,165"
                       active_field="0,0,1"/>
      </electromagnets>
      <radios>
        <radio id="radio" body="vertical-link" duplex_mode="half" medium="nfc"
               position="0.002000,0,0.0071" orientation="0,0,0" range="0.02"/>
      </radios>
    </devices>
  </prototype>
```

Bibliography

[1] Federico Augugliaro, Ammar Mirjan, Fabio Gramazio, Matthias Kohler, and Raffaello D'Andrea. "Building Tensile Structures with Flying Machines". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems.* IEEE, 2013, pages 3487–3492.
DOI https://dx.doi.org/10.1109/iros.2013.6696853

[2] Federico Augugliaro, Emanuele Zarfati, Ammar Mirjan, and Raffaello D'Andrea. "Knot-Tying with Flying Machines for Aerial Construction". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems.* IEEE, 2015, pages 5917–5922.
DOI https://dx.doi.org/10.1109/iros.2015.7354218

[3] Frederico Augugliaro, Sergei Lupashin, Michael Hamer, Cason Male, Markus Hehn, Mark W. Mueller, Jan Sebastian Willmann, Fabio Gramazio, Matthias Kohler, and Raffaello D'Andrea. "The Flight Assembled Architecture Installation: Cooperative Construction with Flying Machines". In: *IEEE Control Systems* 34.4 (2014), pages 46–64.
DOI https://dx.doi.org/10.1109/mcs.2014.2320359

[4] Ralph Beckers, Owen E. Holland, and Jean-Louis Deneubourg. "From Local Actions to Global Tasks: Stigmergy and Collective Robotics". In: *Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems.* MIT Press, 1994, pages 181–189.
DOI https://dx.doi.org/10.1007/978-94-010-0870-9_63

[5] Rainer Bischoff, Ulrich Huggenberger, and Erwin Prassler. "KUKA youBot – A Mobile Manipulator for Research and Education". In: *2011 IEEE International Conference on Robotics and Automation.* IEEE, 2011, pages 1–4.
DOI https://dx.doi.org/10.1109/icra.2011.5980575

[6] Adrienne Bolger, Matt Faulkner, David Stein, Lauren White, Seung-kook Yun, and Daniela Rus. "Experiments in Decentralized Robot Construction with Tool Delivery and Assembly Robots". In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems.* IEEE, 2010, pages 5085–5092.
DOI https://dx.doi.org/10.1109/iros.2010.5651495

[7] Eric Bonabeau, Sylvain Guérin, Dominique Snyers, Pascale Kuntz, and Guy Theraulaz. "Three-Dimensional Architectures Grown by Simple 'Stigmergic' Agents".

In: *BioSystems* 56.1 (2000), pages 13–32.
DOI `https://dx.doi.org/10.1016/s0303-2647(00)00067-8`

[8] Eric Bonabeau, Guy Theraulaz, Jean-Louis Deneubourg, Nigel R. Franks, Oliver Rafelsberger, Jean-Louis Joly, and Stéphane Blanco. “A Model for the Emergence of Pillars, Walls and Royal Chambers in Termite Nests”. In: *Philosophical Transactions of the Royal Society of London B: Biological Sciences* 353.1375 (1998), pages 1561–1576.
DOI `https://dx.doi.org/10.1098/rstb.1998.0310`

[9] Michael Bonani, Valentin Longchamp, Stéphane Magnenat, Philippe Rétornaz, Daniel Burnier, Gilles Roulet, Florian Vaussard, Hannes Bleuler, and Francesco Mondada. “The MarXbot, a Miniature Mobile Robot Opening New Perspectives for the Collective-Robotic Research”. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2010, pages 4187–4193.
DOI `https://dx.doi.org/10.1109/iros.2010.5649153`

[10] Manuele Brambilla, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo. “Swarm Robotics: A Review from the Swarm Engineering Perspective”. In: *Swarm Intelligence* 7.1 (2013), pages 1–41.
DOI `https://dx.doi.org/10.1007/s11721-012-0075-2`

[11] Oebele Herman Bruinsma. “An Analysis of Building Behaviour of the Termite *Macrotermes subhyalinus* (Rambur)”. PhD thesis. Netherlands: Wageningen University, 1979.
URL `http://library.wur.nl/WebQuery/clc/107075`

[12] Scott Camazine, Jean-Louis Deneubourg, Nigel R. Franks, James Sneyd, Guy Theraulaz, and Eric Bonabeau. *Self-Organization in Biological Systems*. Princeton University Press, 2001.
URL `http://press.princeton.edu/titles/7104.html`

[13] Changhyun Choi and Daniela Rus. “Probabilistic Visual Verification for Robotic Assembly Manipulation”. In: *2016 IEEE International Conference on Robotics and Automation*. IEEE, 2016, pages 5656–5663.
DOI `https://dx.doi.org/10.1109/icra.2016.7487786`

[14] Anders Lyhne Christensen, Rehan O’Grady, and Marco Dorigo. “Morphology Control in a Multirobot System”. In: *IEEE Robotics & Automation Magazine* 14.4 (2007), pages 18–25.
