AUTONOMOUS CONSTRUCTION OF THREE-DIMENSIONAL STRUCTURES USING SELF-ACTUATED PARTS:

A layered self-assembly approach

CONSTRUCTION AUTONOME DE STRUCTURES TRIDIMENSIONNELLES À L'AIDE DE PIÈCES AUTO-ACTIONNÉES:

Un approche d'auto-assemblage par couches

A Thesis Submitted to the Division of Graduate Studies
of the Royal Military College of Canada
by

Kleber Macedo Cabral

In Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

September, 2022

To my wife, sons, mom, dad, and brother. You are my reason to keep moving forward.

Acknowledgements

I would first like to thank my wife, Kamylla, my parents, Genilda and Daniel, my sons Guilherme and Mateus, my mother-in-law Aurizete, my aunt Gisela, and all my family and friends for the support you have always provided me. I would also like to thank Dr. Sidney Givigi and Dr. Peter Jardine for their support and guidance in completing this thesis. It was an immense pleasure and I am really grateful to have met such kind, productive, and smart people in my academic life. I would also like to thank my professors, supervisors, and colleagues from all stages of my academic endeavour. Thank you for the inspiration, guidance, support and friendship. In addition, I'd like to thank the staff of the Electrical and Computer Engineering Department at the Royal Military College of Canada (RMC) for all their help and support. Finally, I would like to thank Defence Research and Development Canada and RMC for providing such a great opportunity.

Abstract

Autonomous construction of structures is a broad area of research and includes the use of robots to transfer structural materials or robots serving as part of the structure themselves. The latter is commonly referred to as self-assembly. Motivated by the challenges that design and execution of the assembly process entails, this thesis focuses on the self-assembly of three-dimensional structures. We propose a novel design of the system architecture for executing the assembly, which establishes the structural elements and information flow for the system. The self-assembly process occurs in three stages: moving, ordering, and placing. Moving deals with the transit of autonomous parts from a deployment position to the assembly position and network control methods are used to control the movement of the robots. Once the assembly location is reached, robots move to the structure one at a time, in order to avoid collisions, and take their place within the structure. Ordering represents the stage where robots compute the sequence at which individual agents will arrive at the structure assembly. An auction-based assignment method is used to order robots. Finally, the placing stage deals with how the robot identifies an empty position in the structure to place itself based on local information only. A set of behaviours are proposed for each robot that searches for an empty position given a predefined structure blueprint. Our results show that the proposed graph-based methods, auction processes, and programmed behaviours allow robots to execute the three-dimensional assembly of the structure faster, using less energy, and with less communication bandwidth when compared to baseline methods. The efficacy of the algorithms proposed were validated using real-world experiments. The structure parts used are micro quadrotors fixed inside a light-weight cubic frame. The assembly of two stair-shaped structures took place in a controlled environment and a motion capture system was responsible for computing robotic states.

Résumé

La construction autonome de structures est un vaste domaine de recherche et comprend l'utilisation de robots pour transporter des matériaux structurels ou des robots eux-mêmes en tant que composant de la structure. Cette dernière est communément appelée autoassemblage. Motivée par les défis que représente la conception et l'exécution du processus d'assemblage autonome, cette thèse porte sur l'autoassemblage de structures tridimensionnelles. Nous proposons une nouvelle conception de l'architecture du système qui établit ses éléments et ses flux d'informations. Le processus d'autoassemblage se déroule en trois étapes: déplacement, ordre et placement. Le déplacement traite du transit de parties autonomes d'un poste de déploiement à un poste d'assemblage et les méthodes de contrôle en réseau sont utilisées pour contrôler le mouvement des robots. Une fois le lieu d'assemblage atteint, les robots se déplacent un par un vers la structure, afin d'éviter les collisions, et prennent place à l'intérieur de la structure. L'ordre représente l'étape où les robots calculent la séquence à laquelle les agents individuels arriveront à l'assemblage de la structure. Une méthode d'affectation des tâches basée sur les enchères est programmée dans les robots. Enfin, l'étape de placement traite de la façon dont le robot identifie une position vide dans la structure pour se placer en se basant uniquement sur des informations locales. Un ensemble de comportements est proposé pour chaque robot qui recherche une position vide en fonction du plan de structure prédéfini. Nos résultats montrent que les méthodes basées sur les graphes, les processus d'enchères et les comportements programmés permettent aux robots d'exécuter l'assemblage de structures tridimensionnelles plus rapidement, en utilisant moins d'énergie et avec moins de bande passante de communication par rapport aux méthodes de base. L'efficacité des algorithmes proposés a été validée à l'aide d'expériences réelles. Les éléments de structure utilisés sont des micro quadrotors fixés à l'intérieur d'un cadre cubique léger. L'assemblage de deux structures en forme d'escalier a eu lieu dans un environnement contrôlé et un système de capture de mouvement était chargé de calculer les états robotiques.

Contents

Acknowledgements **iii**

Abstract **iv**

Résumé **v**

List of Tables **ix**

List of Figures **x**

List of Acronyms **xii**

List of Symbols **xiii**

1 Introduction **1**

1.1 Problem definition 2
1.2 Motivation 3
1.3 Objectives 3
1.4 Contributions 4
1.5 Publications from this thesis 5
1.6 Outline 5
1.7 Notation 6

2 Literature background **7**

2.1 Autonomous construction strategies 7
2.2 Robots as structure parts 9
2.3 Structure modelling 11
2.4 Control of robots in motion 12
2.5 Assembly order and task assignment 13

3 Self-assembly fundamentals and formulation **16**

3.1 Self-assembly system architecture 17
3.2 Planning 19
3.3 Control 20
3.4 Robot - Self-actuated Construction Block 21
3.4.1 Quadrotor modelling 23
3.4.2 Robot's simplified model 24
3.5 Structure blueprint and assembly properties 25
3.6 Remarks 29

4 Movement of robots from start location to assembly location 31
4.1 Preliminaries 32
4.1.1 Problem Formulation 33
4.1.2 Swarm control law 33
4.2 Decentralized control strategy for swarm motion 36
4.2.1 Connectivity Metric and Active Group Separation . . . 38
4.2.2 Selection of New Pins 39
4.2.3 Operators and RRT algorithm 40
4.2.4 Virtual Nodes and Control Law 42
4.3 Results 43
4.3.1 Swarm guidance 45
4.3.2 Comparison between pin selection methods 47
4.4 Conclusion 51

5 Ordering of robots to assemble structures 53
5.1 Problem Formulation 54
5.2 Auction-Based Assignment 57
5.3 Algorithm's analysis 60
5.4 Results 62
5.5 Conclusion 65

6 Self-assembly of three-dimensional structures 67
6.1 Problem Formulation 68
6.2 Self-Assembly 70
6.2.1 Robots' behaviours 71
6.2.2 Self assembly algorithm proofs 81
6.3 Results 84
6.3.1 Simulation setup 84
6.3.2 Guidelines for performance analysis 85
6.3.3 Simulation trials 86
6.3.4 Seed selection and assembly performance 86

6.3.5 Assembly of more complex structures 91
6.3.6 Validation using embedded sensors for navigation 91
6.3.7 Experimental results 94
6.4 Conclusion 95

7 Conclusion 98
7.1 Moving of parts in space 99
7.2 Ordering and task assignment 100
7.3 Placement using local information 100
7.4 Future work 101

Bibliography 104

List of Tables

4.1 Parameters for simulation 43
4.2 Tracking error for different pin selection methods 50

5.1 Parameters for simulation 65
5.2 Average battery decrease while a robot wait to be assigned 65

6.1 Movements performed by robots during assembly 88

List of Figures

3.1 Description of the stages of the layered self-assembly process discussed in this thesis. 17
3.2 High-level description of an autonomous construction system. Source [1]. 18
3.3 Construction block using the Crazyflie quadrotor. 22
3.4 VICON system setup. 22
3.5 The relationship between the coordinate frames used in this work. 27
3.6 Structure and its blueprint with grid lines. The blue circles represent desired positions to place a block, P , the red dots are external positions, $\bar{P}$, and layers are indicated as P_l 28

4.1 Swarm control strategy. 38
4.2 Example of possible initial configurations, pin selection method comparison. 45
4.3 Autonomous guidance using 10 agents, $\#operators = 3$, and $N^{min} = 3$. 46
4.4 Connectivity metrics during part of the experiment. 47
4.5 Minimal group avoiding obstacles and maintaining cohesion. 48
4.6 Tracking error of the geometric center using two different pin selection methods. 49
4.7 Control effort of the pin node using two different pin selection methods. 49
4.8 Result of 100 iterations using the two pin selection methods. . . . 50

5.1 Connections in the graph showing the start topology (left side), the computation of the tree and the leaf nodes (right side). 58
5.2 The stages of the assignment process. 63
5.3 Battery usage distributions. 64
5.4 Comparison between assignment strategies with $\bar{bat}_R$ values for the three experiments. 65

6.1 Local sensing of a robot with a top view perspective 69
6.2 Graph view of a 4 × 4 cube with seed locations highlighted. Each layer G_l in the graph contains only the edges in *black*. *Red* edges between seeds represent the shortcuts taken during assembly, when a robot moves to upper layers. 75
6.3 A newly arrived block inquires the seed if it should circle the current layer. Once the layer is already full, the seed denies it and the robot will apply $MOVE.EXT(r, \boldsymbol{\rho}_{s_2})$ to go to the next layer's seed. 76
6.4 Top view: A robot (blue) circling the already placed blocks (grey) in the layer until it reaches an empty position (green). The green arrows describe $CO(\boldsymbol{\rho}_{s_l})$ and $CO(\boldsymbol{\rho}_{11})$. The position to be circled, $\boldsymbol{\rho}_c$, starts as $\boldsymbol{\rho}_{s_l}$ and changes to $\boldsymbol{\rho}_{11}$ as the robot reaches $\boldsymbol{\rho}_{10}$. . . . 78
6.5 A robot that is at an internal position evaluates neighbouring positions $N(\boldsymbol{\rho}_r)^{empty}$ to determine if and where to move. 81
6.6 Exploded view of the structures assembled highlighting each layer. 85
6.7 The assembly of the chair with robots moving clockwise. 87
6.8 The distance traveled by each robot assembling the chair. 87
6.9 The relationship between the closeness and the NCG metrics with the average distance travelled by robots when executing the proposed assembly strategy. 90
6.10 The assembly of a bridge through multiple executions of the proposed self-assembly approach. 92
6.11 Robots performing navigation and self-assembly using embedded sensors. 93
6.12 Robots performing navigation and self-assembly using embedded sensors. 94
6.13 Assembly of stairs with six robots. 96
6.14 Assembly of stairs with twelve robots. 97

List of Acronyms

MDP - Markov Decision Process
MRS - Multi-Robot System
MST - Minimum Spanning Tree
NCG - Network Controllability Gramian
PID - Proportional–Integral–Derivative
RRT - Rapidly-Expanding Random Tree
SA - Self-Assembly
UAV - Unmanned Aerial Vehicle

List of Symbols

$\boldsymbol{a}$ - acceleration vector in the inertial frame;
a_{ij} - elements of a matrix $\mathbf{A}$;
a_{ij}^{p} - adjacency relationship between agents;
$\boldsymbol{b}_j$ - input column vector on NCG computation;
$b_{i,k}^{p}$ - adjacency relationship between agents and obstacles;
$bat_i^{initial}$ - robot i battery level at the beginning of the ordering stage;
bat_i^{final} - robot i battery level at the end of the ordering stage (arriving at the structure location);
$\bar{bat}_R$ - average battery usage among robots in a single assignment run;
$\bar{bat}_{use}$ - mean value of $\bar{bat}_R$ after multiple iterations with different initial conditions;
c_{ij} - score function used during auction;
$c(v_i)$ - closeness metric of vertex v_i;
d_i - number of neighbours of robot i;
$d_{sp}(v_i, v_j)$ - cost of traversing the graph in the shortest path between vertices v_i and v_j;
$\text{dist}(\boldsymbol{\rho}_i, \boldsymbol{\rho}_j)$ - Manhattan distance;
$d_{\alpha\alpha}$ - desired distance between agents while moving;
d_{α} - σ-norm of $d_{\alpha\alpha}$;
$d_{\alpha\beta}$ - allowed distance to an obstacle (α-agent to β-agent);
d_{β} - σ-norm of $d_{\alpha\beta}$;
e_{ij} - edge connecting vertex i to vertex j;
f_m - force exerted by a motor in the direction of Z_q;
f_{all} - total exerted force in the Z_q axis;
g - gravitational constant;
h_{ij} - number of hops in the network between two nodes;
l_{arm} - distance between the center of mass of the quadrotor and the axis of rotation of the rotors;
m_i - connectivity metric on robot i;
m_a^{na} - message from auctioneer to non-auctioneer robots;

m_{na}^{a} - message from non-auctioneer robots to auctioneer;
$\mathrm{m}_{\mathsf{s}}^{a}$ - message from a structure on the ground to the auctioneer to request a new part;
m_{r}^{l} - message from a moving robot to the seed of a layer l;
m_{l}^{r} - message from the seed of a layer l to a moving robot;
m_r - number of movements of each robot in the placing stage;
$max(m_r)$ - the maximum number of movements in the placing stage;
$\boldsymbol{p}$ - position of blocks described in the inertial frame;
$\hat{\boldsymbol{p}}_{i,k}$ - position of a β-agent;
$\boldsymbol{p}_{o}^{k}$ - position of obstacle k;
$\boldsymbol{p}_{vn}^{r}$ - position of the virtual node in a minimal group;
r_{o}^{k} - obstacle radius;
r_{α} - robot's range of sensing/communication in the moving stage;
r_{α} - σ-norm of the sensing/communication range between agents;
r_{β} - interaction range (i.e. detection range) between robot and obstacle;
r_{β} - σ-norm of the obstacles detection range;
$\boldsymbol{u}_i$ - control signal of a robot i (acceleration) in the moving stage;
$\boldsymbol{u}_{i}^{c}$ - cohesion control term;
$\boldsymbol{u}_{i}^{\alpha}$ - control signal given interaction with other robots;
$\boldsymbol{u}_{i}^{\beta}$ - control signal given obstacle avoidance;
$\boldsymbol{u}_{i}^{\gamma}$ - control signal given the collective objective;
$\boldsymbol{v}$ - velocity of blocks described in the inertial frame;
$\hat{\boldsymbol{v}}_{i,k}$ - velocity of a β-agent;
$\boldsymbol{v}_{vn}^{r}$ - velocity of the virtual node in a minimal group;
$\mathbf{A}$ - adjacency matrix of a graph G;
$\mathbf{A}_{dyn}$ - network dynamics matrix;
$CO(\boldsymbol{\rho}_c)$ - cyclic order of $\boldsymbol{\rho}_c$;
D - dictionary containing the position and blueprint for every structure in $\mathbb{S}$;
E - edge set in a graph;
G - graph symbol;
H - control horizon to which the Gramian is computed;
$\mathcal{I}$ - set of robots in the task assignment process;
$\mathcal{J}$ - set of tasks in the task assignment process;
L - number of layers of a structure;
N - number of robots in a structure;
N_t - total number of tasks generated by an assembly process with N_T blocks;
N_R - total number of robots to assemble all structures;

N_i^α - set of robots connected to node i in the graph topology, G in the moving stage;
N_i^β - set of β-agents close to node i;
N_e - number of simulation runs with different initial conditions in the ordering stage;
N_O - number of obstacle in the environment in the moving stage;
$N(\boldsymbol{\rho}_r)$ - set of internal adjacent positions;
$N(\boldsymbol{\rho}_r)^{empty}$ - set of internal adjacent positions that are currently empty;
N^{min} - minimum number of nodes allowed in a subgroup of agents;
N^{SG} - number of robots in a V_{SG} group;
O_I - inertial reference frame;
O_b - block reference frame;
O_q - quadrotor reference frame;
O_B - blueprint reference frame;
P - set of desired block positions in the blueprint, *internal positions*;
$\bar{P}$ - set of positions that should remain empty in the blueprint, *external positions*;
Q - Queue containing parts requests (positions of structure executing the request);
R - set of robots;
R^a - set of robots already assigned to a structure;
R^u - set of unassigned (available) robots;
$\mathbf{R}_q^I$ the rotation matrix from O_q to O_I
S - set of seeds in the structure graph;
$\mathbb{S}$ - set of structures being assembled simultaneously;
V - Set of vertices in a graph;
V_{SG} - set containing a group of agents in the swarm;
$\mathbf{W}_j$ - Network Controllability Gramian;
X - coordinate axis;
Y - coordinate axis;
Z - coordinate axis;
Δbat_i - battery usage of robot i in the ordering stage;
θ - pitch angle of the robot;
ι_{xx} - inertia moment of the quadrotor with respect to the axis X_q;
ι_{yy} - inertia moment of the quadrotor with respect to the axis Y_q;
ι_{zz} - inertia moment of the quadrotor with respect to the axis Z_q;
μ_m - average number of movements for all robots in the placing stage;
ρ - position in the structure coordinate frame O_B;
σ_m - standard deviation of the movements of all robots in the placing stage;

τ_x - torque around X_q axis;
τ_y - torque around Y_q axis;
τ_z - torque around Z_q axis;
ϕ - roll angle of the robot;
ψ - yaw angle of the robot;
ω_x - angular velocity around axes X_q;
ω_y - angular velocity around axes Y_q;
ω_z - angular velocity around axes Z_q;
Γ - minimum spanning tree computed over a connected graph G;

1 Introduction

Despite recent advances in autonomous robotics, construction remains mainly human-based process. Some tasks are aided by automation (e.g. computer-aided design tools), but true autonomous construction is yet to be realized at scale. There are still a variety of applications in which human action is undesirable or impossible [2, 3], such as carrying heavy loads or performing assembly in unsafe settings. This drives the usage and development of more autonomous, scalable, and safe tools and methods for the construction process.

The use of robots to transport, manipulate, and place structural material is a step towards autonomy in construction. Wheeled land vehicles [4, 5] and aerial vehicles [6, 7] are two types of robotic platforms that have been used to manipulate the structural components. Different aspects to the autonomous construction can be studied given the robots used and also the characteristics of structures being assembled. The technologies, methods, and algorithms involved in the construction process are a vast research area. In addition, how different shapes of pieces and final structure impact the construction process can be investigated.

An approach that has recently being developed is the use of self-actuated pieces as structure parts [8, 9, 1]. It considers that the parts are intelligent and autonomous robots, thus the construction becomes the process of guiding each robot to a place in the structure. Such assembly has been studied on a two-dimensional environment, using boats to assemble bridges on water's surface [8], and to build three-dimensional structures using small quadrotors [1].

The autonomous structure parts can be programmed to spontaneously organize themselves into complex structures [10]. Such construction process is called self-assembly and may involve large-scale structures containing multiple parts that are assembled entirely through local interactions and sensing [11]. These assembly processes could be based on a central controller architecture [12] or decentralized control methods [11, 13, 10]. In decentralized approaches, decision is taken locally at every interaction between robots and following specific rules. Designing the proper set of rules that governs the robot's behaviour is the main challenge of self-assembly strategies. These rules are usually programmed at the robot's internal computer, and are executed until the system comes to a halt, when the assembly process finishes.

By definition, if parts are themselves robots, then the assembly process is being executed by a Multi-Robot System (MRS). Different challenges arise when multiple robots are operating simultaneously, such as how to cooperatively and orderly move robots in space, how to plan for robot's trajectories, and how to compute the sequence of placement of parts on the structure. In the literature, complex networks are used to describe real-world systems with multiple elements, such electrical power grids, the internet structure etc [14]. Complex networks are also used to describe MRSs [15, 16]. Moreover, network control theory can be used to control robot navigation [17], model communication between robots [16] or model sensing [18]. Graphs are also used to model the relationship between structure parts in autonomous construction problems [10, 19].

In this thesis, the use of self-assembly methods that promote directed growth using self-actuated parts is investigated. To construct three-dimensional structures, first, robots must move in an organized manner from start position to the assembly location in space. Second, robots coordinate which one of them will move to the desired structure location at any given time. Lastly, self-assembly algorithms must guide the robots to assemble the structure without the need of a central coordinator.