DOI `https://dx.doi.org/10.1109/m-ra.2007.908970`

[15] Martin Dekan, František Duchoň, Ladislav Jurišica, Anton Vitko, and Andrej Babinec. “iRobot Create Used in Education”. In: *Journal of Mechanics Engineering and Automation* 3.4 (2013), pages 197–202.
DOI `https://dx.doi.org/10.17265/2159-5275/2013.04.002`

[16] Jean-Louis Deneubourg, Simon Goss, Nigel Franks, Ana Sendova-Franks, Claire Detrain, and Laeticia Chrétien. “The Dynamics of Collective Sorting Robot-Like Ants and Ant-Like Robots”. In: *Proceedings of the First International Conference on Simulation of Adaptive Behavior*. MIT Press, 1991, pages 356–363.
URL `https://mitpress.mit.edu/books/animals-animats`

[17] Carrick Detweiler, Marsette Vona, Yeoreum Yoon, Seung-kook Yun, and Daniela Rus. “Self-Assembling Mobile Linkages”. In: *IEEE Robotics & Automation Magazine* 14.4 (2007), pages 45–55.
DOI `https://dx.doi.org/10.1109/m-ra.2007.908971`

[18] Mehmet Dogar, Ross A Knepper, Andrew Spielberg, Changhyun Choi, Henrik I. Christensen, and Daniela Rus. “Multi-Scale Assembly with Robot Teams”. In: *The International Journal of Robotics Research* 34.13 (2015), pages 1645–1659.
DOI `https://dx.doi.org/10.1177/0278364915586606`

[19] Mehmet Dogar, Ross A Knepper, Andrew Spielberg, Changhyun Choi, Henrik I. Christensen, and Daniela Rus. “Towards coordinated precision assembly with robot teams”. In: *The Fourteenth International Symposium on Experimental Robotics*. Springer, 2016, pages 655–669.
DOI `https://dx.doi.org/10.1007/978-3-319-23778-7_43`

[20] Mehmet Dogar, Andrew Spielberg, Stuart Baker, and Daniela Rus. “Multi-Robot Grasp Planning for Sequential Assembly Operations”. In: *2015 IEEE International Conference on Robotics and Automation*. IEEE, 2015, pages 193–200.
DOI `https://dx.doi.org/10.1109/icra.2015.7138999`

[21] Marco Dorigo and Mauro Birattari. “Swarm Intelligence”. In: *Scholarpedia* 2.9 (2007), page 1462.
DOI `https://dx.doi.org/10.4249/scholarpedia.1462`

[22] Nigel R. Franks, Andy Wilby, Bernard Walter Silverman, and Chris Tofts. “Self-Organizing Nest Construction in Ants: Sophisticated Building by Blind Bulldozing”. In: *Animal behaviour* 44.2 (1992), pages 357–375.
DOI `https://dx.doi.org/10.1016/0003-3472(92)90041-7`

[23] Kevin C. Galloway, Rekha Jois, and Mark Yim. “Factory Floor: A Robotically Reconfigurable Construction Platform”. In: *2010 IEEE International Conference on Robotics and Automation*. IEEE, 2010, pages 2467–2472.
DOI `https://dx.doi.org/10.1109/robot.2010.5509878`

[24] Jürgen Gausemeier, Thomas Schierbaum, Roman Dumitrescu, Stefan Herbrechtsmeier, and Alexander Jungmann. “Miniature Robot BeBot: Mechatronic Test Platform for Self-X Properties”. In: *Proceedings of the Ninth International Conference on Industrial Informatics*. IEEE, 2011, pages 451–456.
DOI `https://dx.doi.org/10.1109/indin.2011.6034921`

[25] Pierre-Paul Grassé. "Reconstruction of the Nest and Coordination Between Individuals in Terms. *Bellicositermes Natalensis* and *Cubitermes sp.* The Theory of Stigmergy: Test Interpretation of Termite Constructions". In: *Insectes Sociaux* 6.1 (1959), pages 41–80.
DOI `https://dx.doi.org/10.1007/bf02223791`

[26] Stefan Herbrechtsmeier, Ulf Witkowski, and Ulrich Rückert. "BeBot: A Modular Mobile Miniature Robot Platform Supporting Hardware Reconfiguration and Multi-standard Communication". In: *Progress in Robotics*. Springer, 2009, pages 346–356.
DOI `https://dx.doi.org/10.1007/978-3-642-03986-7_40`

[27] Owen E. Holland and Chris Melhuish. "Stigmergy, Self-organization, and Sorting in Collective Robotics". In: *Artificial Life* 5.2 (1999), pages 173–202.
DOI `https://dx.doi.org/10.1162/106454699568737`

[28] Mong-ying Ani Hsieh. *Task Partitioning for Distributed Assembly*. Visited on May 18, 2017.
URL `http://manihsieh.com/research/task-partitioning/`

[29] Mong-ying Ani Hsieh and Joshua Rogoff. "Complexity Measures for Distributed Assembly Tasks". In: *Proceedings of the 10th Performance Metrics for Intelligent Systems Workshop*. ACM, 2010, pages 97–100.
DOI `https://dx.doi.org/10.1145/2377576.2377594`

[30] Graham Hunt, Faidon Mitzalis, Talib Alhinai, Paul A. Hooper, and Mirko Kovac. "3D Printing with Flying Robots". In: *2014 IEEE International Conference on Robotics and Automation*. IEEE, 2014, pages 4493–4499.
DOI `https://dx.doi.org/10.1109/icra.2014.6907515`

[31] Terry L. Huntsberger, Ashitey Trebi-Ollennu, Hrand Aghazarian, Paul S. Schenker, Paolo Pirjanian, and Hari Das Nayar. "Distributed Control of Multi-Robot Systems Engaged in Tightly Coupled Tasks". In: *Autonomous Robots* 17.1 (2004), pages 79–92.
DOI `https://dx.doi.org/10.1023/b:auro.0000032939.08597.62`

[32] Terry Huntsberger, Paolo Pirjanian, Ashitey Trebi-Ollennu, H Das Nayar, Hrand Aghazarian, Anthony J Ganino, Michael Garrett, Shirish S. Joshi, and Paul S Schenker. "CAMPOUT: A Control Architecture for Tightly Coupled Coordination of Multirobot Systems for Planetary Surface Exploration". In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 33.5 (2003), pages 550–559.
DOI `https://dx.doi.org/10.1109/tsmca.2003.817398`

[33] Terry Huntsberger, Ashley Stroupe, and Brett Kennedy. "System of Systems for Space Construction". In: *2005 IEEE International Conference on Systems, Man, and Cybernetics*. IEEE, 2005, pages 3173–3178.
DOI `https://dx.doi.org/10.1109/icsmc.2005.1571634`

[34] Chris Jones and Maja J. Matarić. "Automatic Synthesis of Communication-Based Coordinated Multi-Robot Systems". In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2004, pages 381–387.
DOI https://dx.doi.org/10.1109/iros.2004.1389382

[35] Chris Jones and Maja J Matarić. "Synthesis and Analysis of Non-Reactive Controllers for Multi-Robot Sequential Task Domains". In: *Experimental Robotics IX*. Springer, 2006, pages 417–426.
DOI https://dx.doi.org/10.1007/11552246_40

[36] István Karsai and Zsolt Pénzes. "Comb Building in Social Wasps: Self-Organization and Stigmergic Script". In: *Journal of Theoretical Biology* 161.4 (1993), pages 505–525.
DOI https://dx.doi.org/10.1006/jtbi.1993.1070

[37] Behrokh Khoshnevis. "Automated Construction by Contour Crafting – Related Robotics and Information Technologies". In: *Automation in Construction* 13.1 (2004), pages 5–19.
DOI https://dx.doi.org/10.1016/j.autcon.2003.08.012

[38] Jung-Hwan Kim and Dylan A. Shell. "Improving the Performance of Self-Organized Robotic Clustering: Modeling and Planning Sequential Changes to the Division of Labor". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pages 4314–4319.
DOI https://dx.doi.org/10.1109/iros.2013.6696975

[39] Jung-Hwan Kim and Dylan A. Shell. "A New Model for Self-Organized Robotic Clustering: Understanding Boundary Induced Densities and Cluster Compactness". In: *2015 IEEE International Conference on Robotics and Automation*. IEEE, 2015, pages 5858–5863.
DOI https://dx.doi.org/10.1109/icra.2015.7140019

[40] Jung-Hwan Kim, Yong Song, and Dylan A. Shell. "Robot Spatial Distribution and Boundary Effects Matter in Puck Clustering". In: *2011 AAAI Spring Symposium Series*. AAAI, 2011, pages 16–21.
URL http://www.aaai.org/ocs/index.php/sss/sss11/paper/view/2492

[41] Ross A. Knepper, Todd Layton, John Romanishin, and Daniela Rus. "IkeaBot: An Autonomous Multi-Robot Coordinated Furniture Assembly System". In: *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pages 855–862.
DOI https://dx.doi.org/10.1109/icra.2013.6630673

[42] Dan Ladley and Seth Bullock. "Logistic Constraints on 3D Termite Construction". In: *Proceedings of the Fourth International Workshop on Ant Colony Optimization and Swarm Intelligence*. Springer, 2004, pages 178–189.
DOI https://dx.doi.org/10.1007/978-3-540-28646-2_16

[43] Dan Ladley and Seth Bullock. “The role of Logistic Constraints in Termite Construction of Chambers and Tunnels”. In: *Journal of Theoretical Biology* 234.4 (2005), pages 551–564.
DOI https://dx.doi.org/10.1016/j.jtbi.2004.12.012

[44] Quentin Lindsey and Vijay Kumar. “Distributed Construction of Truss Structures”. In: *Proceedings of the Tenth Workshop on the Algorithmic Foundations of Robotics*. Springer, 2013, pages 209–225.
DOI https://dx.doi.org/10.1007/978-3-642-36279-8_13

[45] Quentin Lindsey, Daniel Mellinger, and Vijay Kumar. “Construction of Cubic Structures with Quadrotor Teams”. In: *Proceedings of Robotics: Science and Systems*. RSS Foundation, 2011, pages 177–184.
DOI https://dx.doi.org/10.15607/rss.2011.vii.025

[46] Quentin Lindsey, Daniel Mellinger, and Vijay Kumar. “Construction with Quadrotor Teams”. In: *Autonomous Robots* 33.3 (2012), pages 323–336.