1.1 Problem definition

In realistic autonomous construction settings, robots must be deployed on the environment where the structure will be assembled. Specific conditions or apparatus may be required to physically insert robots into the environment which can cause robots to be deployed in a different location from where a desired structure will be built with robots moving between the two locations.While moving, objects and structures that are part of the environment must be taken into account as these provide obstacles the robot must avoid. Additionally, if multiple robots are deployed at the same time, moving to reach the assembly location becomes a more complicated task as collisions become harder to anticipate and avoid. Thus, robots need to coordinate their movement as a group while planning for or reacting to conditions of the environment.

A group of robots may be responsible for assembling one or more structures at the same time. Also, certain complex structures can be split into multiple sub-structures, each being constructed interdependently. Robots can compute an assembly plan to solve the problem where multiple robots must assemble multiple structures. An assembly plan could include, for example, which robot will be assigned to which structure (or sub-structure), the path to be taken, and/or when a robot should perform the assembly. The proper design of the elements in the assembly plan can lead to fast and resource-saving construction strategies.

Finally, to assemble three-dimensional structures, parts must be sequentially placed at specific positions in space. Robots must be able to identify empty positions to place themselves such that the structural integrity is maintained throughout assembly. The challenge is determining how to design robot-to-robot interactions such that each robot is capable of finding an empty position without access to global knowledge, i.e., the position of every other robot in the structure.

1.2 Motivation

Autonomous construction of structures is a diverse area of research. The complex nature of a construction process has inspired the study of different robotic platforms, assembly plans, structure shapes and sizes, and also the quantity of intelligent agents performing the task [20]. Consequently, the assembly process poses as a compelling framework for investigation. In it, the self-assembly of structures using self-actuated intelligent parts brings many challenges in robotics.

We were driven by the possibility of increasing common knowledge and developing new methods and algorithms that could be applied to (but not exclusively) autonomous construction. As a result, different research areas that were studied in this thesis are: multi-robot motion, decentralized assembly and control methods, and three-dimensional structure modelling.

1.3 Objectives

Considering the problem and motivation described above, our objective is to investigate the application of *self-assembly* techniques for the construction of three-dimensional structures. The construction process is separated in stages and specifically designed solutions are proposed on each stage.

The objective of this research is met through the investigation of the following research goals:

(i) Propose a system description for the self-assembly problem. Specifically, we characterize the system elements and flow of information. Moreover, the types of robots used and structure shapes that can be assembled are also addressed.

(ii) Develop the necessary control laws to guide the movement of robots in space. We propose the use of network control methods to coordinately move the robotic parts to the assembly location.

(iii) Design an assembly method that is capable of assembling multiple structures simultaneously. Specifically, we propose the use of graph methods and auction algorithms to create a sequence of which robots reach the structures being assembled.

(iv) Design the robotic behaviours that guide the robots to empty positions in the structure without the necessity of a central coordinator, large communication bandwidth, or global knowledge.

1.4 Contributions

Our contributions are mainly distributed along three areas related to the self-assembly process:

- The first is the movement of robots in space, from a deployment position to the assembly area by applying network control methods. To move to the assembly location a control strategy that deals with robot-to-robot interaction, obstacle encounter, and group separation (when robots are too far from each other) is proposed. While moving, a swarm is controlled by using pin robots. Pins are the only robots in the swarm that have access to the position of the assembly location. Moreover, a Gramian-based approach for selecting pin robots in the swarm was studied. This contribution is linked to objective *(ii)* and is discussed in Chapter 4.
- The second area of study is the modelling of the robot ordering as a task assignment problem. This modelling enables the task assignment to be executed by the robots based on the actual disposition of the robots and structures in space. The assignment uses the robots' battery level and position in space to compute the best robot to move to a structure at any given moment. This contribution is linked to objective *(iii)* and is discussed in Chapter 5.
- The third contribution is the assembly process of a three-dimensional structure. Algorithms for a blueprint based assembly process where a robot relies on local information to perform the assembly are proposed. We show the properties the structure and the robots must have for the

assembly to finish properly. We propose the algorithms to be implemented on the robots and the structure blueprint. We also propose a graph-based analysis methodology that can be used to determine the seeds of each layer, in order to speed up the assembly process. This contribution is linked to objective *(iv)* and is discussed in Chapter 6.

1.5 Publications from this thesis

Some publications derived from this work are

- "**Design of a Self-Assembly System of Three-dimensional Structures using Autonomous Construction Blocks**" [1] in which the control architecture that can be used to the autonomous construction problem is discussed. This publication also presents the elements of the system, the flow of information, and the roles played by each element in an autonomous construction process.
- "**Autonomous assembly of structures using pinning control and formation algorithms**" [21] shows how swarm algorithms can be used to compute the assembly plan and guide robot motion during autonomous construction. Using simulation, we show that by applying swarm control methods only, it is possible to command multiple robots to simultaneously move to the structure location in space, performing assembly.
- "**Obstacle Avoidance of Swarms Using Pinning Control**" [22] shows how the swarm of robots can move in space to reach an assembly location while performing obstacle avoidance. The swarm is controlled using pin robots, which are the only robots in a group that are aware of the path to be followed. Also, a selection method of the pin robot based on the network controllability is proposed.
- "**Design of a decentralized strategy for layered self-assembly of 3D structures using robotic blocks**" [23] introduces an approach for self-assembly where robots rely only on local information to assemble a three-dimensional structure. The assembly is executed layer-by-layer, from bottom to top. This paper is currently under review.

1.6 Outline

The remaining of this thesis is divided as follows. Chapter 2 presents previous works related to robotic construction and correlated areas. Chapter 3 describes the autonomous construction problem and the definitions that are used throughout the thesis. Chapter 4 shows the movement of parts in a space with obstacles, from the start position to the assembly location. Chapter 5 describes the decision process represented as a task assignment problem for the MRS to choose the order of robots to move to each structure. Chapter 6 shows the algorithms that are used to construct a three-dimensional structure based on local information. Finally, Chapter 7 finishes with our conclusions.

1.7 Notation

The notation used in this document is:

- a - italic letters describe scalar variables;
- $\boldsymbol{a}$ - bold letters describe vectors;
- A - capital and italic letters describe sets and scalar constants;
- $\mathbf{A}$ - capital and bold letters describe matrices;
- $\mathbf{A}^T$ - is the transpose of the matrix $\mathbf{A}$;
- m - roman style describe the messages among robots.

2 Literature background

In this chapter, the relevant literature on autonomous construction and research areas therein is presented. We discuss the techniques, methods, and robotic platforms applied to the construction problem. First, we explain the different applications of robots in construction processes. Then, we focus on strategies that use robots as structure parts and discuss how the assembly process can be executed based on agent-level rules. Next, we present strategies to represent the desired structure and analyze how the robots use that information. Given the multi-agent characteristic of systems with robotic parts, we proceed to describe strategies used to coordinate the movement of multiple robots in space. Finally, we discuss the order of placement of parts in the assembly and the possible approaches to compute a sequence among robots in self-assembly problems.

2.1 Autonomous construction strategies

In autonomous construction, robots are used to carry parts to assemble a desired structure. Either a single vehicle [24] or multiple platforms [25] can be used for the assembly. Also, different strategies can be used when placing the parts, e.g., vehicles can build structures layer-by-layer [6], where a layer needs to be finalized before moving to the next, or the robots can consider all layers at once [24, 21] and take advantage of the fact that parts of the structure are independent of each other.

Mobile ground robots with manipulators attached can be used to handle the parts of the structure [4, 26, 27]. In general, ground vehicles are very stable when handling structure parts. However, physical devices are prone to imprecise and incorrect movements, and issues may arise when performing picking, transporting, and placing operations. To achieve high precision in the manipulation of parts, redundancy can be added by applying cooperative manipulation of parts using multiple robots [28, 29].

To assemble three-dimensional structures, aerial vehicles can be used. Due to their capability of carrying weight, vertical take-off and landing, and actuating on different heights, quadrotors are applied to transport structure parts [30, 6, 24]. Such vehicles are very versatile regarding the type of structure that can be built, the parts they can carry, etc. The correct control of the vehicle during load transportation and manipulation is crucial to guarantee flight stability. Among others, linear and nonlinear techniques are used to stabilize aerial robots [31, 32].

Not only are there control challenges related to the transportation of parts but also planning challenges to transport and place the pieces. Commonly, the assembly process and the robotic movement are guided by an *assembly plan*. Assembly plans can contain: the location and sequence where parts should be placed [33, 8], the trajectory of the robot [27], and which robot in a group is responsible to transport each part [8, 34]. In that context, *planning strategies* refer to the set of processes, methods, and algorithms used to create the assembly plan. Planning strategies can apply learning and heuristic search [33], be based on local rules (on each robot) [35], or even be executed online in stochastic settings [36].

How a robot reacts to the environment can be written as a set of *assembly rules* [35, 34]. Before placing a part, robots may need to identify allowed positions in the structure. Allowed positions are those that contribute to the assembly avoiding "dead-ends" on the construction (e.g. avoid unfillable gaps). The robot's commands (actions to be executed) are generated based on the set of assembly rules implemented on the agent's algorithms. On rule-based assembly, the *planning strategy* represents the design of the set of assembly rules. Also, when the assembly process is based on rules of interaction, there is not a explicitly defined assembly plan, instead, interactions between robots and environment guide the agents.

Most of the works discussed here apply robots to the manipulation of parts. However, robotic assembly strategies include not only *autonomous construction* by having robots carrying parts, but also *self-assembly*, wherein the parts themselves are mobile and contribute to the building of the structure. A key difference between autonomous construction and self-assembly processes is that, in the former, the number of agents is smaller and vehicles need to go back and forth while assembling the structures. Therefore, the agents may need to be more sophisticated, for example, by adding weight carrying capabilities [30, 7] and also needing to plan the assembly and moving sequence [37, 25] to optimize assembly and avoid collisions. On the other hand, simpler agents are usually used for self-assembly processes, and they must move and place themselves coordinately and cooperatively into the structure being assembled.

2.2 Robots as structure parts

Self-assembly is the process wherein simple autonomous building blocks spontaneously assemble into more complex structures, resulting in large-scale shapes entirely through local interactions and sensing [11, 10]. The main research question of self-assembly methods is how to obtain the desired global outcome (desired shape) by acting on local rules of interaction only [13, 10, 38]. Also, one may want to ensure that the desired structure is the only possible outcome of the group interaction. In some cases, such as in stochastic environments, if not properly defined, interactions may lead to different structures or the necessary interactions to assemble a desired structure may not be guaranteed [13, 10, 36].

In general, self-assembly methods can be additive or subtractive. In additive approaches, the material is added to form the structure [11, 10], whereas in subtractive techniques material is removed from an initial volume until the desired shape is reached [39]. Commonly, self-assembly additive processes start with a seed agent. The remaining agents are added, starting from the seed position, to form a user-specified shape. In the literature, this approach is also called *directed growth* [40].

In self-assembly strategies, inter-agent relationships can be designed such that robots have to *actively* process (e.g, accept, reject) interactions with each other [10, 8]. On the other hand, agents can also *passively* process interactions, for example by connecting only to specific DNA strands attached to its surface [13]. Henceforth, we refer to the elements of the structure as *robots* for active approaches and as *parts* to cover both active and passive approaches, when needed. Also, self-assembly strategies can be based on a central controller architecture [12] or decentralized control methods [11, 13, 10]. Note that, the communication load tends to be higher on centralized approaches, given that more information from individual agents must be sent to the central controller to compute the robot's actions.

Different robotic platforms have been used to study self-assembly problems. For example, autonomous boats are used to construct planar structures on the surface of the water [41, 8, 9, 19] in which robots dock to each other using a male-to-female connection mechanism and construct structures, such as a landing platform or a bridge. Similarly, quadrotors mounted within a cubic frame are used to construct structures in midair [42, 43]. Different physical challenges are present on both platforms. Therefore, control laws and assembly plans have to be specifically tailored for each application.

In self-assembly, agents can be defined as *static*, i.e., they have labels that allow them to connect only with robots of certain labels. The agents' labels can be defined by distinct physical characteristics of each agent, such as having DNA strands attached to the particle's surface [38, 13]. The labels and the set of connections of each label are designed to form a specific shape. The possible bonds between the strands will direct which particles will attach to each other.

In contrast with static types, dynamic labels can be used to specify the type of each robot. For example, a label can identify the number of other agents a robot is attached to [10]. When robots physically attach and connections are established, the agents' labels are updated to represent their new "connectivity status". New connections are created depending on the actual label of a robot (some labels allow new connections with adjacent robots, others do not). The assembly stops when a given graph topology (and therefore a shape) is reached. Note that decentralized methods for actuating on the graph topology permeate other areas besides assembly tasks and can add, for example, robustness to noise to multi-agent networks [44].

Instead of using labels, robots' behaviour can be based on the local state of the environment (the neighbourhood of the robot). Robots act (move, rearrange in space, or change connections with others) until they reach a stable and desired. The motion of the robots can be guided by a gradient descent approach, with agents moving to be as close as possible to a reference point [11] or use optimization methods to find stable configurations on the ensemble [43].

The self-assembly is in many ways, similar to the control-related formation problem. Local and imprecise measurements may be taken by individual robots, and, based on that, robots must find a way to cooperate and rearrange into specified shapes [45, 46]. In [46], the authors also use a specific set of rules for each robot. However, instead of rearranging into a structure, robots have to reorganize themselves into a desired swarm formation. The robots do not know their positions in the ensemble nor the global goal. Instead, each robot stores a set of stable local configurations (relative position to neighbours), and react whenever they are on non-stable configurations. Robots reorganize themselves until all of them reach a stable configuration, which only happens when the desired formation of the ensemble is achieved.

Some of the approaches discussed above consider that the assembly takes place in a weightless environment or a two-dimensional plane. These assumptions simplify the assembly process by removing physical constraints intrinsic to construction, such as order/sequence of placement (lower height comes first) and structural integrity during the process. Other assumptions and properties can be derived directly from how the structure being assembled is modelled, for example, the possible formats of structure parts and the possible shapes of the final structure.

2.3 Structure modelling

Self-assembly generally must be accomplished without global knowledge, with multiple agents coordinating their activities locally. Algorithmically, designing a structure that is easy to assemble by a group of robots with local information is a challenging problem [37]. Structure modelling is tightly coupled with the algorithms used to execute the assembly, influencing algorithm performance or even simplifying assembly strategies. The focus of this section lies on the structure model and its impact on the planning and execution of assembly. We refrain from categorizing the strategies on autonomous construction, self-assembly, active, or passive where it is not relevant to the discussion.

The simplest model of structures uses lists with the position and orientation of each part in a blueprint [1, 33, 28]. List formats tend to be easily computed, stored on the robot's memory, or transmitted over communication channels. In addition, the assembly plan describes how the positions of the list must be filled, i.e., how to transport a part (autonomous construction) or how to move the robotic part to a position in the list (self-assembly).

Shape maps can also be used to model the desired structure, either by specifying the desired overall shape of the group [11] or by representing relative positions of individual parts [36]. Structure shape could be embedded into other data formats, such as Octrees [47] and polygon-based descriptions [39]. Commonly, shape-based models do not use globally defined positions and the planning strategies become more elaborated. In contrast with lists, planning strategies using shape modelling may require different capabilities from the assembly agents, such as awareness of surroundings and capability to identify if the part is within the shape in space.

The models mentioned above (e.g. lists) can be used in rule-based assembly strategies [35]. The assembly process is generic and adaptable to more than one structure shape. In other cases, the structure model merges with the concept of assembly rules [38, 13]. The proper design of the rules guarantees that the structure is the only possible outcome of a group interaction. In this case, there is not an explicitly defined assembly plan on how the process should be performed. The passive self-assembly process is performed in stochastic environments where interactions between parts happen following a probability distribution.

In addition to these methods, graphs are another common approach to model structures and can embed the desired position of the parts directly [8] or indirectly (via graph topology) [10]. An advantage of graph-based modelling is that, commonly, graph methods and properties are used to represent complex algorithmic logic and simplify planning. Therefore, assembly plans can be derived directly from graph analysis [8]. Graphs (or trees) can also be used to describe the assembly plan itself, containing the sequence of connections that parts must make [48].

In this thesis, we combine shape modelling with graph methods to represent the structure (Chapter 3). Also, the assembly is based on rules of interaction between parts (Chapter 6). Assembly rules are used to guide the robots considering their surroundings and the relative positioning in a shape is used to create different types of structures. Graph methods are used to analyze the structures and the behaviour of the robots.

The model of the structure provides some insight into the assembly process, e.g., the type of structures that can be assembled. However, models do not define how robots interact with the environment when assembling the structure. Specifically, it is not explicit how robots should move to reach a position inside the desired shape. The next section will discuss techniques that can be applied by the robots to move in the environment.

2.4 Control of robots in motion

The location in the space where robots start the construction process may not be the same as the assembly location [9]. Therefore, the task of finding paths or designing local assembly rules must be coordinated with no collision with other robots or obstacles (static/moving objects, terrain, etc.) in the environment. Planning strategies may compute the trajectories that each robot should follow to avoid possible collisions [12, 34, 27, 37, 9].

For robots acting as structure parts, one can take the advantage that robotic parts form an MRS and obstacle avoidance on swarms is a recurrent topic of study [49, 50]. Networks and graph representations are commonly used to model a group of robots moving in space [17]. A common practice is to model the network edges as the sensor/communication range of agents. In addition, when the nodes in the network are controlled or regulated, classic control concepts can be used to analyze the behaviour of the whole system [51, 52]. This is usually called network control theory.

Network control concepts are used to synchronize the states of nodes in chaotic systems [14], synchronize delayed networks [53], synchronize networks with changing topology [54], and to achieve consensus (agreement over certain quantity) [16, 55, 56, 57]. In MRSs, network control is applied to formation [17, 18, 16], area coverage [18, 58], flocking [59, 16, 49], and rendezvous [15, 17].

To the self-assembly problem, flocking [49] strategies are particularly interesting, given that similar design characteristics are implemented. Markedly, rules based on local information (sensing) are implemented on each robot and enable movement in a crowded space. Also, the mathematical description of swarms uses graphs allowing the application of network control tools. Specifically, pinning control [54, 14] can also be applied to MRSs. On *pinning control*, local feedback controllers are placed on a small fraction of the nodes in the network. Such nodes are called pins or pinned nodes. The agents that are not directly actuated, the pins, will be influenced only through their connections in the network topology [60, 61, 62].

Network-based control methods are an alternative to the computation of trajectories and enable coordinated and cooperative motion in space without a heavy planning stage. One advantage of network control approaches is that they can reduce problem dimension. Instead of computing trajectories for every single agent of the MRS, a trajectory for the whole group can be designed, and the group moves as a single entity [62].

Modelling and controlling swarms with network methods allows analysis of the group behaviour with standard control techniques [51, 62, 52, 63]. That increases the predictability of the MRS dynamics and behaviour. In this thesis, such network control methods are applied to enable the navigation of a swarm of robotic structure parts in an environment with obstacles (Chapter 4). Commonly, by applying such methods, agents can operate in more dynamical environments given that environmental aspects are processed on the control level (not on the planning stage). On the other hand, it is not common for control approaches to take high-level decisions needed during assemble, for example, establishing an order on the assembly and deciding where each robot should move to. Strategies for computing the order of placement of parts are discussed in the next section.

2.5 Assembly order and task assignment

To execute the construction, it may be necessary to compute the order of which robots move to a position in the structure, given that some positions must be filled before others and some robots may have priority. Recall that on Section 2.1 the sequence of which parts are placed in the structure is a component of the assembly plan. We denote this sequence as the *assembly order*. This section discusses the computation of the assembly order focusing on active self-assembly problems (robots as parts).

For ease of understanding, let us categorize the assembly order on self-assembly problems into two areas: *sequencing* and *ordering*. Sequencing denotes the sequence of which positions in the structure will be occupied by a robot and ordering refers to the process of selecting a robot to occupy an empty position given that there are multiple robots equally capable of being selected.