DOI https://dx.doi.org/10.1007/s10514-012-9305-0

[47] Quentin Lindsey, Daniel Mellinger, and Vijay Kumar. *Construction with Quadrotor Teams*. Visited on May 19, 2017.
URL https://youtu.be/W18Z3UnnS_0

[48] Felix Lütteke, Xu Zhang, and Jörg Franke. “Implementation of the Hungarian Method for Object Tracking on a Camera Monitored Transportation System”. In: *Proceedings of the Seventh German Conference on Robotics*. VDE, 2012, pages 343–348.
URL https://www.vde-verlag.de/proceedings-en/453418063.html

[49] Marinus Maris and René Boeckhorst. “Exploiting Physical Constraints: Heap Formation Through Behavioral Error in a Group of Robots”. In: *1996 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 1996, pages 1655–1660.
DOI https://dx.doi.org/10.1109/iros.1996.569034

[50] Alcherio Martinoli, Edo Franzi, and Olivier Matthey. “Towards a Reliable Set-Up for Bio-Inspired Collective Experiments with Real Robots”. In: *Experimental Robotics V*. Springer, 1998, pages 595–608.
DOI https://dx.doi.org/10.1007/bfb0112995

[51] Alcherio Martinoli, Auke Jan Ijspeert, and Luca Maria Gambardella. “A Probabilistic Model for Understanding and Comparing Collective Aggregation Mechanisms”. In: *Proceedings of the Fifth European Conference on Artificial Life*. Springer, 1999, pages 575–584.
DOI https://dx.doi.org/10.1007/3-540-48304-7_77

[52] Alcherio Martinoli, Auke Jan Ijspeert, and Francesco Mondada. “Understanding Collective Aggregation Mechanisms: From Probabilistic Modelling to Experiments with Real Robots”. In: *Robotics and Autonomous Systems* 29.1 (1999), pages 51–63.
DOI `https://dx.doi.org/10.1016/s0921-8890(99)00038-x`

[53] Alcherio Martinoli and Francesco Mondada. “Collective and Cooperative Group Behaviours: Biologically Inspired Experiments in Robotics”. In: *Experimental Robotics IV*. Springer, 1997, pages 1–10.
DOI `https://dx.doi.org/10.1007/bfb0035192`

[54] Alcherio Martinoli and Francesco Mondada. “Probabilistic Modeling of a Bio-Inspired Collective Experiments with Real Robots”. In: *Distributed Autonomous Robotic Systems 3*. Springer, 1998, pages 289–298.
DOI `https://dx.doi.org/10.1007/978-3-642-72198-4_28`

[55] John Matson. “Unfree Spirit: NASA’s Mars Rover Appears Stuck for Good”. In: *Scientific American* 302.4 (2010), page 16.
DOI `https://dx.doi.org/10.1038/scientificamerican0410-16a`

[56] Chris Melhuish, Owen E. Holland, and Steve Hoddell. “Collective Sorting and Segregation in Robots with Minimal Sensing”. In: *Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*. MIT Press, 1998, pages 465–470.
URL `https://mitpress.mit.edu/books/animals-animats-5`

[57] Chris Melhuish, Ana B. Sendova-Franks, Sam Scholes, Ian Horsfield, and Fred Welsby. “Ant-Inspired Sorting by Robots: The Importance of Initial Clustering”. In: *Journal of the Royal Society Interface* 3.7 (2006), pages 235–242.
DOI `https://dx.doi.org/10.1098/rsif.2005.0081`

[58] Chris Melhuish, Jason Welsby, and Charles Edwards. *Using Templates for Defensive Wall Building with Autonomous Mobile Ant-Like Robots*. Technical report UMCS-99-3-1. University of Manchester, 1999.
URL `http://apt.cs.manchester.ac.uk/ftp/pub/TR/UMCS-99-3-1.html`

[59] Chris Melhuish, Matt Wilson, and Ana B. Sendova-Franks. *Multi-Object Clustering: Patch Sorting with Simulated Minimalist Robots*. Technical report UMCS-01-4-1. University of Manchester, 2001.
URL `http://apt.cs.manchester.ac.uk/ftp/pub/TR/UMCS-01-4-1.html`

[60] Chris Melhuish, Matt Wilson, and Ana B. Sendova-Franks. “Patch Sorting: Multi-object Clustering Using Minimalist Robots”. In: *Proceedings of the Sixth European Conference on Artificial Life*. Springer, 2001, pages 543–552.
DOI `https://dx.doi.org/10.1007/3-540-44811-x_62`

[61] Olivier Michel. “Webots: Symbiosis Between Virtual and Real Mobile Robots”. In: *Virtual Worlds*. Springer, 1998, pages 254–263.
DOI `https://dx.doi.org/10.1007/3-540-68686-x_24`

[62] Ammar Mirjan, Federico Augugliaro, Raffaello D'Andrea, Fabio Gramazio, and Matthias Kohler. "Building a Bridge with Flying Robots". In: *Robotic Fabrication in Architecture, Art and Design.* Springer, 2016, pages 34–47.