Sequencing is related to the current status of the assembly and can be solved by selecting positions that are close to a starting point (seed locations) [11], through direct interpretation of the structure model using graph methods [8], or using assembly rules that dictate available positions on the structure [35]. In this thesis, we opt for a rule-based solution given its capability of assembling the structure based on local information on each robot and the possibility of online computation.

While sequencing approaches involve the analysis of the environment and the structure being assembled, ordering methods contemplate intrinsic robot parameters and their relationship with the structure. In the literature, the ordering problem is solved in different ways, for example, using optimization of robots' trajectories (robot with best trajectory is selected) [9], selecting the closest part to the structure based on an attraction force [36], or using rules based on robots' surroundings, where a robot is allowed to move to the structure if it is not physically blocked by others [11]. Also, ordering is indirectly achieved on stochastic environments [10, 38], on them, the order of "selected" parts depend random interactions with the structure.

The attribution of a robot to a position in the structure being assembled can be seen as a task to be performed by the robot. Thus, task assignment methods can be considered to perform ordering. Two relevant aspects of the task assignment are: where the relevant information for the assignment problem is processed and what is the assignment strategy.

Algorithms for task assignment can be centralized, decentralized, or hierarchical. On centralized methods, the information from all agents are sent to a single decision agent [64]. On decentralized methods, the information reaches multiple decision agents distributed on the group [65, 66]. On hierarchical approaches, a larger group of agents is split into multiple subgroups where information is shared locally and tasks are assigned within the subgroups [67]. Centralized algorithms tend to find globally optimal solutions easier, given the amount of information available. On the other hand, decentralized approaches generally make use of consensus algorithms to achieve the same information on different robots and are more scalable and robust to failures [68].

Auction processes [69, 70, 66] are well known strategies for assigning tasks, and are applicable for centralized and decentralized methods. In auction-based approaches, robots bid for a task based on the internal computation of a cost function. The auctioneer selects the highest (or lowest) bid, yielding the task to the winner. In general, auction algorithms are computationally efficient but not robust to communication losses [70].

In addition to auction strategies, the use of Markov Decision Processes (MDP) is another approach for modelling decisions with multiple agents. MDP modelling is particularly useful since optimal (or near-optimal) decision-making is achieved in stochastic environments [71, 72, 73]. In a similar fashion, MDP based modellings can use a central computing agent [74] or be inherently decentralized [72, 75].

Two favorable aspects of MDP modelling approaches strengthen its use on self-assembly problems. First, the capability of applying learning techniques to compute the decision policy based on specific characteristics of the problem. Second, decentralized strategies with multiple decision agents can be used increasing robustness and scalability. On the other hand, when compared to other methods, a disadvantage of MDP modelling is the higher complexity to model the decision problem in a self-assembly process. In addition, learning the decision policy is an additional, time consuming, step when designing the assignment solution.

In this thesis, we propose an explicit computation of the ordering by modelling it as a task assignment problem (Chapter 5). In the self-assembly context, the objective is to distribute tasks (which are positions to be occupied) to a group of robotic structure parts. We opt for a centralized auction-based given modelling simplicity with satisfactory results. However, the communication network and the processing power of robots in self-assembly problems enable, if needed, the implementation of decentralized or hierarchical strategies. Also, assignment policies based on MDP modelling can be learned offline and previously programmed to the robots performing assembly, similar to the learn-based assembly plan in [33].

3 Self-assembly fundamentals and formulation

In this chapter, we introduce the autonomous construction problem, focusing on the self-assembly of three-dimensional structures using autonomous cubic parts. We describe the fundamentals of the self-assembly, as well as the premises used to develop our solution. Self-assembly strategies commonly rely on the proper design of agent-to-agent interactions. How to obtain a desired structure by designing local rules of interaction is a major research question on self-assembly systems [13, 10, 11]. Also, one may want to guarantee that the desired structure is the only possible outcome of the local-level interactions [10].

The assembly can be done using different strategies. For this work, structures are built layer-by-layer, from the bottom to the top. The self-assembly problem, as proposed in this thesis is divided into three stages:

1. Moving - refers to the stage of the assembly when robots have to move, from an initial deployment location towards the location where structures should be built.
2. Ordering - refers to the stage of the assembly when robots have to collectively decide the order in which they will move to a structure being assembled. This stage is relevant for two reasons, first, there may exist more then one structure being assembled at the same time. Second, the formulation of the placing stage requires that robots only move one at a time to a structure.
3. Placing - this is the final stage of the assembly when robots will actively search for an empty position in the three-dimensional structure.

Fig. 3.1 provides a visual representation to the self-assembly stages. In the figure, grey blocks represent the robotic parts. Also, green squares represent structure positions that must be occupied by a robot. Finally, the arrows provide a generic notion of robotic motion in space, from the deployment area

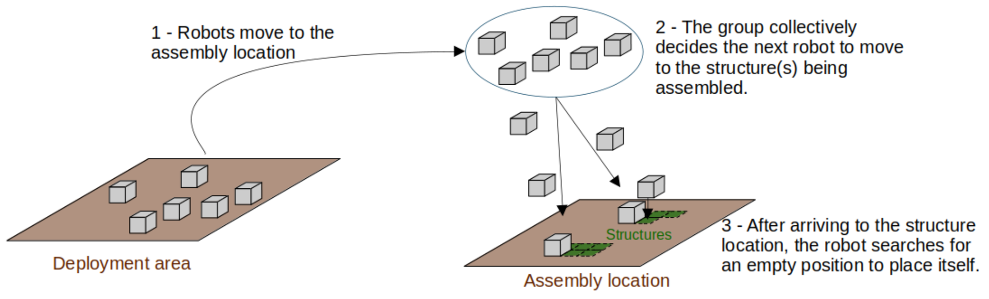

1 - Robots move to the assembly location
2 - The group collectively decides the next robot to move to the structure(s) being assembled.
Structures
3 - After arriving to the structure location, the robot searches for an empty position to place itself.
Deployment area
Assembly location

Figure 3.1: Description of the stages of the layered self-assembly process discussed in this thesis.

to structure position and through the stages. Some assumptions and premises are considered at each stage, depending on the problem being solved. The following sections discuss common (e.g., system architecture) and specific (e.g., planning algorithms) items. Finally, the robot's dynamic model, the structure blueprint, and the properties of the self-assembly problem are presented.

3.1 Self-assembly system architecture

Let us propose a high-level system architecture organization containing the entities and information involved in the construction problem [1]. The architecture shown in Fig. 3.2 has two distinct components that can be categorized as: a) *System Modules* represented as blue boxes; and b) *System Information* represented as black arrows. The System Modules and their roles are:

- Planning: this module coordinates the desired tasks for each robot to perform the construction of the structure.
- Control: this module is responsible for regulating the operation of the robot to follow the desired references. For example, in a wheeled land robot the controllers could be used (but not only) to regulate the speed of the wheels, and for a quadrotor, the controllers can regulate the attitude angles, velocity, and position.
- Robot: this represents the autonomous vehicle used for the construction task. In this thesis, the robot is also the autonomous construction block, which will form the final structure when grouped with other robots.
- Sensing: this module describes all sensors in the robot and, consequently, the hardware and software needed to process their information. Physical devices, signal processing algorithms, filters, state estimators and observers are all contained in this module.

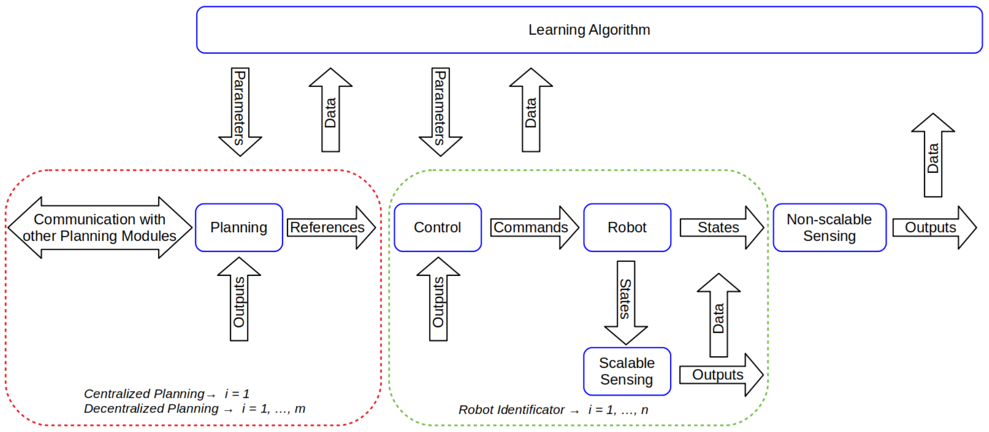


Figure 3.2: High-level description of an autonomous construction system. Source [1].

- Learning Algorithm (LA): this module represents any approach to optimize the planning task or the control operation. The machine learning algorithms, heuristic search, or optimization techniques to improve the performance of the whole system are expressed through this module.

Likewise, the System Information is:

- References: the information that the Planning module outputs are the references that the robot must follow. For example, desired position, velocities, trajectories, or maneuvers.
- Commands: the output of the controllers are commands given to the robot, to execute a desired action. For example, voltage input of electric motors.
- States: these are the physical states of the robot at a given moment, that will be measured by the sensors. For example, the actual position in space.
- Outputs: the outputs of the sensors are the actual measured or estimated states of the vehicle. This information can be useful for the Control, Planning or LA modules.
- Parameters: the results of an optimization process are the parameters to be modified inside the other modules of the system, to achieve better performance. For example, values for the gains of the control laws.
- Data: all information that can be extracted from the system, and is useful to the LA, is considered as *Data*.

Commonly, each robot has its own dedicated set of controllers. However, for other System Modules, the number of elements may not increase as the number of robots increases. Thus, let us consider a few approaches:

- Centralized planning: there is only one planning algorithm responsible for sending the references for all robots in the system.
- Decentralized planning: there are multiple planning algorithms. Each planning effort can operate with one robot or with a group of robots. It may be desirable that planning algorithms can communicate to coordinate the tasks of the robots, hence this stage can be referred to as *distributed planning.*
- Non-scalable sensing: this represents a set of sensors that will not be replicated or expanded by increasing the number of robots. For example, an infrared camera system that calculates the position of the robots in space, or the number of GPS satellites in the constellation.
- Scalable sensing: by increasing the number of robots operating it is also necessary to increase the number of these sensors. For example: embedded sensors such as LIDAR, accelerometers, or gyroscopes.

The following sections describe the problem to be solved on the *System Modules*, under the context of the three-dimensional self-assembly. Moreover, the particularities of each assembly stage are highlighted as needed.

3.2 Planning

As seen in Sections 2.1 and 2.2, different approaches can be taken for the placement of parts. In this thesis, it is assumed that horizontal layers are assembled sequentially, from the bottom to the top of the structure, similar to [6]. A layer-by-layer approach is particularly advantageous in real-world assembly scenarios, given that new layers are placed on top of others already constructed, providing structural integrity during the process [76]. Moreover, it is assumed that directed growth from a *seed position* is obtained at every layer. The use of directed growth is beneficial given that it is independent from the final structure and initial configuration [40, 11], being applicable to various shapes.

Planning strategies have a different meaning for each assembly stage. In the moving stage, planning refers to the design of control rules and computation of reference trajectories to guide the robots from a start position to the assembly location. While moving, it is assumed that robots can sense other robots and obstacles in the vicinity. The planning strategies for the movement of robots can be found in Chapter 4.

After arriving at the assembly area, robots are then ready to move to one of the (possibly) many structures being assembled. A key requisite of the assembly algorithm developed in this thesis is that the robots must arrive at the structure being constructed one by one. Therefore, in the ordering stage, planning algorithms must be able to define a sequence of agents to move from a hovering state to a structure. In this stage, robots are aware of the distance to the structures being assembled and their internal sensory information (e.g., battery level). Planning algorithms that arbitrate which robot should move and to which structure being built are presented in Chapter 5.

In the placing stage, planning algorithms are responsible for searching for an empty position in the structure, without access to global knowledge (the empty positions are not known a priori). Robots can locate themselves with respect to other parts already placed in the structure and carry a structure blueprint. The algorithms presented in Chapter 6 use the local information to search for empty positions on the structure. As mentioned previously, in the ordering stage, planning algorithms account for multiple structures being simultaneously assembled by the same group of robots. Thus, in the placing stage, each instance of the planning algorithms assembles a single structure.

3.3 Control

Control refers to the System Module that regulates the robots' states to achieve desired reference values. These setpoints are defined by the planning stage, based on the local task that the robots are pursuing. The set of controllers on each vehicle is dependent on the platform, i.e., controllers regulating autonomous surface boats [9] are not the same as those regulating aerial vehicles [42].

In this thesis quadrotors are used as the robotic platform in the structure parts (see Section 3.4). The control process on quadrotors involve attitude, velocity and/or position controllers, with the planning algorithms commanding vehicles through setpoints. Commonly, for quadrotors, a set of nested Proportional–Integral–Derivative (PID) controllers is used [32]. For this thesis, low-level controllers, namely attitude and angular rate are defined based on the multicopter's characteristics. Moreover, high-level controllers such as position, velocity, and acceleration, are designed depending on the task to be performed at each stage.

In the moving stage, the goal consists of designing control laws that can regulate a group of parts moving as a swarm in a crowded environment. Therefore, robots coordinate the swarm movement with the collision avoidance task.

Note that higher-level strategic planning for reference trajectories is executed by planning algorithms. Swarm controllers explained in Chapter 4 are decentralized, use local information, and help planning algorithms simplifying the trajectories that must be planned.

The process executed in the ordering stage focuses on computing the sequence of which robots will move to structures being assembled. Thus, a simple hover procedure is required from the robots. Hovering is a basic multicopter functionality, being achieved by a set of cascaded PID controllers embedded in the quadrotor. Moreover, the same low-level controller previously mentioned on the moving stage can be reused in this state.

Finally, in the placing stage, it is assumed that the robots can sense their immediate surroundings and navigate to position waypoints using such information. Therefore, angular rate, attitude, acceleration, velocity, and position controllers must be implemented in the quadrotor to allow waypoint navigation. As it is shown in Chapter 6, controllers can be fed signals only by onboard sensors, and still perform the required waypoint navigation. The planning module computes waypoints until the robot is placed in an empty position in the structure.

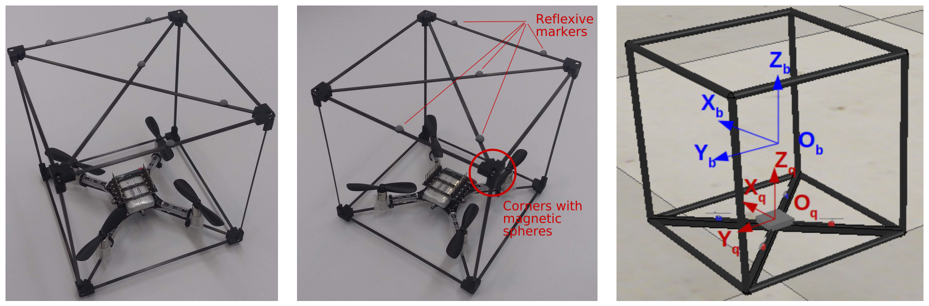
3.4 Robot - Self-actuated Construction Block

In our approach, the structure parts are cubic frames attached to a quadrotor, capable of moving autonomously. This aerial vehicle is similar to the BitDrones, described by Rubens [77]. In contrast, in our approach, the construction blocks use the Bitcraze Crazyflie 2.1 quadrotor[1] as the quadrotor platform, with 3D-printed pieces and carbon fibre structure around it. Fig. 3.3 shows the real-life autonomous part and the simulated version.

The quadrotor is embedded with sensors that allow it to estimate its attitude, linear acceleration and angular speed. The vehicles are also embedded with controllers to regulate their attitude and position in the environment. Small magnetic spheres are placed at the corners of the cubic frame to assist in attaching two adjacent blocks and reflective markers are placed on top of each block (Fig. 3.3b). Such markers are visible to a VICON infrared camera system[2], providing the position, velocity and attitude of each part in real-time. Fig. 3.4 shows the a high-level view of the loop of information when using the camera system. Camera data is processed on a dedicated server and the robots' states are transmitted to the quadcopters.

[1]www.bitcraze.io

[2]www.vicon.com



(a) Physical robot (b) Physical robot (c) Simulated version

Figure 3.3: Construction block using the Crazyflie quadrotor.

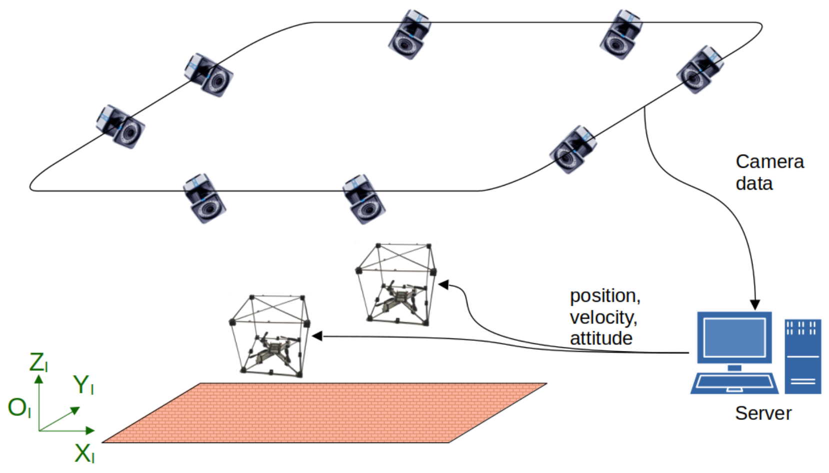


Figure 3.4: VICON system setup.

Consider an inertial frame fixed in the environment, $O_I = (X_I, Y_I, Z_I)$. The horizontal plane is described by the axes X_I and Y_I, and Z_I points upwards (see Fig. 3.4). Moreover, each block has its own coordinate frame, $O_b = (X_b, Y_b, Z_b)$, with the origin at its geometric center. Lastly, a third coordinate frame, $O_q = (X_q, Y_q, Z_q)$ is fixed at the center of gravity of the quadrotor. The axes X_q and Y_q are coplanar with the plane formed by the four propellers. X_q points toward the front of the quadrotor between the two front motors. The Y_q axis is directed 90° to the left of X_q. Z_q points upwards (towards the origin of O_b), perpendicular to the plane of the motors. In Fig. 3.3c one can see both O_b and O_q. The coordinate system O_b is translated with respect to O_q along the $Z_q = Z_b$ axis, with no rotation between them.

The relationship between the inertial frame (O_I) and the block frame (O_b) is given by the angles of pitch (rotation about Y_b), roll (rotation about X_b), and yaw (rotation about Z_b). Moreover, the angles of (roll=ϕ, pitch=θ, yaw=ψ) describe the rotation between quadrotor frame (O_q) and inertial (O_I), as well as between block frame (O_b) and inertial (O_I), given that O_b and O_q are perfectly aligned.

3.4.1 Quadrotor modelling

Denote by $\mathbf{R}_q^I$ the rotation matrix from O_q to O_I,

$$\mathbf{R}_q^I = \begin{bmatrix} c\theta c\psi & s\phi s\theta c\psi - c\phi s\psi & c\phi s\theta c\psi + s\phi s\psi \\ c\theta s\psi & s\phi s\theta s\psi + c\phi c\psi & c\phi s\theta s\psi - s\phi c\psi \\ -s\theta & s\phi c\theta & c\phi c\theta \end{bmatrix} \tag{3.1}$$

where $c(\cdot)$ and $s(\cdot)$ denote *cosine* (.) and *sine* (.), respectively.

Each motor generates a force f_m in the direction of Z_q. The total exerted force, f_{all}, is in the Z_q axis. Also, the torques τ_x, τ_y, τ_z (around axes X_q, Y_q, Z_q, respectively) can be computed as

$$\begin{aligned} f_{all} &= f_{m1} + f_{m2} + f_{m3} + f_{m4} \\ \tau_x &= \frac{\sqrt{2}}{2} l_{arm} \left(f_{m1} + f_{m4} - f_{m2} + f_{m3}\right) \\ \tau_y &= \frac{\sqrt{2}}{2} l_{arm} \left(-f_{m1} - f_{m2} + f_{m3} + f_{m4}\right) \\ \tau_z &= \frac{k_M}{k_F} \left(f_{m1} - f_{m2} + f_{m3} - f_{m4}\right) \end{aligned} \tag{3.2}$$

where k_F and k_M are the propulsion and moment coefficients produced by each rotor/propeller set. l_{arm} is the distance between the center of mass of the quadrotor and the axis of rotation of the rotors. Also, it is assumed that the rotors numbers 1, 2, 3, and 4 are on the front-left, front-right, back-right, and back-left positions respectively.

Given the linear velocities in the axes X_q, Y_q, and Z_q (u, v, and w, respectively), the linear acceleration vector is denoted by $\begin{bmatrix} \dot{u} & \dot{v} & \dot{w} \end{bmatrix}^T$ and it is measured at the center of mass of the quadrotor (origin of O_q).

The linear acceleration vector $\ddot{\boldsymbol{p}} = \begin{bmatrix}\ddot{x} & \ddot{y} & \ddot{z}\end{bmatrix}^T$ in the inertial reference system O_I as can be computed as

$$\ddot{\boldsymbol{p}} = \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} + \mathbf{R}_q^I \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} \tag{3.3}$$

where g is the gravitational constant. The linear acceleration in Z_q is computed as $\dot{w} = \frac{f_{all}}{m}$, where m is the mass of quadrotor and cubic frame combined.

The rotational dynamics of the vehicle is

$$\begin{aligned} \omega_x &= \left[\left(\iota_{yy} - \iota_{zz}\right)\omega_y\omega_z/\iota_{xx}\right] + \tau_x/\iota_{xx} \\ \omega_y &= \left[\left(\iota_{zz} - \iota_{xx}\right)\omega_x\omega_z/\iota_{yy}\right] + \tau_y/\iota_{yy} \\ \omega_z &= \left[\left(\iota_{xx} - \iota_{yy}\right)\omega_x\omega_y/\iota_{zz}\right] + \tau_z/\iota_{zz} \end{aligned} \tag{3.4}$$

where ω_x, ω_y, and ω_z are the angular velocities of roll, pitch, and yaw (around axes X_q, Y_q, and Z_q). Also, ι_{xx}, ι_{yy}, and ι_{zz} are the inertia moments of the quadrotor with respect to the axis X_q, Y_q, and Z_q respectively.

Denote by $\begin{bmatrix}\omega_x & \omega_y & \omega_z\end{bmatrix}^T$ the angular velocity vector described in O_q, and by $\begin{bmatrix}\dot{\omega}_x & \dot{\omega}_y & \dot{\omega}_z\end{bmatrix}^T$ the angular acceleration vector (also in O_q). The angular acceleration vector, $\begin{bmatrix}\ddot{\phi} & \ddot{\theta} & \ddot{\psi}\end{bmatrix}^T$, in the inertial coordinate frame (O_I) can be obtained by computing

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & s\phi t\theta & c\phi t\theta \\ 0 & c\phi & -s\theta \\ 0 & s\phi sec\theta & c\phi sec\theta \end{bmatrix} \begin{bmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} \tag{3.5}$$

where $t(\cdot)$ and $sec(\cdot)$ denote *tangent* (.) and *secant* (.), respectively.

3.4.2 Robot's simplified model

Consider a set of robots, R, participating in the assembly. Each robot ($r_i \in R$), could be described as a double integrator,

$$\begin{aligned} \dot{\boldsymbol{p}}_i &= \boldsymbol{v}_i \\ \dot{\boldsymbol{v}}_i &= \boldsymbol{a}_i \end{aligned} \tag{3.6}$$

where $\boldsymbol{p}_i$ and $\boldsymbol{v}_i$ are respectively the position and velocity of each agent. Also, a_i is the acceleration input signal of the system. Finally, $\boldsymbol{p}_i$, $\boldsymbol{v}_i$, and $\boldsymbol{a}_i \in \mathbb{R}^3$.

The double integration model is a simplification of the quadrotor dynamics shown in Section 3.4.1. In the literature, the use of double-integrator systems is shown to yield the same results of more complex models [49]. Note that, the double-integrator model encapsulates the behaviour of complex control architectures, such as those needed to control quadrotors. The simplified model is particularly useful when the focus of the analysis is not the individual robot's control, but the group behaviour, such as in Chapter 4.

Given that the autonomous structure part was properly defined on this section, we now proceed to describe the structure blueprint to be carried by the robots. The blueprint is a representation of the structure to be assembled and is used to derive some properties of the construction process.

3.5 Structure blueprint and assembly properties

Our structure parts are cube-shaped robots and the structures of interest are those composed of N blocks. Let us propose a representation of the structure composed of a) a *blueprint* B and b) a *graph* G. The blueprint carries information about the distribution of blocks in space, while the graph is an interpretation of the blueprint used to establish mathematical properties of the structure and for our proofs. Note that, given that the self-assembly problem was organized into three stages, the structure representation is particularly relevant only in the last one, placing (Chapter 6).

In the *blueprint*, space is organized as a three-dimensional grid (3D lattice). Let $P \subset \mathbb{Z}^3$ be the set of desired block positions in the blueprint referred to as *internal positions*. The set of positions that should remain empty (outside the structure) in the blueprint are called *external positions* and are denoted by $\bar{P}$. Notice that $P \cap \bar{P} = \emptyset$. Every grid position $\boldsymbol{\rho}_i$ is a possible block position and it is addressed using coordinates $\boldsymbol{\rho}_i = \begin{bmatrix} x_i & y_i & z_i \end{bmatrix}^T \in \mathbb{Z}^3$ in the structure's reference frame, $O_B = (X_B, Y_B, Z_B)$. Note that a position $\boldsymbol{\rho}_i$ can be either in P or $\bar{P}$.

Positions in the blueprint are split into 2D horizontal layers $P_l \subset P$ and $\bar{P}_l \subset \bar{P}$, $1 \leq l \leq L$. The coordinates of a part refer to the centre of the block, i.e., of the robots. It is also assumed that the robots have access to their required orientation locally, i.e., they are all oriented to an arbitrary orientation. Also, *perimeter* positions are the positions in P that are horizontally *adjacent* to positions in $\bar{P}$. In that sense, a block in a perimeter position has at least one of its sides facing the exterior of the structure. Throughout the text, the word *adjacent* is used to describe positions $\boldsymbol{\rho}_i = \begin{bmatrix} x_i & y_i & z_i \end{bmatrix}^T$ and $\boldsymbol{\rho}_j = \begin{bmatrix} x_j & y_j & z_j \end{bmatrix}^T$ in the same layer whose $\text{dist}(\boldsymbol{\rho}_i, \boldsymbol{\rho}_j) = 1$. Moreover, dist$(\boldsymbol{\rho}_i, \boldsymbol{\rho}_j)$ is defined as the Manhattan distance,

$$\text{dist}(\boldsymbol{\rho}_i, \boldsymbol{\rho}_j) = |x_i - x_j| + |y_i - y_j| + |z_i - z_j| \tag{3.7}$$

with $\boldsymbol{\rho}_i$ and $\boldsymbol{\rho}_j$ described with respect to O_B.

Note that the blueprint contains the set of positions to be occupied by a block and the set of seeds positions, $B = (P, S)$. Seed positions are preselected at each layer of the blueprint and selection methods are discussed in Chapter 6. Let the set of seed positions be $S = \{\boldsymbol{\rho}_{s_1}, \boldsymbol{\rho}_{s_2}, ..., \boldsymbol{\rho}_{s_L}\} \subset P$. The seed position at the bottom layer of the structure is $\boldsymbol{\rho}_{s_1}$ and $\boldsymbol{\rho}_{s_L}$ is the seed position at the top layer. Finally, the origin of the reference frame O_B is placed at position $\boldsymbol{\rho}_{s_1}$. Therefore, without loss of generality, for all robots assembling this structure $\boldsymbol{\rho}_{s_1}$ is the origin of the coordinate system, i.e., $\boldsymbol{\rho}_{s_1} = [0\ 0\ 0]^T$. Fig. 3.6 shows an example of a pyramidal structure and its blueprint with seed positions and reference frame. In the figure, $L = 3$.

Recall that O_b denotes the coordinate system of a block (see Fig. 3.3c). Thus, the position $\boldsymbol{\rho}$ describes the origin of O_b with respect to structure frame (origin of O_B). Also, structures are assembled at a specific position in space. Denote by $\boldsymbol{p}^{\mathsf{s}}$ as the position of a structure (its origin, O_B) with respect to the inertial frame, O_I. Fig. 3.5 helps visualize these relationships between assembly frames. Note that $\boldsymbol{\rho}$ is discrete (relative block positions inside the structure) and $\boldsymbol{p}$ is continuous (positions in the environment). Chapters 4 and 5 only consider the structure location in space and use inertial coordinates. Also, Chapter 6 describes local rules of assembly and only consider local coordinates (with respect to O_B).

For ease of explanation, an underlying graph $G = (V \cup \bar{V}, E)$ is extrapolated from the proposed blueprint B using a bijection f where $f : V \to P$ and $f : \bar{V} \to \bar{P}$. Two vertices v_1, v_2 are adjacent in G if and only if they are on the same layer and $dist(f(v_1), f(v_2)) = 1$. In the same fashion as before, vertices in V are called *internal vertices* and must be occupied by a block. Also, vertices in $\bar{V}$ are *external vertices* and are not meant to be occupied. For each layer l, denote V_l and $\bar{V}_l$ as the set of vertices that are mapped by f to P_l and $\bar{P}_l$, respectively.

Let us introduce the notion of *induced subgraph*. Given a graph $G = (V, E)$ and a subset of its vertices denoted by $\mathcal{V} \subset V$, $G[\mathcal{V}]$ is the subgraph of G induced by $\mathcal{V}$. The subgraph $G[\mathcal{V}]$ consists only of vertices in $\mathcal{V}$ and edges $\{u, v\} \in E$ connecting such vertices $(u, v \in \mathcal{V})$. Therefore, given that $V_l \subset V$ and $\bar{V}_l \subset \bar{V}$, two special subgraphs can be defined, $G_l = G[V_l]$ and $\bar{G}_l = G[\bar{V}_l]$, consisting of only *internal* and *external* vertices, respectively. Furthermore, internal vertex of G is either a *perimeter vertex* or a *non-perimeter vertex*

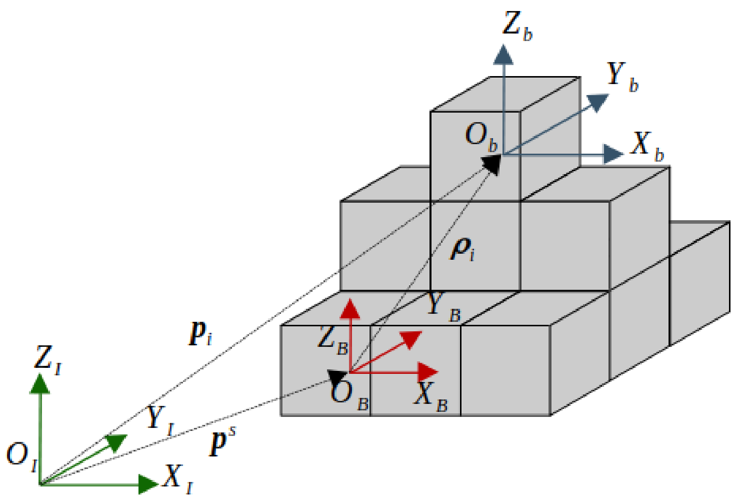


Figure 3.5: The relationship between the coordinate frames used in this work.

depending on whether it is adjacent to an external vertex or not in G. Finally, s_l denotes the seed vertex of graph G at layer l, corresponding to position $\boldsymbol{\rho}_{s_l}$ in the blueprint B such that $f(s_l) = \boldsymbol{\rho}_{s_l} \in S$.

Let us define the following properties of the graph required by the construction algorithms.

Property 1. *For each layer $1 \leq l \leq L$ of the structure, G_l is connected.*

Property 2. *For each layer $1 \leq l \leq L$ of the structure, $\bar{G}_l$ is connected.*

Property 3. *All seeds are perimeter vertices, i.e., $s_l \in G_l$ implies that there is at least one external vertex $\bar{v} \in \bar{V}_l$ such that s_l and $\bar{v}$ are adjacent in G.*

Property 4. *Every internal vertex in G_1 is mapped by f to a position on the ground. For $l > 1$, each internal vertex v_i in G_l must have a corresponding vertex v_j in G_{l-1} such that for $f(v_i) = [x_i \; y_i \; z_l]$, $f(v_j) = [x_i \; y_i \; z_{l-1}]$.*

Properties 1 and 2 ensure that there are no holes in the graph. Moreover, Property 2 is also used to ensure that any perimeter vertex can be directly reachable by a robot moving from outside the structure. Property 3 states that all seeds need to be accessible from an external vertex. Property 4 guarantees that each block in the structure is supported by blocks beneath it, thereby ensuring structural stability. Chapter 6 shows that while these properties limit the number and types of structures that can be assembled, there are several important structures that can be represented using these constraints.

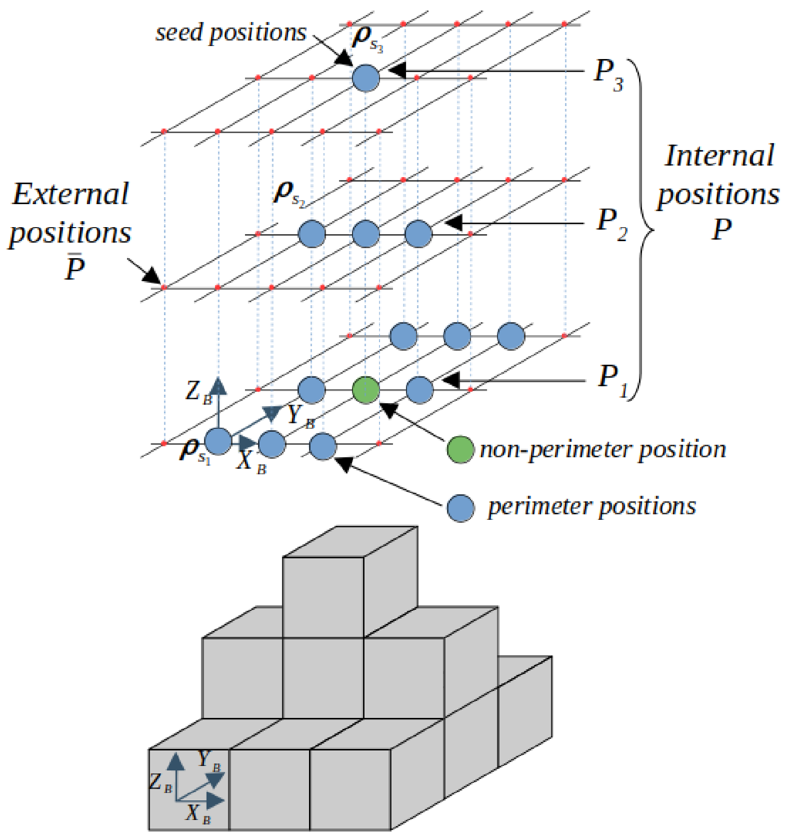


Figure 3.6: Structure and its blueprint with grid lines. The blue circles represent desired positions to place a block, P , the red dots are external positions, $\bar{P}$, and layers are indicated as P_l

The following additional properties are assumed:

Property 5. *Each robot is an entity that is capable of* ***internal computation****,* ***three-dimensional movement****,* ***communication with other robots*** *and* ***sensing its immediate surroundings*** *(detect other blocks in the vicinity).*

Property 6. *There exists a communication network that can be used by the robots. Robots may use it to broadcast control messages while the assembly process is taking place.*

It is reasonable to assume Properties 5, and 6 because for the self-assembly task, structure parts may have different levels of intelligence and capabilities.

This is done by embedding the robots with processing unit, actuators, and sensors [21, 1, 9, 41]. Also, Property 5 benefits from the use of multicopters (such as those used in this thesis), given their movement capabilities.

Property 7. *The set of structures being assembled simultaneously by the same group of robots is* $\mathbb{S} = \{\mathbf{s}_1, \ldots, \mathbf{s}_\Sigma\}$. *Thus, the total number of robots in the swarm is equal to the sum of the number of robots on each structure,* $N_R = N_1 + N_2 + ... + N_\Sigma$.

Property 8. *The set of robots in the assembly of a single structure,* $R = \{r_1, r_2, ...r_N\}$, *equals the number of internal positions in the planned structure, making* $|P| = |R| = N$.

Property 9. *Once a robot lands in a position, it shuts down. Thus, it will no longer move or communicate any information with others. There are some exceptions to this, with robots at seed positions. A robot at a seed position (also called seed robot) continues to be active, capable of communicating with others, but not moving. There is one seed position per horizontal layer of the structure.*

The rationale behind these properties follows. Properties 7 and 8 are stated for completeness, and considers a system without redundancy of blocks. We assume Property 9 for three reasons. First, shutting down mobile robots can be used as a strategy for saving battery charge. Second, keeping a single robot active can enable the system to store local information, e.g., local position and orientation reference for other robots, as in [11]. Finally, to obtain directed growth from a single point at every level of the structure, it is considered that there is one seed per horizontal layer.

3.6 Remarks

Let us state some final remarks regarding the properties and characteristics of the assembly discussed in this chapter.

Remark 1. *Real-world cubic robotic parts can be modelled and studied using their dynamic equations and the simplified model, if the focus of the analysis is the on the group dynamics, that is, the interaction between robots.*

Remark 2. *Planning algorithms can be implemented on centralized or decentralized approaches.*

Remark 3. *Robots form a communication network based on distance throughout the assembly. Robots are capable of communicating the necessary information to others without compromising the assembly or the available bandwidth.*

Remark 4. *Robots can store structure blueprints on their internal memory and are able to access its information when needed.*

Remark 5. *If multiple structures are being assembled simultaneously, the ordering stage selects which structure a robot is responsible for assembling, and therefore, which structure blueprint it should use, as will be discussed in Chapter 5.*

Each one of the parts of the architecture described in here will be discussed in the following chapters.

4 Movement of robots from start location to assembly location

In this chapter, a method for coordinating the movement of the self-assembly parts from an arbitrary start to the assembly location is presented. The swarm of robots navigate cooperatively in an environment with obstacles. Cooperative and coordinated movement refers to the process of translating all robots in space as a single entity. While moving, robots must be sufficiently close to each other and must not collide with other robots nor other objects in space. After traversing the environment, all robots should reach the assembly location and stop moving, waiting for the next stage of the self-assembly process.

Guidance and navigation of multi-robot systems is a type of application where obtaining the correct group behaviour is crucial. In such systems, robots may collide if the rules of interaction are not properly configured. Moreover, the obstacle encounters may change the spatial disposition of the swarm. As a result, groups can split and connections (communication/sensing) between robots may be broken, causing group fragmentation [49].

A swarm can be modelled as a network, represented by a graph. Network modelling is useful in this context once the relationship between agents can be formalized, and rules for the interactions between entities can be specifically designed. In this chapter, local-level control laws are developed to govern the interactions between robots, and between robots and obstacles. Moreover, it is studied here the *group separation* phenomenon (also called *group splitting* or *network rupture*) that happens at an obstacle encounter. The group separation problem is approached by breaking specific network connections, splitting a larger group of agents into smaller groups. A connectivity metric is proposed for a robot to identify if it is moving away from its neighbours and breaking its network connections. The active redesign of the network topology aims to reduce the level of swarm fragmentation.

Pinning control technique is used to control the swarm [62]. In fact, only the pin robot in the group is aware of the assembly location in space (the group objective). Other robots move given their direct or indirect connections with the pin. Pin nodes must be properly selected to control the swarm. In addition, each subgroup must have its own pin robot after group separation. To select pins, a controllability-based selection using the controllability Gramian is proposed and it is shown to perform better than random selection. The selection of a control node in a network using the controllability Gramian was presented by [52, 51].