DOI https://dx.doi.org/10.1007/978-3-319-26378-6_3

[63] Ammar Mirjan, Fabio Gramazio, Matthias Kohler, Federico Augugliaro, and Raffaello D'Andrea. "Architectural Fabrication of Tensile Structures with Flying Machines". In: *Green Design, Materials and Manufacturing Processes.* CRC Press, 2013, pages 513–518.
DOI https://dx.doi.org/10.1201/b15002-99

[64] Francesco Mondada, Edoardo Franzi, and Paolo Ienne. "Mobile Robot Miniaturisation: A Tool for Investigation in Control Algorithms". In: *Experimental robotics III.* Springer, 1994, pages 501–513.
DOI https://dx.doi.org/10.1007/bfb0027617

[65] Nils Napp and Eric Klavins. "Robust by Composition: Programs for Multi-Robot Systems". In: *2010 IEEE International Conference on Robotics and Automation.* IEEE, 2010, pages 2459–2466.
DOI https://dx.doi.org/10.1109/robot.2010.5509776

[66] Nils Napp and Eric Klavins. "A Compositional Framework for Programming Stochastically Interacting Robots". In: *The International Journal of Robotics Research* 30.6 (2011), pages 713–729.
DOI https://dx.doi.org/10.1177/0278364911403018

[67] Nils Napp and Eric Klavins. "Load Balancing for Multi-Robot Construction". In: *2011 IEEE International Conference on Robotics and Automation.* IEEE, 2011, pages 254–260.
DOI https://dx.doi.org/10.1109/icra.2011.5979634

[68] Nils Napp and Radhika Nagpal. "Distributed Amorphous Ramp Construction in Unstructured Environments". In: *Robotica* 32.2 (2014), pages 279–290.
DOI https://dx.doi.org/10.1017/S0263574714000113

[69] Nils Napp, Olive R. Rappoli, Jessica M. Wu, and Radhika Nagpal. "Materials and Mechanisms for Amorphous Robotic Construction". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems.* IEEE, 2012, pages 4879–4885.
DOI https://dx.doi.org/10.1109/iros.2012.6385718

[70] Rehan O'Grady, Anders Lyhne Christensen, and Marco Dorigo. "SWARMORPH: Multirobot Morphogenesis Using Directional Self-Assembly". In: *IEEE Transactions on Robotics* 25.3 (2009), pages 738–743.
DOI https://dx.doi.org/10.1109/tro.2008.2012341

[71] Edwin Olson. "AprilTag: A Robust and Flexible Visual Fiducial System". In: *2011 IEEE International Conference on Robotics and Automation.* IEEE, 2011, pages 3400–3407.
DOI https://dx.doi.org/10.1109/icra.2011.5979561

[72] Chris A. C. Parker and Hong Zhang. “Robot Collective Construction by Blind Bulldozing”. In: *2002 IEEE International Conference on Systems, Man and Cybernetics*. IEEE, 2002, pages 59–63.
DOI https://dx.doi.org/10.1109/icsmc.2002.1173385

[73] Chris A. C. Parker and Hong Zhang. “Collective Robotic Site Preparation”. In: *Adaptive Behavior* 14.1 (2006), pages 5–19.
DOI https://dx.doi.org/10.1177/105971230601400101

[74] Christopher A. C. Parker, Hong Zhang, and C Ronald Kube. “Blind Bulldozing: Multiple Robot Nest Construction”. In: *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2003, pages 2010–2015.
DOI https://dx.doi.org/10.1109/iros.2003.1248950

[75] Kirstin Petersen, Radhika Nagpal, and Justin Werfel. “TERMES: An Autonomous Robotic System for Three-Dimensional Collective Construction”. In: *Proceedings of Robotics: Science and Systems*. RSS Foundation, 2011, pages 257–264.
DOI https://dx.doi.org/10.15607/rss.2011.vii.035

[76] Carlo Pinciroli. *The ARGoS User Manual*. Visited on April 26, 2017.
URL http://www.argos-sim.info/user_manual.php

[77] Carlo Pinciroli et al. “ARGoS: A Modular, Multi-engine Simulator for Heterogeneous Swarm Robotics”. In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011, pages 5027–5034.
DOI https://dx.doi.org/10.1109/iros.2011.6094829

[78] Carlo Pinciroli et al. “ARGoS: A Modular, Parallel, Multi-Engine Simulator for Multi-Robot Systems”. In: *Swarm Intelligence* 6.4 (2012), pages 271–295.
DOI https://dx.doi.org/10.1007/s11721-012-0072-5

[79] John W. Romanishin, Kyle Gilpin, Sebastian Claici, and Daniela Rus. “3D M-Blocks: Self-Reconfiguring Robots Capable of Locomotion via Pivoting in Three Dimensions”. In: *2015 IEEE International Conference on Robotics and Automation*. IEEE, 2015, pages 1925–1932.
DOI https://dx.doi.org/10.1109/icra.2015.7139450

[80] John W. Romanishin, Kyle Gilpin, and Daniela Rus. “M-blocks: Momentum-Driven, Magnetic Modular Robots”. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pages 4288–4295.
DOI https://dx.doi.org/10.1109/iros.2013.6696971

[81] Daniela Rus. *DRL Wiki*. Visited on May 18, 2017.