It is assumed that every pin is capable of communicating with an agent external to the swarm, called *operator*. An operator is an abstraction of an external entity such as a person, a computer, or a more complex system responsible for controlling and monitoring the robots in the swarm. In this work, operators provide to the group the position of the assembly location, as well as the position of obstacles in the environment. Also, operators are a limited resource, i.e., there is a countable number of operators available. In addition, a pin executes the rapidly-expanding random tree (RRT) algorithm [78] that provides the collective objective in the form of a set of waypoints that a group must follow.

This chapter presents an adaptation of the work published in the IFAC2020 World Congress, under the name “Obstacle avoidance of swarms using pinning control” [22]. Thus, equations and explanations are adapted as needed to fit the context of this thesis.

4.1 Preliminaries

In this section, we present a network-based model for swarms of vehicles. First, assumptions over the network model and the premises of the control problem are established. Given that core modelling of the self-assembly problem and its structure parts were already introduced in Chapter 3, in this chapter, only complementary information about the group movement problem is introduced.

Second, the control law for a swarm of moving robots is stated based on the literature. The basic rules of interactions between agents as a dynamical network are extended/improved throughout the chapter to solve the problem of group fragmentation during movement.

4.1.1 Problem Formulation

The focus of the group behaviour of the autonomous parts, while moving from start to assembly location, is on the collective outcome given the control laws we propose. Therefore, the internal dynamics and control characteristics of individual robots may be neglected [79, 80, 62]. Thus, robots are approached using the simplified model, described in section 3.4.2.

All robots start from the start area at random locations on the ground level and with the minimum horizontal distance of $d_{\alpha\alpha}$ between each other. When deployed, robots should fly at the same height with the altitude control being approached individually. Thus, group dynamics analyzed in this work considers only the two-dimensional coordinates in the horizontal plane, being $\boldsymbol{p}_i, \boldsymbol{v}_i \in \mathbb{R}^2$. The objective of the swarm is to cooperatively move to the assembly location where the structure must be built.

Each agent has limited sensing capabilities, with a neighbouring sensing/communication region of radius r_α. Therefore, the relationship between robots can be described as a spatial graph, $G = (V, E)$, where each robot is a vertex $i \in V$. In addition, if two robots, i and j, are within a sensing/communication range of each other, then exists $e_{ij} \in E$.

Let us assume that there could be objects, structures or even other robots in the environment. These are modelled as obstacles, which are represented as circles with radius r_o^k and position $\boldsymbol{p}_o^k$, with $0 \leq k \leq N_O$. Also, obstacles could be static or move around the environment. The states of robots and obstacles (position and velocity) and the control signal may change over time. However, for simplicity, we represent them without the parameter "(t)". Finally, r_β is the interaction range (i.e. detection range) between robot and obstacle.

4.1.2 Swarm control law

An acceleration input signal, $\boldsymbol{a}_i = \boldsymbol{u}_i$, can be defined to control the position of agents in a flocking as the sum of three terms [49],

$$\boldsymbol{u}_i = \boldsymbol{u}_i^\alpha + \boldsymbol{u}_i^\beta + \boldsymbol{u}_i^\gamma \tag{4.1}$$

where $\boldsymbol{u}_i^\alpha$ is a network term that determines the interaction with other agents; $\boldsymbol{u}_i^\beta$ is an obstacle avoidance for collision term; and $\boldsymbol{u}_i^\gamma$ is a navigational term with the collective objective. Furthermore,

$$
\begin{aligned}
\boldsymbol{u}_i^\alpha &= c_1^\alpha \sum_{j\in N_i^\alpha} \phi_\alpha(||\boldsymbol{p}_j - \boldsymbol{p}_i||_\sigma)\boldsymbol{n}_{i,j} + c_2^\alpha \sum_{j\in N_i^\alpha} a_{ij}^p(\boldsymbol{v}_j - \boldsymbol{v}_i)\\
\boldsymbol{u}_i^\beta &= c_1^\beta \sum_{k\in N_i^\beta} \phi_\beta(||\hat{\boldsymbol{p}}_{i,k} - \boldsymbol{p}_i||_\sigma)\hat{\boldsymbol{n}}_{i,k} + c_2^\beta \sum_{k\in N_i^\beta} b_{i,k}^p(\hat{\boldsymbol{v}}_{i,k} - \boldsymbol{v}_i)\\
\boldsymbol{u}_i^\gamma &= c_1^\gamma \sigma_1(\boldsymbol{p}_r - \boldsymbol{p}_i) + c_2^\gamma(\boldsymbol{v}_r - \boldsymbol{v}_i)
\end{aligned}
\tag{4.2}
$$

where

- $N_i^\alpha = \{||\boldsymbol{p}_j - \boldsymbol{p}_i|| < \mathsf{r}_\alpha\}$ is the set of robots connected to node i in the spatial graph topology, G;
- $N_i^\beta = \{||\hat{\boldsymbol{p}}_{i,k} - \boldsymbol{p}_i|| < \mathsf{r}_\beta\}$ is the set of β-agents close to node i. A β-agent is a imaginary agent (with dynamics $\hat{\boldsymbol{p}}_{i,k}, \hat{\boldsymbol{v}}_{i,k}$) at the surface of the obstacle k to be avoided. The dynamics of the β-agent for an obstacle k are computed as

$$
\begin{aligned}
\nu &= \tfrac{r_o^k}{||\boldsymbol{p}_i - \boldsymbol{p}_o^k||},\\
\boldsymbol{a}_k &= \tfrac{\boldsymbol{p}_i - \boldsymbol{p}_o^k}{||\boldsymbol{p}_i - \boldsymbol{p}_o^k||},\\
\hat{\boldsymbol{p}}_{i,k} &= \nu\boldsymbol{p}_i + (1-\nu)\boldsymbol{p}_o^k,\\
\hat{\boldsymbol{v}}_{i,k} &= \nu(\mathbf{I} - \boldsymbol{a}_k\boldsymbol{a}_k^T)\boldsymbol{v}_i,
\end{aligned}
\tag{4.3}
$$

  where $\boldsymbol{a}_k$ is a unit normal vector from obstacle center to robot position. Note that the control law accounts for mobile obstacles ($\boldsymbol{p}_o^k$ changing over time) by constantly updating the states of β-agents using the equations above.
- The vectors $\boldsymbol{n}_{i,j}$ and $\hat{\boldsymbol{n}}_{i,k}$ are

$$
\begin{aligned}
\boldsymbol{n}_{i,j} &= \tfrac{\boldsymbol{p}_j - \boldsymbol{p}_i}{\sqrt{1+\epsilon||\boldsymbol{p}_j - \boldsymbol{p}_i||^2}},\\
\hat{\boldsymbol{n}}_{i,k} &= \tfrac{\hat{\boldsymbol{p}}_{i,k} - \boldsymbol{p}_i}{\sqrt{1+\epsilon||\hat{\boldsymbol{p}}_{i,k} - \boldsymbol{p}_i||^2}}.
\end{aligned}
\tag{4.4}
$$

- $c_{1,2}^{\alpha,\beta,\gamma}$ are control law gains;
- $(\boldsymbol{p}_r, \boldsymbol{v}_r)$ is the reference or γ-agent;
- $||.||_\sigma$ is the σ-norm. The σ-norm is used instead of the $||.||$ (the Euclidean norm) because it is differentiable everywhere;
- $\phi_\alpha(\cdot)$ is the *attractive/repulsive action function*, used to maintain the distance between agents approximately equal to $d_{\alpha\alpha}$;
- $\phi_\beta(\cdot)$ is the *repulsive action function*, used to avoid collision with obstacles;
- a_{ij}^p are adjacency relationships between agents and $b_{i,k}^p$ between agents and obstacles.

The terms in the control law equation (4.2) are explained below in more detail. First, the σ-norm is defined as a map $\mathbb{R}^n \to \mathbb{R}_{\geq 0}$, where $n \geq 1$

$$||\boldsymbol{k}||_\sigma = \frac{1}{\epsilon}(\sqrt{1+\epsilon||\boldsymbol{k}||^2}-1) \tag{4.5}$$

with $\epsilon \in (0,1)$.

Consider that, hereafter, the Greek letters $\boldsymbol{\zeta} \in \mathbb{R}^2$ and $\xi \in \mathbb{R}$ are used to generically represent the input of the function being described. Also, the $\boldsymbol{\zeta}$ is two-dimensional given that the control task is executed on the horizontal plane. Also, $\sigma_1(\cdot)$ is

$$\sigma_1(\boldsymbol{\zeta}) = (\boldsymbol{\zeta})/\sqrt{1+||\boldsymbol{\zeta}||^2}. \tag{4.6}$$

Furthermore, $\phi_\alpha(\cdot)$ and $\phi_\beta(\cdot)$ are

$$\begin{aligned} \phi_\alpha(\xi) &= \rho_h(\xi/r_\alpha)\phi(\xi-d_\alpha), \\ \phi(\xi) &= 0.5((a+b)\sigma_1(\xi+c)+(a-b)), \end{aligned} \tag{4.7}$$

$$\phi_\beta(\xi) = \rho_h(\xi/r_\beta)(\sigma_1(\xi-d_\beta)-1), \tag{4.8}$$

where

- $0 < a \leq b$, $c = |a-b|/\sqrt{4ab}$;
- $r_\alpha = ||\mathsf{r}_\alpha||_\sigma$ is the σ-norm of the sensing/communication range between agents;
- $d_\alpha = ||d_{\alpha\alpha}||_\sigma$, where $d_{\alpha\alpha}$ the desired distance between agents.
- $r_\beta = ||\mathsf{r}_\beta||_\sigma$ is the σ-norm of the detection range to obstacles agents;
- $d_\beta = ||d_{\alpha\beta}||_\sigma$, being $d_{\alpha\beta}$ the allowed distance to an obstacle (α-agent to β-agent).

The bump function $\rho_h(\cdot)$, used in (4.7) and (4.8), is a scalar function that smoothly varies between 0 and 1,

$$\rho_h(\xi) = \begin{cases} 1, & \xi \in [0,h) \\ 0.5(1+cos\left(\pi\frac{\xi-h}{1-h}\right)), & \xi \in [h,1] \\ 0, & otherwise \end{cases}, \tag{4.9}$$

with $h \in (0,1)$.

Lastly, the adjacency relationships can be written as

$$\begin{aligned} a_{ij}^p(\boldsymbol{p}_j,\boldsymbol{p}_i) &= \rho_h(||\boldsymbol{p}_j-\boldsymbol{p}_i||_\sigma/r_\alpha) \in [0,1], j \neq i, \\ b_{i,k}^p(\hat{\boldsymbol{p}}_{i,k},\boldsymbol{p}_i) &= \rho_h(||\hat{\boldsymbol{p}}_{i,k}-\boldsymbol{p}_i||_\sigma/d_\beta) \in [0,1]. \end{aligned} \tag{4.10}$$

Note that a_{ij}^p and $b_{i,k}^p$ depend on the distance between agents $(\boldsymbol{p}_j,\boldsymbol{p}_i)$, distance between agents and obstacles $(\hat{\boldsymbol{p}}_{i,k},\boldsymbol{p}_i)$, the sensing radius r_α, and the desired distance to obstacle d_β. In addition, one may consider that iff $a^p_{ij} \neq 0$, then $e_{ij} \in E$ exists.

For a detailed explanation on the meaning of terms described above (bump functions, σ-norm, etc), and a deep analysis of their role on the swarm control task, we kindly refer the reader to [49].

4.2 Decentralized control strategy for swarm motion

In this this section, a strategy is proposed to control robots moving in an environment with obstacles. It is the control law that regulates robots' movement, the interaction with each other and with obstacles. Moreover, given the impact that obstacles may have in the swarm, a process to split the network into smaller groups based on a connectivity metric and the selection of new pins for the subgroups is introduced.

As robots move during operation their inter-agent distance change. Such distance is mainly impacted by the presence of obstacles in the environment, which each robot will individually try to avoid (given $\boldsymbol{u}^\beta_i$). As a result, agents may break and create new connections (edges) in the network topology (note that a^p_{ij} depends on the distance between robots). The control law, equation (4.1), does not guarantee that an agent will not break all of its connections, separating itself from the group. Note that breaking network connections is not desired, once it leads to disconnected groups or robots, and the organization/cooperativeness aspect of the swarm is lost. Moreover, in some missions/modes of operation, it is actually an undesirable behaviour as it may increase the necessity of monitoring more agents independently.

Assume that the number of operators (monitoring/controlling the swarm) is smaller than the number of robots, which invalidates monitoring robots individually. Pinning control strategy can be used to actuate on the swarm while respecting such communication constraints. In pinning control the reference trajectory is only sent to the pin node (i.e, only the pin node has the information to compute $\boldsymbol{u}^\gamma_i$). An operator is therefore assigned to each pin. Other robots are indirectly controlled by their connections withing the network. Also, the number of pins must not exceed the number of available operators. Each group in the swarm contains only one pin robot and the swarm may be divided in more than one group at a given time. Each pin is connected to an operator, which provides the global objective, the location of the assembly location, to the pin.

Every time the swarm breaks its connections the group can be fragmented and create disconnected subgroups. Operators must be assigned to monitor/control each subgroup. However, since there is only a finite number of operators, only a finite number of subgroups can be formed. To restrict the fragmentation of the group, we define the concept of *minimal group*, which are groups with specially defined control laws that avoid subdivisions. Minimal groups are discussed on Section 4.2.1. Recall that N_R is the total number of agents in the swarm. Thus, we can compute N^{min} as the minimum number of nodes allowed in a subgroup of agents. $N^{min} = floor(N_R/\#operators)$ is computed using the number of available operators. The *floor* function rounds the nearest integer less than or equal to the original number.

Given that the main function of the autonomous parts is to perform the assembly task, commonly such robots are not designed to move at high speeds or execute aggressive maneuvers. Therefore, moving speeds tend to be relatively low. On the other hand, the communication between agents through the network is faster (than moving) and the time for the information to spread around the network is negligible. It is considered that information spreads amongst the robots of a given group fast enough to enable the execution of graph based algorithms (which require knowledge of the topology) or to compute the cohesion control term (responsible for avoiding subdivision of a minimal group, see Section 4.2.4).

To avoid unnecessary fragmentation of the swarm, we propose a connectivity metric to be computed locally. The connectivity metric indicates if robots are moving away from the swarm (being "less connected"), which triggers the necessity of preemptively separating the swarm into smaller groups. If there is the necessity for network redesign, that is communicated to the pin robot, which actively computes a cut to be made in the graph to separate a larger group into smaller ones.

Fig. 4.1 shows the proposed control strategy, described as a sequence of steps. Note that there is a monitoring stage and an action stage. During the monitoring, local metrics are constantly being computed to identify the cohesion (how close and capable of organized movement the swarm is). During the action stage, the network topology is redrawn, to accommodate for an increase in the connectivity metric. Also, the path planning stage computes a viable trajectory for the groups in the swarm. The controller and the connectivity metric are computed at each node, and the decision over the rupture of network connections is based on the connectivity metric. Lastly, after the division of a group, new pins are selected and new paths are computed for each pin.

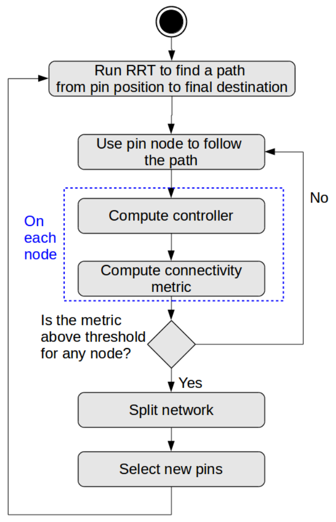

Run RRT to find a path from pin position to final destination
Use pin node to follow the path
On each node
Compute controller
Compute connectivity metric
Is the metric above threshold for any node?
No
Yes
Split network
Select new pins

Figure 4.1: Swarm control strategy.

4.2.1 Connectivity Metric and Active Group Separation

In order to measure the spatial disposition of robots in space, let us propose the *connectivity metric* m_i . This metric quantifies both the number of adjacent robots to a given agent in space ($e_{ij} \in E$) and how far the robot is from its neighbours. The connectivity metric is given by

$$m_i = \frac{max(||\boldsymbol{p}_j - \boldsymbol{p}_i||)}{\mathsf{d}_i + 1}, \; j \in N_i^\alpha, \tag{4.11}$$

with $\mathsf{d}_i = |N_i^\alpha|$ being the number of neighbours of robot i. The distance between i to the farthest robot in its sensing range is computed by $max(||\boldsymbol{p}_j - \boldsymbol{p}_i||) \; \forall j$.

Denote $V_{SG} \subseteq V$ as a group of agents in the swarm if there is a path in the network topology between any two agents in V_{SG}. The swarm may contain one or many groups at any time.

As a group moves in the environment inter-robot distances could change due to interactions with each other and with obstacles. Therefore their connections, (network edges) could also change. A group V_{SG} can potentially be split into multiple sub-groups, with no path in the graph between distinct groups.

Note that, $|V_{SG}| \geq N^{min}$ necessarily. Therefore, a *minimal group* denotes an ensemble with less than $2N^{min}$ agents. Any group with less than $2N^{min}$

agents is considered indivisible. That is, a minimal group cannot be split into two new groups without violating $|V_{SG}| \geq N^{min}$.

The connectivity metric of a robot, m_i, is used to arbitrate when the network topology must be redesigned. To do that, m_i is monitored at every robot i. If m_i reaches a predefined threshold value, then the pin of the group is advised to redesign network topology. Specific network connections are removed (removing e_{ij}), splitting a group (V_{SG}) into two new groups. The cut is made such that the group containing the node with m_i above threshold is necessarily a minimal group with N^{min} robots. Note that, groups may reconnect if their agents get closer than the connection threshold, see (4.10).

4.2.2 Selection of New Pins

After separation of a bigger group into new smaller groups of agents, pins are selected to act as group leaders and compute their desired trajectory to the goal using Algorithm 1. The selection of a new pin happens after the cut in the network, and it is executed by the pin, given that it holds the actual topology information. New nodes selected as pins are advised of their new roles. To select new pins, two approaches were used, the random selection of one of the group's agents or the decision based on the Network Controllability Gramian (NCG).

Pins control the behaviour of the whole group, directly (influencing its close neighbours) or indirectly through network interactions. Therefore, it may be advantageous to identify influential robots in the network and select those as pins. Centrality measures are a common way to identify highly influential nodes [51]. Given that in our case we have one control robot (the pin) which has to control the position (states) of other nodes to move in space, we explore the analysis of the network controllability. NCG is an energy related metric which comprises the cost of changing the states of the network towards different directions of the state space [51, 52]. Shortly, higher the control effort, higher the energy needed to actuate in a network. The NCG depends on the network topology (described as a graph) and on which nodes are selected as control inputs. In our case, the control input of the network is the pin robot. In the literature, well connected nodes in the network are known to yield higher controllability and therefore require less control energy [52, 81, 82].

To compute the NCG, first the adjacency matrix of the group V_{SG}, $\mathbf{A}_{SG} = [a_{ij}]$, is computed. For robots that are sufficiently close, $a_{ij} = a_{ji} = 1$. Vertices that are not connected yield $a_{ij} = a_{ji} = 0$.

$$\mathbf{A}_{dyn} = e^{(-\alpha \mathbf{I} + \mathbf{A}_{SG})} \tag{4.12}$$

where α is selected such that $(-\alpha\mathbf{I} + \mathbf{A}_{SG})$ is stable (negative eigenvalues) [51]. The network controllability matrix is

$$\mathbf{C}_j = \begin{bmatrix} \boldsymbol{b}_j & \mathbf{A}_{dyn}\boldsymbol{b}_j & \dots & (\mathbf{A}_{dyn})^{H-1}\boldsymbol{b}_j \end{bmatrix} \tag{4.13}$$

where the column vector $\boldsymbol{b}_j$ represents the node, j, selected as *control input* (possible pin robot). Thus, $\boldsymbol{b}_j$ has a value 1 at the $j^{th}-$row, and 0 otherwise. Also, H is the control horizon (number of time steps) to which the Gramian is computed. The NCG given the selection of vertex j as control input can be computed as

$$\mathbf{W}_j = \sum_{h=0}^{H-1} (\mathbf{A}_{dyn})^h \boldsymbol{b}_j \boldsymbol{b}_j^T ((\mathbf{A}_{dyn})^T)^h = \mathbf{C}_j \mathbf{C}_j^T \tag{4.14}$$

Finally, the trace of the NCG, $trace(\mathbf{W}_j)$, is inversely related to the average energy to control the network [81]. Thus, choosing the node j that yields the maximum controllability represents selecting the agent that will require less energy to move the group in space [52]. Thus, $H > 0$ is defined such that the values of $\mathbf{W}_j$ converged.

4.2.3 Operators and RRT algorithm

Recall that a pin is monitored by an operator and that an operator represents a remote computer, a person or a more complex system. In any case, it is assumed that the operator is able to communicate with the pin providing the position of the assembly location as the goal of the group. A pin robot computes a trajectory to reach the assembly location, while avoiding possible obstacles in the environment. Possible strategies to compute a trajectory in the environment include grid-based algorithms, such as A* [83] and D* [84], or sampling-based strategies, such as Probabilistic Road Maps [85] and the Rapidly-Expanding Random Tree (RRT) [78]. Grid-based algorithms find optimal path if it exists but are computationally heavy for bigger search spaces. On the other hand, sampling-based methods are more computationally efficient but the optimal solution may take more time to be found.

In this thesis, to compute the trajectory to be followed by the swarm, the pin implements an RRT. The use of RRT as a trajectory planner is appealing because of its simplicity, fast convergence, and for being probabilistically complete (the solution will be found if there is one). RRT has been broadly studied and its variations include operation in real time [86], adaptation to different system dynamics [87], and path optimization [88].

Algorithm 1 shows the implemented version of the RRT. In it, a tree is expanded from the current position of the pin to the goal position. Each node in the tree contains candidate waypoints in the trajectory to be followed. When the algorithm is executed, the pin robot instantiates $\boldsymbol{p}_{init}$ as the current pin position in space. Then, the algorithm proceeds by generating a random position in space ($\boldsymbol{p}_{rand}$, line 4). A search within the tree finds the closest node to $\boldsymbol{p}_{rand}$ (line 5). We call this node $\boldsymbol{p}_{near}$. The next is step is to extend the three from $\boldsymbol{p}_{near}$ towards $\boldsymbol{p}_{rand}$ by a fixed edge distance, $\Delta\boldsymbol{p}$. This process (line 6) creates a node at position $\boldsymbol{p}_{new}$ (note that $\boldsymbol{p}_{new}$ is between $\boldsymbol{p}_{near}$ and $\boldsymbol{p}_{rand}$). If this newly generated node is located inside any obstacle, it is ignored and a new $\boldsymbol{p}_{new}$ position will be generated. If $\boldsymbol{p}_{new}$ does not collide with an obstacle, then it is added to the tree. By doing this sequentially, new nodes are added and the tree expands through the search space. Moreover, if $\boldsymbol{p}_{new}$ is close to $\boldsymbol{p}_{goal}$ (distance smaller than $\Delta\boldsymbol{p}$), then $\boldsymbol{p}_{new}$ and $\boldsymbol{p}_{goal}$ are connected and added to the tree and the algorithm finishes (lines $12-13$). The algorithm returns the path (list of waypoints) from $\boldsymbol{p}_{init}$ to $\boldsymbol{p}_{goal}$.

Algorithm 1 Rapidly-Expanding Random Tree Algorithm

```
 1: function RRT($p_{init}$, $p_{goal}$, $obstacles$)
 2:     $T$.init($p_{init}$), initialize tree at position $p_{init}$
 3:     while $p_{goal}$ not reached do
 4:         $p_{rand}$ ← random(), generate a random position
 5:         $p_{near}$ ← find_near($p_{rand}$, $T$), find nearest vertex
 6:         $p_{new}$ ← new_node($p_{near}$, $p_{rand}$, $\Delta p$), create new node at distance
    $\Delta p$ of $p_{near}$
 7:         if $p_{new}$ does not collide with any obstacle then
 8:             $T$.add_vertex($p_{new}$), add node to tree
 9:             $T$.add_edge($p_{near}$, $p_{new}$), connect the new node to nearest node
10:         end if
11:     end while
12:     $T$.add_edge($p_{new}$, $p_{goal}$), connect the new node to nearest node
13:     return $T$.trace_back(), return path from initial position to goal
14: end function
```

4.2.4 Virtual Nodes and Control Law

In order to maintain cohesion in a minimal group, let us introduce the concept of virtual nodes. A virtual node is computed as the average position of all nodes in a group. Thus, the virtual node is located at the geometric center of the group. The real position and velocity of the virtual node at time t is

$$\begin{aligned} \boldsymbol{p}^r_{vn}(t) &= \tfrac{1}{N^{SG}} \sum_{j \in V_{SG}} \boldsymbol{p}_j(t), \\ \boldsymbol{v}^r_{vn}(t) &= \tfrac{1}{N^{SG}} \sum_{j \in V_{SG}} \boldsymbol{v}_j(t). \end{aligned} \tag{4.15}$$

where V_{SG} is a minimal group with N^{SG} agents. $N^{SG} < 2N^{min}$.

However, note that an agent accesses the information of another agent in the group delayed by a specific time. Therefore, each node (i) computes the virtual node position and velocity separately as

$$\begin{aligned} \boldsymbol{p}^{(i)}_{vn}(t) &= \tfrac{1}{N^{SG}} \sum_{j \in V_{SG}} \boldsymbol{p}_j(t - h_{ij}\Delta t), \\ \boldsymbol{v}^{(i)}_{vn}(t) &= \tfrac{1}{N^{SG}} \sum_{j \in V_{SG}} \boldsymbol{v}_j(t - h_{ij}\Delta t). \end{aligned} \tag{4.16}$$

where h_{ij} is the number of hops (edges) in the shortest path between two nodes. Moreover, the delay for a node i to access information of a node j equals to $h_{ij}\Delta t$, Δt being the agent-to-agent communication time. If communication happens faster than the robotic movement, the delay is negligible and the virtual node computed by any agent will be is located approximately at the real geometric center of the group.

The virtual node is used to compute the *cohesion control term*, $\boldsymbol{u}^c_i$, as

$$\boldsymbol{u}^c_i = c^{vn}_1(\boldsymbol{p}^{(i)}_{vn} - \boldsymbol{p}_i) + c^{vn}_2(\boldsymbol{v}^{(i)}_{vn} - \boldsymbol{v}_i), \tag{4.17}$$

where $c^{vn}_{1,2}$ are the control term gains.

Also, note that in a minimal group, instead of guiding the pin controller to the predefined trajectory, the navigation term is replaced to guide the geometric center of the ensemble (virtual node) to the trajectory. Thus,

$$\boldsymbol{u}^\gamma_i = \begin{cases} g_i[c^\gamma_1(\boldsymbol{p}_{goal} - \boldsymbol{p}^{(i)}_{vn}) + c^\gamma_2(\boldsymbol{v}_{goal} - \boldsymbol{v}^{(i)}_{vn})], \\ \qquad i \in V_{SG}, V_{SG} \text{ is a minimal group} \\ g_i[c^\gamma_1(\boldsymbol{p}_{goal} - \boldsymbol{p}_i) + c^\gamma_2(\boldsymbol{v}_{goal} - \boldsymbol{v}_i)], \\ \qquad i \in V_{SG}, V_{SG} \text{ is not a minimal group.} \end{cases} \tag{4.18}$$

where $g_i = 1$ if i is a pin node, and it is 0 otherwise.

The complete control law signal at each node is then given by

$$\boldsymbol{u}_i = \boldsymbol{u}_i^{\alpha} + \boldsymbol{u}_i^{\beta} + \boldsymbol{u}_i^{\gamma} + \boldsymbol{u}_i^{c} \tag{4.19}$$

Note that, the $\boldsymbol{u}_i^c$ term is only activated if the group containing i is a minimal group. Thus, maintaining group cohesion while moving through space becomes another objective (along with obstacle avoidance and navigation) of a minimal group.

4.3 Results

This section shows results obtained when simulating the proposed swarm control strategy. First, the simulation procedure and the parameters used to generate the results are presented. Second, it is shown the swarm guidance process using the proposed control strategy. Finally, the Gramian-based pin selection method is compared with a random selection in the guidance context.

As mentioned previously, the analysis on this chapter is constrained to the two-dimensional space, given that the height control task is executed individually. Therefore, $\boldsymbol{p}_i, \boldsymbol{v}_i \in \mathbb{R}^2$ with $\boldsymbol{p}_i = \begin{bmatrix} p_i^x & p_i^y \end{bmatrix}^T$ and $\boldsymbol{v}_i = \begin{bmatrix} v_i^x & v_i^y \end{bmatrix}$. Table 4.1 shows the parameters used in simulation. Also, in the RRT algorithm, a safe distance to an obstacle (point where a new waypoint is allowed in the path) is r_β.

Table 4.1: Parameters for simulation

Parameters	Value
c_1^{α}, ${}_2^{\alpha}$	2, 2.83
c_1^{β}, c_2^{β}	3, 3.46
c_1^{γ}, c_2^{γ}	1, 2
c_1^{vn}, c_2^{vn}	1, 2
ϵ, h	0.5, 0.5
$d_{\alpha\alpha}$, r_α	1, 1.3
$d_{\alpha\beta}$, r_β	0.6, 0.78
Connectivity Threshold	0.4

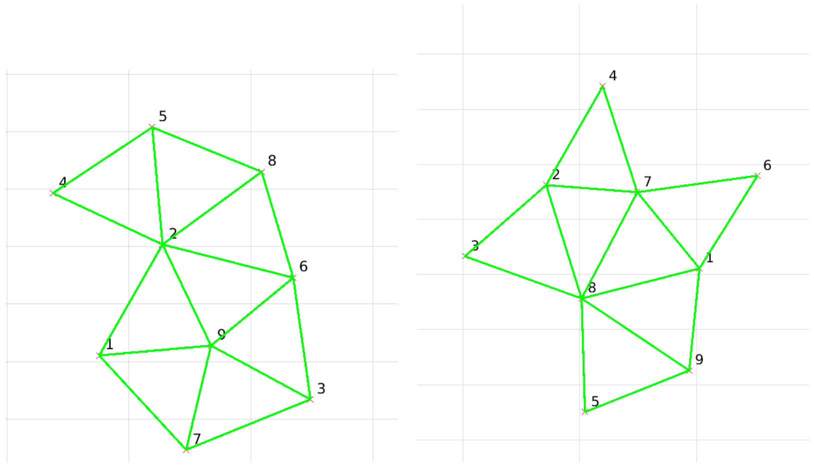
It is useful to observe the selection method for the parameters on Table 4.1. First, parameters are selected to match the same order of magnitude of the positions and velocities in the system (10^0). Guidelines, followed in here, were proposed in [49] for the definition of gains $c_{1,2}^{\alpha,\beta,\gamma}$, where $c_2 > c_1$ generated stable control operation. Also, ϵ and h were arbitrarily defined as the middle value of their respective ranges. Distances and ranges $d_{\alpha\alpha}$, r_α, $d_{\alpha\beta}$, and r_β were designed to obtain a stable swarm behaviour, i.e., robots should not move too close nor be spread in space (few connections in the network topology). Finally, the connectivity metric was defined to be below the maximum value. To compute the maximum value for m_i, we use the sensing range r_α (two robots are not topologically connected if they are above this range) and set $\mathsf{d}_i = 1$ (only connected to one other robot). In this scenario, the maximum value is $m_i = 1.3/2 = 0.65$.

Swarm guidance - In the simulation showed in Section 4.3.1, ten agents performed the navigation in an environment with obstacles. In the beginning, the agents started in random positions and network connections (e_{ij}) were established only for every pair of agents closer than r_α. After that, one agent was selected as the pin node, responsible to follow a desired trajectory and guide the group. Moreover, it was assumed that three operators are available ($\#operators = 3$), resulting in $N^{min} = 3$.

Pin selection method - A simulation of a swarm with nine agents was executed to characterize the error in position tracking using both approaches for pin selection: random agent or Gramian-based selection. To obtain the results showed in Section 4.3.2, the following procedure was applied in simulation:

1. Start robots in random positions and establish network connections for every pair of agents closer than r_α;
2. Select a pin node using the Gramian-Based method;
3. Apply the navigational term, equation (4.18), on the control law of the pin node to follow a desired trajectory. The trajectory is a sequence of waypoints, 1 meter away from each other. The waypoints define the desired position of the geometric center of the group;
4. Measure the tracking error, the pin node's control effort, and the distance travelled by each node;
5. Restart simulation using the same positions obtained in step 1;
6. Randomly select a node as the pin;
7. Apply the navigational term on the control law to follow the desired trajectory;
8. Measure the tracking error, the pin node's control effort, and the distance travelled by each node;

Fig. 4.2 shows examples of possible initial configurations. The figure shows robots identified by numbers, and connections in the graph topology are highlighted by green lines. Remember that a connection in the network topology



(a) (b)

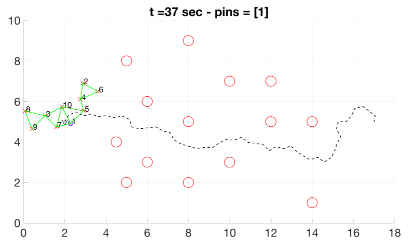
Figure 4.2: Example of possible initial configurations, pin selection method comparison.

represent that robots have access to each other states. Moreover, one of the robots in the swarm is selected as the pin, being the only robot that computes the goal term, $\boldsymbol{u}_i^{\gamma}$, in the control law proposed. Other nodes will react to the movement of the pin, being indirectly controlled.

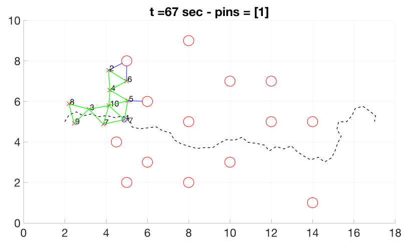
4.3.1 Swarm guidance

In this section, we show the results of the swarm guidance using our control strategy. Fig. 4.3 shows snapshots of the autonomous guidance task. It is possible to see the beginning of the task, the moment when the agents break their connections (e.g, Fig. 4.3c), and the guidance of minimal groups (e.g, Fig. 4.3h). Furthermore, the figure shows the new pins selected after separation, the creation of virtual nodes, and computation of the agents' paths to the goal as dashed lines. Note that, the trajectory computed is, in fact, a series of sequential waypoints to be reached by a pin node (or a virtual node in a minimal group).

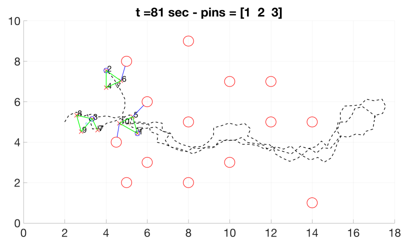
Fig. 4.4 shows the connectivity metric for each node in the group as they move through the environment with obstacles. In the figure, the moments where the network topology is actively redesigned (splitting group and selecting new pins) are highlighted. Note that, after redesign, the cohesion term



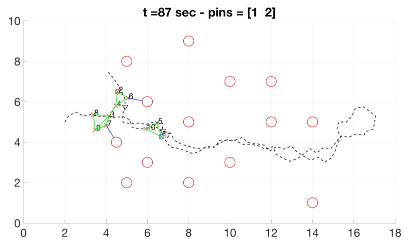
(a) Initial configuration



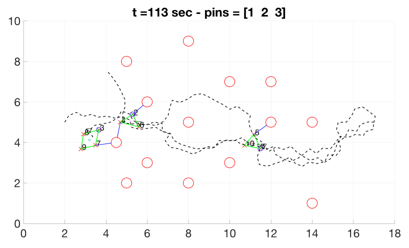
(b) Object encounter



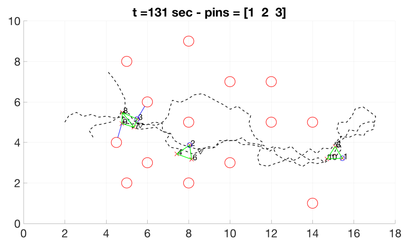
(c) Network rupture



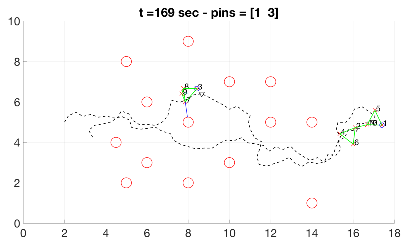
(d) Two groups reunite



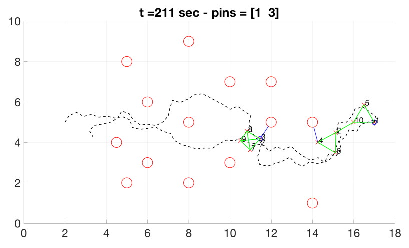
(e) Network rupture



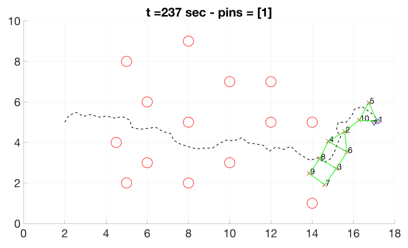
(f) Minimal groups navigating



(g) Two groups reunite



(h) Minimal groups navigating



(i) Groups reunite and end of mission

Figure 4.3: Autonomous guidance using 10 agents, $\#operators = 3$, and $N^{min} = 3$.

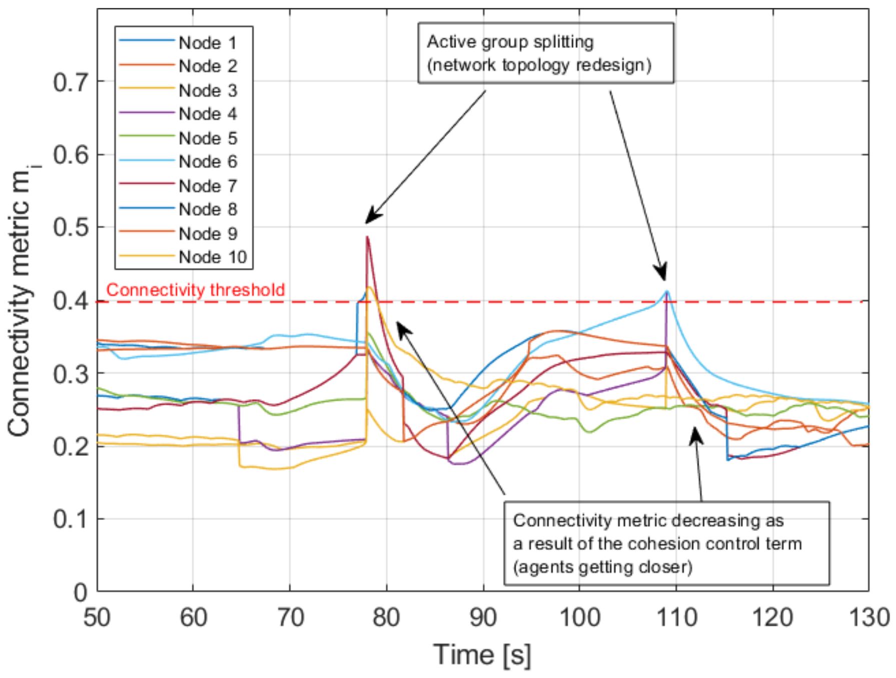


Figure 4.4: Connectivity metrics during part of the experiment.

was activated in minimal groups which causes the decrease of connectivity metric value. In Fig. 4.4 two moments where the network is redesigned and groups split are shown. These separation moments can be seen in Fig. 4.3c and Fig. 4.3e.

Fig. 4.5 shows a minimal group during navigation. The group is composed of four robots, the agent number $i = 3$ is the pin node, and $N^{min} = 3$. Since $N^{min} < |V_{SG}| < 2N^{min}$, the group is indivisible (see Section 4.2.1). Thus, instead of splitting the group, the agents rearrange their positions in order to avoid the obstacles.