URL http://groups.csail.mit.edu/drl/wiki/index.php

[82] Sam Scholes, Matt Wilson, Ana B. Sendova-Franks, and Chris Melhuish. “Comparisons in Evolution and Engineering: The Collective Intelligence of Sorting”. In: *Adaptive Behavior* 12.3–4 (2004), pages 147–159.
DOI https://dx.doi.org/10.1177/105971230401200302

[83] Touraj Soleymani, Vito Trianni, Michael Bonani, Francesco Mondada, and Marco Dorigo. “Autonomous Construction with Compliant Building Material”. In: *Intelligent Autonomous Systems 13*. Springer, 2016, pages 1371–1388.
DOI https://dx.doi.org/10.1007/978-3-319-08338-4_99

[84] Touraj Soleymani, Vito Trianni, Michael Bonani, Francesco Mondada, and Marco Dorigo. *Bio-inspired Construction with Mobile Robots and Compliant Pockets*. Visited on May 18, 2017.
URL http://iridia.ulb.ac.be/supp/IridiaSupp2015-003/index.html

[85] Yong Song, Jung-Hwan Kim, and Dylan A. Shell. “Self-Organized Clustering of Square Objects by Multiple Robots”. In: *Proceedings of the Eighth International Conference on Swarm Intelligence*. Springer, 2012, pages 308–315.
DOI https://dx.doi.org/10.1007/978-3-642-32650-9_32

[86] David Stein, Timothy Ryan Schoen, and Daniela Rus. “Constraint-Aware Coordinated Construction of Generic Structures”. In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011, pages 4803–4810.
DOI https://dx.doi.org/10.1109/iros.2011.6094682

[87] Robert L. Stewart and R. Andrew Russell. “Building a Loose Wall Structure with a Robotic Swarm Using a Spatio-Temporal Varying Template”. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2004, pages 712–716.
DOI https://dx.doi.org/10.1109/iros.2004.1389436

[88] Robert L. Stewart and R. Andrew Russell. “A Distributed Feedback Mechanism to Regulate Wall Construction by a Robotic Swarm”. In: *Adaptive Behavior* 14.1 (2006), pages 21–51.
DOI https://dx.doi.org/10.1177/105971230601400104

[89] Robert L. Stewart, R. Andrew Russell, and Lindsay Kleeman. “Modelling a Deposition Process in Collective Construction”. In: *Turkish Journal of Electrical Engineering & Computer Sciences* 15.2 (2007), pages 227–255.
URL https://journals.tubitak.gov.tr/elektrik/abstract.htm?id=8830

[90] Ashley Stroupe, Terry Huntsberger, Avi Okon, and Hrand Aghazarian. “Precision Manipulation with Cooperative Robots”. In: *Multi-Robot Systems. From Swarms to Intelligent Automata Volume III*. Springer, 2005, pages 235–248.
DOI https://dx.doi.org/10.1007/1-4020-3389-3_19

[91] Ashley Stroupe, Terry Huntsberger, Avi Okon, Hrand Aghazarian, and Matthew Robinson. “Behavior-Based Multi-Robot Collaboration for Autonomous Construction Tasks”. In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2005, pages 1989–1994.
DOI https://dx.doi.org/10.1109/iros.2005.1545269

[92] Ashley Stroupe, Avi Okon, Matthew Robinson, Terry Huntsberger, Hrand Aghazarian, and Eric Baumgartner. “Sustainable Cooperative Robotic Technologies for Human and Robotic Outpost Infrastructure Construction and Maintenance”. In: *Autonomous Robots* 20.2 (2006), pages 113–123.
DOI `https://dx.doi.org/10.1007/s10514-006-5943-4`

[93] Ken Sugawara and Yohei Doi. “Collective Construction of Dynamic Structure Initiated by Semi-Active Blocks”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2015, pages 428–433.
DOI `https://dx.doi.org/10.1109/iros.2015.7353408`

[94] Ken Sugawara and Yohei Doi. “Collective Construction by Cooperation of Simple Robots and Intelligent Blocks”. In: *Intelligent Robotics and Applications*. Springer, 2016, pages 452–461.
DOI `https://dx.doi.org/10.1007/978-3-319-43506-0_40`

[95] Ken Sugawara and Yohei Doi. “Collective Construction of Dynamic Equilibrium Structure Through Interaction of Simple Robots with Semi-Active Blocks”. In: *Distributed Autonomous Robotic Systems 12*. Springer, 2016, pages 165–176.
DOI `https://dx.doi.org/10.1007/978-4-431-55879-8_12`

[96] Andry Tanoto, Ulrich Rückert, and Ulf Witkowski. “Teleworkbench: A Teleoperated Platform for Experiments in Multi-robotics”. In: *Web-Based Control and Robotics Education*. Springer, 2009, pages 267–296.
DOI `https://dx.doi.org/10.1007/978-90-481-2505-0_12`

[97] Andry Tanoto, Ulrich Rückert, and Ulf Witkowski. *Heinz Nixdorf Institut: Teleworkbench*. Visited on May 24, 2017.