4.3.2 Comparison between pin selection methods

In this section, we show the comparison between the two pin selection methods, Gramian-Based and the Random pin selection (Section 4.2.2). For each selection method, 100 iterations of 30 seconds of the procedure described in Section 4.3 were performed. In each iteration of the simulation, the same initial configuration (nodal position and network topology) was used to compare

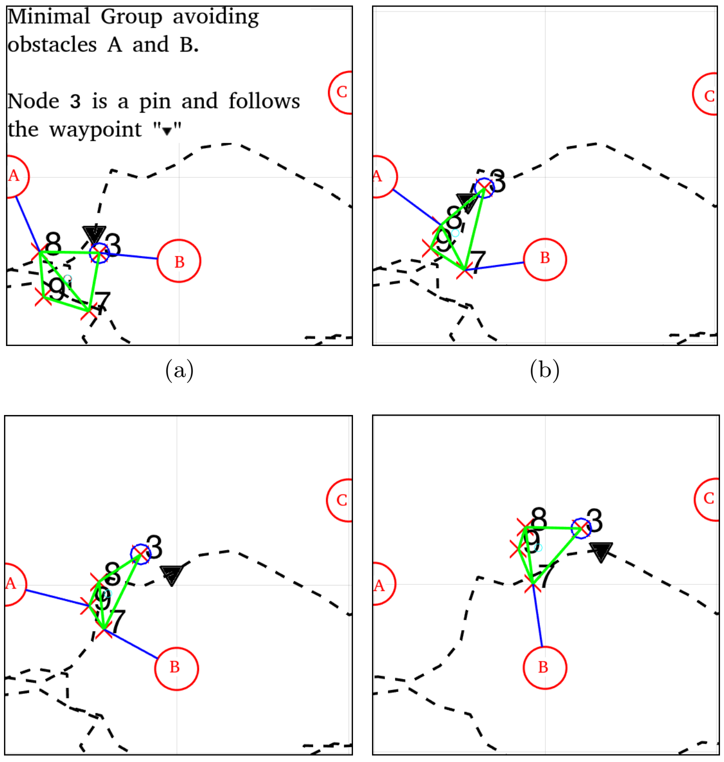

Minimal Group avoiding obstacles A and B.
Node 3 is a pin and follows the waypoint "▾"
A
B
C
3
7
8
9
(a)
(b)
(c)
(d)

Figure 4.5: Minimal group avoiding obstacles and maintaining cohesion.

both approaches.

Fig. 4.6 shows the resulting tracking error (distance between the *geometric center* of the ensemble and the *goal position*), during one of the iterations. Only the last ten seconds of simulation are shown in the figure to highlight the difference between both methods. It is possible to see that, for the same initial configuration, the swarm moved slower to the next waypoint using the randomly selected pin.

Fig. 4.7 shows the magnitude of the control effort vector $|\boldsymbol{u}_i|$ for the pin node. In the figure, one can see that the selection of a non-optimal pin node (random) yields a higher effort to be exerted by the pin node.

Table 4.2 shows the average RMS value of the Euclidean distance from the swarm geometric center to the target location (tracking error). However, only analyzing the geometric center does not provide a clear idea of the behaviour of

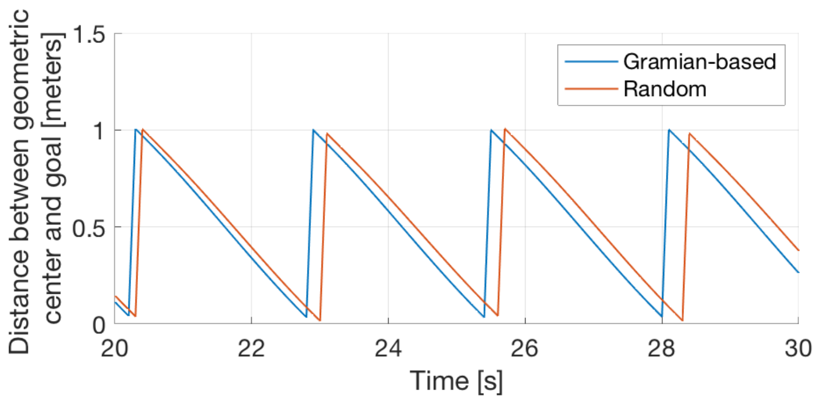


Figure 4.6: Tracking error of the geometric center using two different pin selection methods.

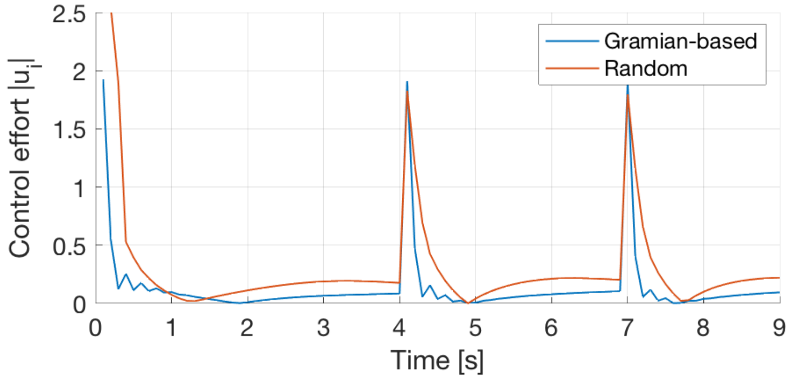


Figure 4.7: Control effort of the pin node using two different pin selection methods.

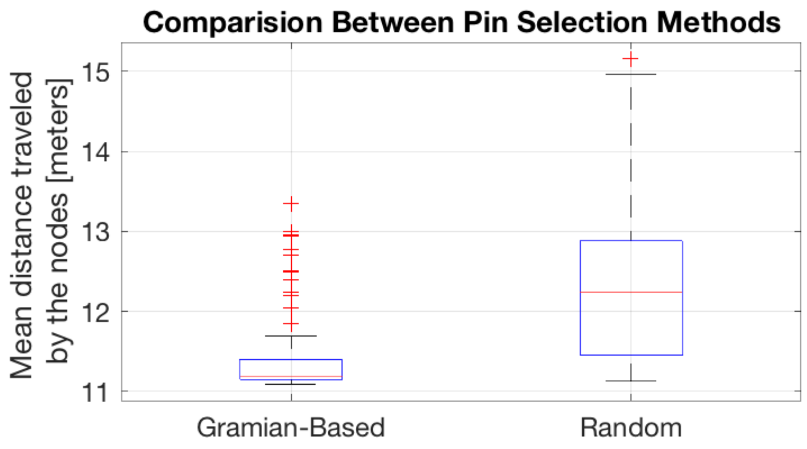


Figure 4.8: Result of 100 iterations using the two pin selection methods.

the whole swarm during navigation. Therefore, the distance travelled by each agent was also analyzed. Let us consider $\boldsymbol{p}_i^t$ as the total distance travelled by the node i during operation. Thus, $\bar{p}^t = (1/N)\sum_{i=1}^{N} \boldsymbol{p}_i^t$ is the average distance travelled by all agents in the swarm.

Table 4.2: Tracking error for different pin selection methods

Pin selection strategy	RMS position error [meters]
Random selection	0.6771
Gramian based	0.6632

Fig. 4.8 shows a boxplot with the average travelled distance ($\bar{p}^t$) after 100 iterations. One can interpret from the figure that selecting a random robot as pin caused all agents in the swarm to move more than using a Gramian-selected pin. The higher travelled distance is the result of the whole group rearranging its positions to accommodate the motion of the pin node. For example, the randomly selected pin robot could be physically located at one end of the network, and its motion may cause the whole group to rotate, or even force network connections to be broken/created.

4.4 Conclusion

In this chapter, the guidance of a swarm of robotic structure parts in an environment with obstacles was studied. The problem was approached using pinning control (applying the navigational control term in only one node) to guide the swarm to the goal position. The parts were approached by their simplified model given that the focus was on the group dynamics and behaviour rather than individual characteristics. A control strategy was proposed that: actively breaks the network connections creating new groups, dynamically defines new pins, and maintains group cohesion while navigating in space. The effectiveness of the approach was demonstrated using simulation trials. Monitoring the agent's connections (distances to neighbours) and actively deciding on splitting a group is shown as a valid approach for avoiding unnecessary group fragmentation while navigating an environment with obstacles.

Moreover, the Gramian-based selection of new pins when compared to the random selection showed to be a viable approach for pin selection. The Gramian-based approach resulted in selecting a node that: i) guided the group to its objective with a smaller overall error, ii) required less "control energy" to control the group, and iii) required less motion (distance travelled) from all robots.

Note that this chapter focused on movement of robots using ideal information and the noises associated with sensors' outputs are not discussed. However, a certain level of robustness to the control operation can be expected given that in equation (4.2) the control signal is computed as a sum of different measurements. It is a combination of the measured states of other robots and obstacles in the environment. Therefore, individual measurement errors tend to be diluted into the calculation, and the control signal reflects the *average* knowledge about the states of other entities around the robot.

The results presented in this chapter can be compared with platooning of agents [89, 90]. Platooning is a viable way of transposing an environment with obstacles, once the vehicles align themselves in a straight line and can avoid obstacles more easily. However, the area covered using platooning is smaller, when compared to the swarm groups presented here. The distributed displacement of the vehicles in a swarm is advantageous in certain applications, such as surveillance. Also, the time delay of the information in the platooning members could be a problem for bigger groups. In our approach, the active rupture of the network and the cohesion control term help to solve such time delay issues.

Finally, guidance and control strategies discussed in this chapter enable the autonomous structure parts to traverse the environment, reaching the assembly location. At the end of operation, all robots successfully and efficiently arrived at the goal, while minimizing communication and power consumption. The trajectory taken by the swarms and the power consumed to move are relevant for the next stages of the assembly, given that a robot must have enough energy to place itself in a structure. Also, robots finalize their moving stage by forming a connected graph and waiting on top of the assembly location, which we denote as the waiting stage. The communication between robots in the waiting state is crucial to algorithms in the ordering stage to be discussed in the next chapter.

5 Ordering of robots to assemble structures

In this chapter, we study another area related to the autonomous assembly process, which involves establishing premises and rules for the sequence of placement of parts in the structure. Two possible strategies stand out regarding how parts are added in a self-assembly process. Parts can be inserted all at once in the environment and locally compute how to rearrange their positions to form the structure [10], or new parts can be sequentially added to the structure, one by one [11].

Adding parts one at a time enables the directed growth of the structure from a seed position. Directed growth assemblies are independent from the final structure and initial configuration, being applicable to various shapes [40, 11]. In this thesis, it is assumed that new blocks sequentially move to a structure being assembled and directed growth is obtained at every layer of the structure (see Chapter 6).

This chapter focuses on how to compute a sequence of which robots move to a structure being assembled. As a result of the procedure described in Chapter 4, it is assumed that a group of structure parts moved in space in a coordinated manner. After moving, robots arrive at the *assembly location* and hover above the ground in a *waiting state*. The *assembly location* is a region in space where one or more structures are being assembled simultaneously. At the end of the movement, the swarm forms a connected graph, i.e, there is a path for communication between every pair of robots. Also, there is one pin robot.

The objective is to design an assignment strategy that computes the next robot in a swarm that will be selected to perform an assembly task, starting from a waiting state. Note that an order is created between the robots in the swarm by executing the assignment strategy iteratively. Moreover, after exiting the waiting state, robots move in the environment towards the structure being assembled. As they arrive at the structure, robots start the self-assembly procedure, studied in Chapter 6.

This thesis seeks an assignment approach that could be applied to real-world robots. As expected, these robots need power to operate. Commonly, that is provided by a battery embedded in the robot. In our case, UAVs use battery energy to move in the environment, sense its surroundings, communicate with other robots, and perform internal computation. To move to the assembly location it is expected that the battery level of each robot has decreased depending on the path taken. That path, however, is nondeterministic and changes from robot to robot as the interactions between agents and with obstacles may lead to unpredictable trajectories. Regardless of the trajectory, each robot must have enough energy to continue executing the assembly process. Thus, the distance from the waiting position of a robot to the structure being assembled and the current battery level are relevant factors when computing the robot order.

The ordering strategy must be able to compute the sequence of which robots perform the assembly. In this chapter, at any given moment, the most suitable robot to execute the assembly task is identified by using an auction process. Auction-based approaches are applicable on centralized and decentralized decision methods and are known to be computationally efficient [66, 70]. In our case, the auction uses a score function that is based on the battery level of a robot and its distance to the structure being assembled. Furthermore, the existing communication network is exploited by the decision algorithms to exchange the necessary information.

Given the simplicity of algorithms and messages proposed, the auction strategy does not compromise the communication bandwidth or the computational capacity of robots. Simulated results show the efficiency of the auction-based assignment strategy when compared to a random selection of robots. To compare the performance of different strategies we analyse the energy consumed by robots while waiting to be assigned to a structure. The simulations performed the assignment of robots to assemble three structures simultaneously. The number of robots being requested by each structure varied from 5 to 100 robots. Results show that the auction strategy is more energy efficient, leading robots to save battery power.

5.1 Problem Formulation

Assume that the swarm is organized in a proximity-based network topology and that sufficiently close robots can communicate. Thus, the relationship between robots can be described as a spatial graph, $G = (V, E)$, where each robot is a vertex $i \in V$. In addition, edges $e_{ij} \in E$ exist iff robots i and j are within a sensing/communication range of each other.

Robots can communicate using such network topology directly or indirectly, by having the messages relayed through others. Moreover, after arriving in the assembly area, there is only one pin node. The pin has access to the current status of the network (its topology), and the is capable of receiving messages from robots on the ground (in structures being assembled).

Denote by $\mathbb{S}$ the set of structures being assembled at any given moment. A structure $\mathtt{s} \in \mathbb{S}$ is an arrangement in space where robots place themselves, working as parts. The first position in the structure occupied by a block, on the bottom level, is called a seed position. The seed is the origin of the O_B coordinate system (as per Section 3.5) and it marks a position where the structure grows from (see Chapter 6). Furthermore, the position of a structure in space is given by $\boldsymbol{p}^{\mathtt{s}}$ (see Fig. 3.5).

Robots can be grouped into two separate sets, $R^u \subset R$ as the set of unassigned (available) robots and $R^a \subset R$ as the set of robots already assigned to a structure. Moreover, a robot is either available or assigned, $R^a \cap R^u = \emptyset$ and $R^a \cup R^u = R$. Given that, the main research question is from the set of robots available at any given instant t, which robot, r_i, can be assigned to which structure, $\mathtt{s}$, being assembled?

The assignment process describes the action of commissioning a robot to a structure. After assigned, the robot first moves to the location of $\mathtt{s}$ in space, $\boldsymbol{p}^{\mathtt{s}}$, and then starts its self-assembly procedure at arrival. A robot that was already assigned is considered to be in R^a and not R^u. Note that, the vertex set in V is only composed of robots in R^u. Also, network connections (E) are broken and the topology of G is affected every time a robot is assigned and moves away from other waiting robots.

Under this context, the structures being assembled can be denoted as a set of tasks that must be executed by the available robots. A structure with $N^{\mathtt{s}}$ parts yields $N_t^{\mathtt{s}}$ tasks to be executed. Let us assume that N_t is the combination of the number of tasks of all structures, $N_t = \sum_{\forall \mathtt{s} \in \mathbb{S}} N_t^{\mathtt{s}}$. Moreover, there are as many robots, N_R, as positions to be filled (tasks to complete), thus $N_t = N_R$. Given that each robot is a structure part, a robot can be assigned a maximum of one task. After assigned a block can not be decommissioned. The assignment is said to be completed once all robots have been assigned to a structure.

The task to be executed can be decomposed into two steps: *i)* the robot moves to the structure that it was assigned to at position $\boldsymbol{p}^{\mathtt{s}}$; and *ii)* it executes the self-assembly procedure. The first part of the task can be executed using properly designed control laws, as discussed in Section 3.3. The second part of the task requires the block to find the specific location to place itself in the structure. Such procedure involves a specific set of data and behaviours that is discussed in Chapter 6. The length of the task, i.e, how long it takes for a robot to execute both steps, is not relevant to the assignment problem, given the fact that a robot cannot be decommissioned.

The assignment problem can be written as an integer program with binary decision variables, x_{ij}, that indicate whether or not task j is assigned to robot i. In addition, a score function, c_{ij}, is used to represent the cost to assign a robot to a task. Therefore, the global objective function is assumed to be the sum of local reward values (on each robot),

$$max \sum_{i=1}^{N_R} \left(\sum_{j=1}^{N_t} c_{ij} \mathsf{x}_{ij} \right) \tag{5.1}$$

subject to

$$\begin{aligned} \sum_{j=1}^{N_t} \mathsf{x}_{ij} = 1 \qquad & \forall i \in \mathcal{I} \\ \sum_{i=1}^{N_R} \mathsf{x}_{ij} = 1 \qquad & \forall j \in \mathcal{J} \\ \sum_{i=1}^{N_R} \sum_{j=1}^{N_t} \mathsf{x}_{ij} = N_t = N_R & \\ \mathsf{x}_{ij} \in \{0, 1\} \qquad & \forall (i,j) \in \mathcal{I} \times \mathcal{J} \end{aligned} \tag{5.2}$$

where $\mathsf{x}_{ij} = 1$ if agent i is assigned to task j and 0 otherwise. Also, the index sets are defined as $\mathcal{I} = \{1, ..., N_R\}$ and $\mathcal{J} = \{1, ..., N_t\}$. Note that $\mathcal{J}$ encodes the part requests for all structures being assembled. The summation term inside the parenthesis in (5.1) represents the local reward for the agent i. A general version of the global objective function where reassignment is possible can be found at [66].

Finally, a task is generated every time a structure under construction requests a new part. We assume that the seed robot, located at the bottom layer can communicate with the pin. Thus, seed robots of the structures under construction send requests for new parts to the pin from time to time. The pin then gathers requests from all structures in $\mathbb{S}$ and act as an auctioneer. A discussion about how often a structure can request a new block is postponed until Chapter 6.

5.2 Auction-Based Assignment

In this section, it is proposed an auction-based algorithm to solve the assignment problem. The auctioneer is responsible to decide which robot must execute the task at a given moment. Moreover, the assignment strategy can be broken into some steps: *i)* the minimum spanning tree (MST), Γ, is computed on G (spatial graph given swarm disposition); *ii)* from the set of the robots unassigned R^u, the robots that are leaves in the MST are identified; *iii)* new requests for the parts are received by the auctioneer; and *iv)* the pin runs an auction to select the available robot to be assigned to the structure.

The MST is interesting in this scenario for two reasons. First, it helps selecting a path for the information to flow in the network. Second, by analyzing the structure of the tree, the robots that are crucial for the flow of information are identified. Note that as robots are assigned, they move to the structure location, leaving the ensemble and breaking their connections with the nodes in G. The computation of MSTs is a widely studied area with a range of algorithms being proposed over the years [91, 92]. After computing Γ, the root of the tree is set to be the pin node and the nodes with no children are the leaves. Note that if a leaf node is removed, the flow of information between other nodes in the tree is not affected. Denote by R^u_l the set of "leaf unassigned" robots, being $R^u_l \subseteq R^u$.

Figure 5.1 shows the different stages of the process. On the left side, the robots are displayed assuming they just arrived and are in the waiting state. Note that the swarm forms a connected graph, and hovers above the assembly location. On the right side, the pin node computes the MST and its topology is highlighted. Also, the leaves (robots in R^u_l) in the tree are marked with "*".