URL `https://www.hni.uni-paderborn.de/system-and-circuit-technology/projects/abgeschlossene-projekte/teleworkbench/`

[98] Guy Theraulaz and Eric Bonabeau. “Modelling the Collective Building of Complex Architectures in Social Insects with Lattice Swarms”. In: *Journal of Theoretical Biology* 177.4 (1995), pages 381–400.
DOI `https://dx.doi.org/10.1006/jtbi.1995.0255`

[99] Guy Theraulaz and Eric Bonabeau. “A Brief History of Stigmergy”. In: *Artificial Life* 5.2 (1999), pages 97–116.
DOI `https://dx.doi.org/10.1162/106454699568700`

[100] Bernhard Thomaszewski, Andreas Gumann, Simon Pabst, and Wolfgang Straßer. “Magnets in Motion”. In: *ACM Transactions on Graphics* 27.5 (2008), 162:1–162:9.
DOI `https://dx.doi.org/10.1145/1409060.1409115`

[101] Andrew Vardy. “Accelerated Patch Sorting by a Robotic Swarm”. In: *The Ninth Conference on Computer and Robot Vision*. IEEE, 2012, pages 314–321.
DOI `https://dx.doi.org/10.1109/crv.2012.48`

[102] Andrew Vardy, Gregory Vorobyev, and Wolfgang Banzhaf. "Cache Consensus: Rapid Object Sorting by a Robotic Swarm". In: *Swarm Intelligence* 8.1 (2014), pages 61–87.
DOI `https://dx.doi.org/10.1007/s11721-014-0091-5`

[103] Sean Verret, Hong Zhang, and MQ-H Meng. "Collective Sorting with Local Communication". In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2004, pages 2687–2692.
DOI `https://dx.doi.org/10.1109/iros.2004.1389814`

[104] Gregory Vorobyev, Andrew Vardy, and Wolfgang Banzhaf. "Conformity and Nonconformity in Collective Robotics: A Case Study". In: *Proceedings of the Twelfth European Conference on Artificial Life*. MIT Press, 2013, pages 981–988.
DOI `https://dx.doi.org/10.7551/978-0-262-31709-2-ch146`

[105] Gregory Vorobyev, Andrew Vardy, and Wolfgang Banzhaf. "Supervised Learning in Robotic Swarms: From Training Samples to Emergent Behavior". In: *Distributed Autonomous Robotic Systems 11*. Springer, 2014, pages 435–448.
DOI `https://dx.doi.org/10.1007/978-3-642-55146-8_31`

[106] Jue Wang, Fadel Adib, Ross Knepper, Dina Katabi, and Daniela Rus. "RF-Compass: Robot Object Manipulation Using RFIDs". In: *Proceedings of the Nineteenth Annual International Conference on Mobile Computing & Networking*. ACM, 2013, pages 3–14.
DOI `https://dx.doi.org/10.1145/2500423.2500451`

[107] Jens Wawerla, Gaurav S. Sukhatme, and Maja J. Matarić. "Collective Construction with Multiple Robots". In: *2002 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2002, pages 2696–2701.
DOI `https://dx.doi.org/10.1109/irds.2002.1041677`

[108] Justin Werfel. "Building Blocks for Multi-robot Construction". In: *Distributed Autonomous Robotic Systems 6*. Springer, 2007, pages 285–294.
DOI `https://dx.doi.org/10.1007/978-4-431-35873-2_28`

[109] Justin Werfel. "Robot Search in 3D Swarm Construction". In: *First International Conference on Self-Adaptive and Self-Organizing Systems*. IEEE, 2007, pages 363–366.
DOI `https://dx.doi.org/10.1109/saso.2007.45`

[110] Justin Werfel, Yaneer Bar-Yam, and Radhika Nagpal. "Building Patterned Structures with Robot Swarms". In: *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 2005, pages 1495–1502.
URL `http://www.ijcai.org/proceedings/2005`

[111] Justin Werfel, Yaneer Bar-Yam, Daniela Rus, and Radhika Nagpal. "Distributed Construction by Mobile Robots with Enhanced Building Blocks". In: *2006 IEEE International Conference on Robotics and Automation*. IEEE, 2006, pages 2787–2794.
DOI `https://dx.doi.org/10.1109/robot.2006.1642123`

[112] Justin Werfel, Donald Ingber, and Radhika Nagpal. “Collective Construction of Environmentally-Adaptive Structures”. In: *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2007, pages 2345–2352.
DOI `https://dx.doi.org/10.1109/iros.2007.4399462`

[113] Justin Werfel and Radhika Nagpal. “Extended stigmergy in collective construction”. In: *IEEE Intelligent Systems* 21.2 (2006), pages 20–28.
DOI `https://dx.doi.org/10.1109/mis.2006.25`

[114] Justin Werfel and Radhika Nagpal. “Three-Dimensional Construction with Mobile Robots and Modular Blocks”. In: *The International Journal of Robotics Research* 27.3–4 (2008), pages 463–479.
DOI `https://dx.doi.org/10.1177/0278364907084984`

[115] Justin Werfel, Kirstin Petersen, and Radhika Nagpal. “Designing Collective Behavior in a Termite-Inspired Robot Construction Team”. In: *Science* 343.6172 (2014), pages 754–758.
DOI `https://dx.doi.org/10.1126/science.1245842`

[116] Jan Sebastian Willmann, Federico Augugliaro, Thomas Cadalbert, Raffaello D’Andrea, Fabio Gramazio, and Matthias Kohler. “Aerial Robotic Construction Towards a New Field of Architectural Research”. In: *International Journal of Architectural Computing* 10.3 (2012), pages 439–460.
DOI `https://dx.doi.org/10.1260/1478-0771.10.3.439`

[117] Matt Wilson, Chris Melhuish, and Ana B. Sendova-Franks. “Multi-Object Segregation: Ant-Like Brood Sorting Using Minimalist Robots”. In: *Proceedings of the Seventh International Conference on Simulation of Adaptive Behavior*. MIT Press, 2002, pages 369–370.
URL `https://mitpress.mit.edu/books/animals-animats-7`

[118] Matt Wilson, Chris Melhuish, Ana B. Sendova-Franks, and Samuel Scholes. “Algorithms for Building Annular Structures with Minimalist Robots Inspired by Brood Sorting in Ant Colonies”. In: *Autonomous Robots* 17.2 (2004), pages 115–136.
DOI `https://dx.doi.org/10.1023/b:auro.0000033969.52486.3d`

[119] James Worcester and Mong-ying Ani Hsieh. “Task Partitioning via Ant Colony Optimization for Distributed Assembly”. In: *Proceedings of the Eighth International Conference on Swarm Intelligence*. Springer, 2012, pages 145–155.
DOI `https://dx.doi.org/10.1007/978-3-642-32650-9_13`

[120] James Worcester, Mong-ying Ani Hsieh, and Rolf Lakaemper. “Distributed Assembly with Online Workload Balancing and Visual Error Detection and Correction”. In: *The International Journal of Robotics Research* 33.4 (2014), pages 534–546.
DOI `https://dx.doi.org/10.1177/0278364913509125`

[121] James Worcester, Rolf Lakaemper, and Mong-ying Ani Hsieh. “3-Dimensional Tiling for Distributed Assembly by Robot Teams”. In: *The Thirteenth International Symposium on Experimental Robotics*. Springer, 2013, pages 143–154.
DOI https://dx.doi.org/10.1007/978-3-319-00065-7_11

[122] James Worcester, Joshua Rogoff, and Mong-ying Ani Hsieh. “Constrained Task Partitioning For Distributed Assembly”. In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011, pages 4790–4796.
DOI https://dx.doi.org/10.1109/iros.2011.6095046

[123] Yeoreum Yoon and Daniela Rus. “Shady3D: A Robot That Climbs 3D Trusses”. In: *2007 IEEE International Conference on Robotics and Automation*. IEEE, 2007, pages 4071–4076.
DOI https://dx.doi.org/10.1109/robot.2007.364104

[124] Seung Kook Yun and Daniela Rus. “Optimal Distributed Planning for Self Assembly of Modular Manipulators”. In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2008, pages 1346–1352.
DOI https://dx.doi.org/10.1109/iros.2008.4651127

[125] Seung Kook Yun and Daniela Rus. “Optimal Self Assembly of Modular Manipulators with Active and Passive Modules”. In: *Autonomous Robots* 31.2 (2011), pages 183–207.
DOI https://dx.doi.org/10.1007/s10514-011-9236-1

[126] Seung Kook Yun and Daniela Rus. “Adaptive Coordinating Construction of Truss Structures Using Distributed Equal-Mass Partitioning”. In: *IEEE Transactions on Robotics* 30.1 (2014), pages 188–202.
DOI https://dx.doi.org/10.1109/tro.2013.2279643

[127] Seung-kook Yun, David Alan Hjelle, Hod Lipson, and Daniela Rus. “Planning the Reconfiguration of Grounded Truss Structures with Truss Climbing Robots That Carry Truss Elements”. In: *2009 IEEE International Conference on Robotics and Automation*. IEEE, 2009, pages 1327–1333.
DOI https://dx.doi.org/10.1109/robot.2009.5152714

[128] Seung-kook Yun and Daniela Rus. “Optimal Distributed Planning of Multi-Robot Placement on a 3D Truss”. In: *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2007, pages 1365–1370.
DOI https://dx.doi.org/10.1109/iros.2007.4399176

[129] Seung-kook Yun and Daniela Rus. “Adaptation to Robot Failures and Shape Change in Decentralized Construction”. In: *2010 IEEE International Conference on Robotics and Automation*. IEEE, 2010, pages 2451–2458.
DOI https://dx.doi.org/10.1109/robot.2010.5509737

[130] Seung-kook Yun, Mac Schwager, and Daniela Rus. “Coordinating Construction of Truss Structures Using Distributed Equal-Mass Partitioning”. In: *Proceedings of the Fourteenth International Symposium of Robotic Research*. Springer, 2011, pages 607–623.
DOI `https://dx.doi.org/10.1007/978-3-642-19457-3_36`

[131] Zhengyou Zhang. “Microsoft Kinect Sensor and Its Effect”. In: *IEEE MultiMedia* 19.2 (2012), pages 4–10.
DOI `https://dx.doi.org/10.1109/mmul.2012.24`