There is a blueprint, B^{s}, associated to every structure in $\mathsf{s} \in \mathbb{S}$ being assembled. The pair *position* in the environment and associated *blueprint* ($\boldsymbol{p}^{\mathsf{s}}$,$B^{\mathsf{s}}$) for each structure can be already stored on the auctioneer's memory, or be provided by the operator at the beginning of the ordering stage. At the beginning of the operation, the information $D = (\boldsymbol{p}^{\mathsf{s}},B^{\mathsf{s}})\ \forall \mathsf{s} \in \mathbb{S}$ is broadcast over the network, and is stored by each robot in the swarm as a *dictionary* position-blueprint. When a robot is assigned to move to structure (move to $\boldsymbol{p}^{\mathsf{s}}$), it will look into such dictionary and retrieve the blueprint that will be assembled at that location. Furthermore, the first round of robots must be sent to start the assembly of each structure. As a result, the first assignment round happens without any request, with the auctioneer iteratively assigning leaves in Γ (R^u_l) by providing the location of the structures in space, $\boldsymbol{p}^{\mathsf{s}}$. These robots will be the first to arrive on the structures. Therefore, they will act

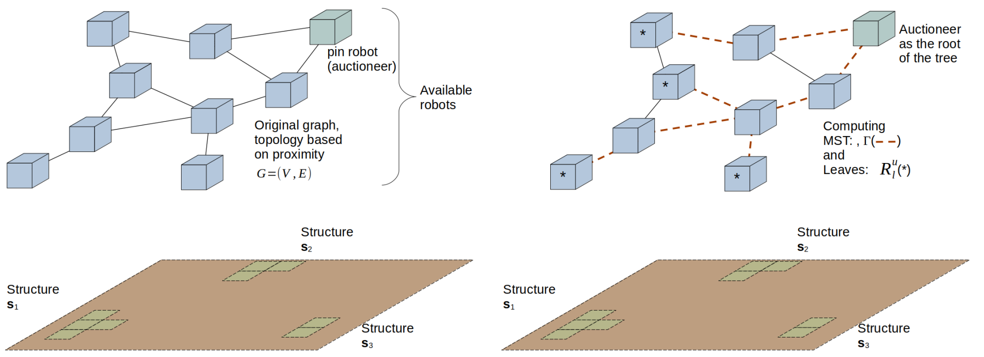

pin robot (auctioneer)
Available robots
Original graph, topology based on proximity
$G=(V,E)$
Auctioneer as the root of the tree
Computing MST: , Γ(- -) and Leaves: $R_l^u(*)$
Structure $\mathbf{s}_1$
Structure $\mathbf{s}_2$
Structure $\mathbf{s}_3$

Figure 5.1: Connections in the graph showing the start topology (left side), the computation of the tree and the leaf nodes (right side).

as the bottom layer seed robots placing themselves at the seed location and requesting new parts for the structures. Each seed robot requests parts by messaging the auctioneer.

The requests for structure parts are handled through an auction process. In auction methods, the bids are submitted to an auctioneer to determine the winner based on the highest bids, given

$$i^* = argmax_i \; c_{ij}(i, \boldsymbol{p}^{\mathsf{s}}) \tag{5.3}$$

where i^* is the selected robot. The task scores c_{ij} are based on the reward of assigning task j to robot i,

$$c_{ij}(i, \boldsymbol{p}^{\mathsf{s}}) = \frac{1}{bat_i} + \frac{1}{||\boldsymbol{p}_i - \boldsymbol{p}^{\mathsf{s}}||} \tag{5.4}$$

where $||\boldsymbol{p}_i - \boldsymbol{p}^{\mathsf{s}}||$ refers to the Euclidean distance between the robot r_i and the structure s that generated the task j. Also, bat_i is the current battery level of the robot. Note that, to compute the score value, c_{ij}, a robot needs both an internal parameter (battery) and a information sent from the auctioneer (position of the structure).

Allow us to define the set of messages that circulate between robots during the auction process. Denote by m_a^{na} the message sent *from* the auctioneer *to* non-auctioneer robots. Depending on the stage of the auction process, the message that is broadcast over the network can be

$$\mathrm{m}_a^{na} = \begin{cases} \boldsymbol{p}^{\mathsf{s}} \;, & \text{the position of the structure that generated task } j \textit{ or} \\ i^* \;, & \text{the winner of the auction process.} \end{cases} \tag{5.5}$$

At the beginning of the auction, the auctioneer sends the position of the structure that requested a part, $\boldsymbol{p}^{\mathsf{s}}$. That information is then used for the non-auctioneer robots to compute their bids. Afterwards, auctioneer messages other robots with the winner id of the auction process, i^*. This message is used by the winner robot to move to the structure.

Denote by m_{na}^{a} the message that is sent back *from* non-auctioneer robots *to* the auctioneer. The message contains both the id of the robot and its bid (score function value),

$$\mathrm{m}_{na}^{a} = [i, c_{ij}(i, \boldsymbol{p}^{\mathsf{s}})]. \tag{5.6}$$

Finally, a request for a new part from a structure already placed on the ground contains the position of the bottom layer seed,

$$\mathrm{m}_{\mathsf{s}}^{a} = \boldsymbol{p}^{\mathsf{s}}. \tag{5.7}$$

Let us assume that each request for a part is received by the auctioneer in an asynchronous process, separate from the auction process. As such, every time message $\mathrm{m}_{\mathsf{s}}^{a}$ arrives, the position of the structure requiring a robot ($\boldsymbol{p}^{\mathsf{s}}$, content of message) is placed in a queue, Q. Note that Q is a first-in-first-out queue of positions of structures being assembled. Therefore, the auctioneer algorithm receives a new task by dequeuing a position from Q.

Algorithm 2 assumes that each agent is aware of its own unique identification number in the swarm. Thus, when initiated, the algorithm is given the robot's id. On the other hand, the auctioneer only broadcasts messages to the network, and its algorithm does not need to be initialized with the robot's id. The assignment procedure happens as follows. First, the request of a new part is obtained from the queue (Algorithm 3 line 4). Then, the auctioneer sends the position of the structure to all robots (Algorithm 2 lines 5 and Algorithm 3 line 8). Robots compute their scores, c_{ij} and send their bids in return (Algorithm 2 lines $7, 8$ and Algorithm 3 line 9). Note that every robot bids only once. Bids from robots that are not leaves are discarded and the lowest cost is selected by the auctioneer as the winner (Algorithm 3 lines $11, 12$). The auctioneer broadcasts the result of the auction and updates its internal representation of Γ, removing the winner robot from the tree (Algorithm 3 lines $13, 14$). Non-auctioneer robots receive the result of the auction and the winner robot moves to the structure location (Algorithm 2 lines $9, 10, 11, 15$) If the only robot available at the moment is the auctioneer, then the robot assigns itself and moves to the structure. Finally, after moving to the structure location, the robot retrieves the blueprint to be assembled from the dictionary (D) stored in its memory (Algorithm 2 line 16 and Algorithm 3 line 19).

Algorithm 2 Non-auctioneer algorithm

```
 1: function WAITING_STATE($robot\ id, D$)
 2:     $i \leftarrow robot\ id$
 3:     assigned ← false
 4:     while not assigned do
 5:         Wait for message from auctioneer with structure pos. ($\text{m}_a^{na} = \boldsymbol{p}^{\mathsf{s}}$)
 6:         if message received then
 7:             Compute local score, $c_{ij}(i, \boldsymbol{p}^{\mathsf{s}})$, using equation (5.4)
 8:             Send bid to auctioneer, $\text{m}_{na}^{a} = [i, c_{ij}]$
 9:             Wait for message from auctioneer with winner id ($\text{m}_a^{na} = i^*$)
10:             if message received and $i^* = i$ then          ▷ this is the winner
11:                 assigned ← true
12:             end if
13:         end if
14:     end while
15:     Move to $\boldsymbol{p}^{\mathsf{s}}$
16:     Retrieve blueprint $B^{\mathsf{s}}$ from the dictionary stored ($D$)
17: end function
```

5.3 Algorithm's analysis

In this section, we analyze the proposed algorithms to prove that they successfully assign all robots to structures.

Lemma 1. *For every structure part request (task j) there is always a robot that can be assigned,* $\forall j \leq N_t$

Proof. Recall that the total number of tasks being requested (N_t) equals the total number of parts in all structures to be assembled. Also, there are as many robots in the swarm as parts needed for each structure, $N_t = N_R$ (as per Property 7). If a robot is still not assigned, it is part of the R^u. Additionally, robots in the swarm form a connected graph, G, on which the MST, Γ, is computed. In a finite rooted tree where $|R^u| > 1$, there is always at least one leaf node. Thus, $R_l^u \subseteq R^u$ is never $\emptyset$. If $|R^u| = 1$ the auctioneer is the only robot in Γ. Given Algorithm 3 line 5, in that scenario, the auctioneer will assign itself. □

Lemma 2. *Algorithm 3 sequentially assigns all robots.*

Algorithm 3 Auctioneer algorithm

```
 1: function AUCTION($D$)
 2:     assigned ← false
 3:     while not assigned do
 4:         dequeue a request (position $\boldsymbol{p}^{\mathsf{s}}$) from $Q$            ▷ task $j$
 5:         if $|R^u| = 1$ then       ▷ the auctioneer is the only robot available
 6:             assigned ← true
 7:         else
 8:             Broadcast structure position, $\mathrm{m}_a^{na} = \boldsymbol{p}^{\mathsf{s}}$        ▷ request bids
 9:             Wait for message $\mathrm{m}_{na}^{a}$ containing $c_{ij}$ from all $r_i \in R^u$
10:             if message received from all robots then
11:                 Select only bids ($c_{ij}$) for robots in $r_i \in R_l^u$, ignore others
12:                 Compute the winner ($i^*$) using equation (5.3)
13:                 Broadcast winner's id, $\mathrm{m}_a^{na} = i^*$   ▷ assign winner to task $j$
14:                 Remove the vertex $i^*$ from the tree ($\Gamma$)
15:             end if
16:         end if
17:     end while
18:     Move to $\boldsymbol{p}^{\mathsf{s}}$
19:     Retrieve blueprint $B^{\mathsf{s}}$ from the dictionary stored ($D$)
20: end function
```

Proof. In the auction process for each requisition, at least one of the robots in R_l^u will have a maximum score value. Thus, one of the nodes with the maximum score is always assigned and it is removed from R_l^u. As per Lemma 1, at every request one robot will be assigned (see Algorithm 3 lines $6, 12$) . The number of robots in R^u reduces by 1 per iteration. There are only a finite number of nodes in R^u, thus, the size of the set will eventually reach 0. □

Theorem 1. *All structures will receive the sufficient number of parts to be assembled.*

Proof. The assembly starts with no robots assigned to structures, $R^a = \emptyset$. As per Lemma 2, at every request $|R^u|$ decreases by 1 and the number of robots assigned, $|R^a|$, increases by 1. Thus, eventually, all nodes are selected. A structure can only generate as many requests as there are spots to be filled by a robot. Thus, after all requests have been completed, all robots have been assigned and there are no positions yet to be filled by a robot. □

5.4 Results

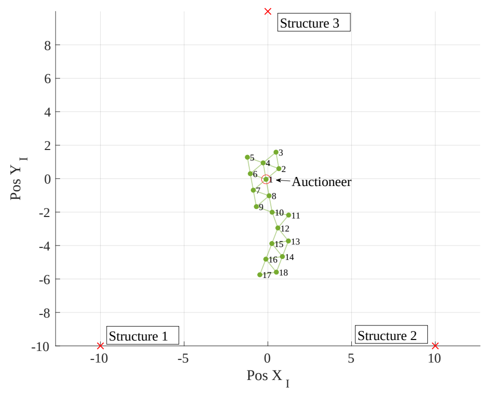
In this section, simulation trials are used to analyze the performance of the auction-based assignment algorithms. Moreover, the proposed strategy was compared with a random selection methods of available robots.

The simulations performed used three different scenarios. In each scenario, the number of parts needed to assemble the structures changed, but the position where structures should be assembled remained the same. Table 5.1 shows the number of parts per structure for the different experiments. Note that only the number of parts of the structures is relevant to the assignment problem and the shape of the structure is a topic of study in Chapter 6. The position of structures are $\boldsymbol{p}^1 = [-10, -10, 0]\ m$, $\boldsymbol{p}^2 = [10, -10, 0]\ m$, $\boldsymbol{p}^3 = [0, 10, 0]\ m$, in the O_I frame.

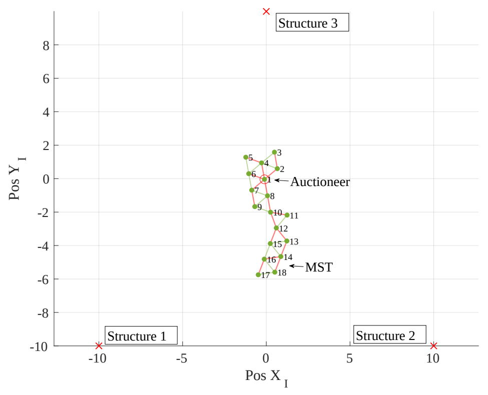
The procedure described in Chapter 4 ends with the swarm flying above the assembly location. Therefore, in simulation, robots were initiated at the waiting state at random locations in space, hovering above ground at 1 m. The minimum distance between robots was set to 1 m, and any robots within the sensing range of each other (1.33 m) were connected in the graph topology, G. Robots are instantiated in space such that G is a connected graph. Moreover, the pin was chosen using the NCG metric, following Section 4.2.2. Finally, robots start the simulation with random battery levels, varying from 50% to 90%.

Two simulation runs were executed using the same initial disposition of robots, one with available robots being randomly selected and one using the proposed auction-based strategy. To compare both strategies being evaluated, The algorithms proposed are executed almost the same way, with exception of the auction winner. For the random selection method, instead of a using the bids from robots, a random robot in R_l^u was assigned (Algorithm 3 line 12). With that, the MST component of the algorithm is still used, guaranteeing that there is always a robot in the swarm that can be assigned to a structure (see Section 5.3). To create a more realistic scenario, the battery level of the robots decreased while they were active. First, the battery would be reduced by 0.1% per iteration, while the robot is hovering above the assembly area. Second, when a robot moves to a structure, the battery level would be discounted an amount proportional to $||\boldsymbol{p}_i - \boldsymbol{p}^{\mathbf{s}}||$. Note that an iteration represents the processing of fulfilling one part request, i.e., from reception of a task until a robot was assigned and moved to the structure location. Finally, the queue Q was initiated with the requests from all structures randomly ordered.

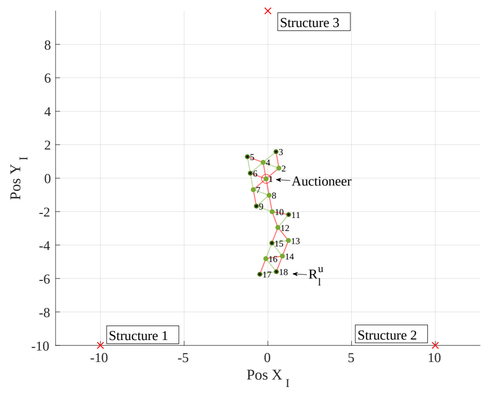
Figure 5.2 helps describe the assignment process for one robot. It shows



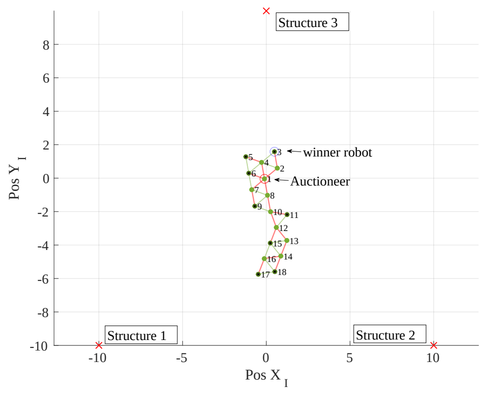
(a) Start of the assignment



(b) Computation of MST



(c) Identifying leaves in the tree

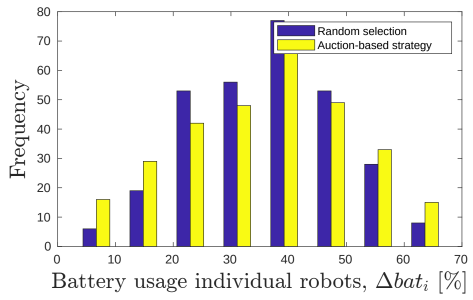


(d) Selection of a winner bid after all bids were received by the auctioneer

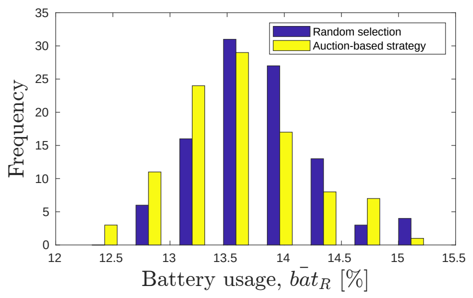
Figure 5.2: The stages of the assignment process.

the displacement of robots at the beginning of operation, the computation of the MST and the leaf nodes. It also shows a robot that was selected after all bids were received by the auctioneer. Note that the winner will be removed from the network as it moves to the structure location and robot 2 will take its place as leaf.

The average battery use ($\bar{bat}_{use}$) is selected to analyze the performance of the proposed strategy. Given that the initial level of robots' batteries are randomly defined, the battery level decrease (battery usage while waiting and to move to structure) is more relevant than focusing on absolute final values.



(a) Individual battery usage on a single simulation run.



(b) Average battery usage over 100 trials with different initial conditions

Figure 5.3: Battery usage distributions.

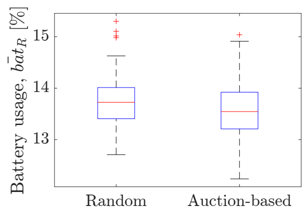
The metric is computed as

$$\begin{aligned} \Delta bat_i &= bat_i^{initial} - bat_i^{final} \\ \bar{bat}_R &= \tfrac{1}{N_R}\sum_{i=1}^{N_R} \Delta bat_i \\ \bar{bat}_{use} &= \tfrac{1}{N_e}\sum^{N_e} \bar{bat}_R, \end{aligned} \tag{5.8}$$

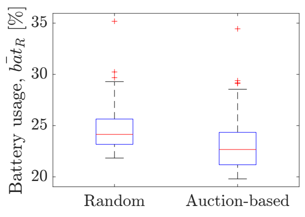
where $bat_i^{initial}$ and bat_i^{final} are the initial and final values for robot's i battery, respectively. The final battery level is the value that each robot has when it reaches the structure location. $\bar{bat}_R$ is the average battery usage among robots in a single simulation run and $\bar{bat}_{use}$ is the mean value of $\bar{bat}_R$ after multiple iterations with different initial conditions. Also, N_e is the number of simulation runs and it was set to $N_e = 100$ for our trials.

Fig. 5.3 shows two histograms related to the battery usage during the assignment process. On the left side, the battery usage of individual robots during a task assignment process is presented. One can see that the energy consumption among all robots participating in the assembly follows a normal distribution, regardless of the assignment strategy. Given the curve profile, the mean energy consumption within a run can be computed (obtaining $\bar{bat}_R$). The second histogram shows that $\bar{bat}_R$ also followed a normal distribution over the multiple repetitions of the experiment. Which, by its turn, enables the analysis of the $\bar{bat}_{use}$ metric proposed above.

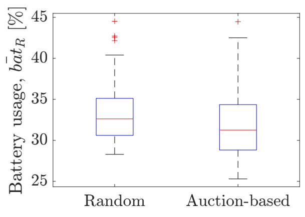
Fig. 5.4 presents boxplots comparing the assignment strategies for the three experiments proposed. Notice that, given the position of the blue boxes (the majority of the data) and the red line (median value), there is a tendency for robots to consume less energy when applying the auction method proposed.



(a) Experiment 1



(b) Experiment 2



(c) Experiment 3

Figure 5.4: Comparison between assignment strategies with $\bar{bat}_R$ values for the three experiments.

Table 5.1: Parameters for simulation

Experiment	Number of robots per structure	N_t
1	5, 6, 7	18
2	50, 60, 70	180
3	100, 100, 100	300

Table 5.2: Average battery decrease while a robot wait to be assigned

Experiment	$\bar{bat}_{use}$	
	Random selection	Auction-based strategy
1	13.76%	13.59%
2	24.67%	23.13%
3	33.48%	31.96%

However, the standard deviation of $\bar{bat}_R$ was higher for the auction assignment strategy, which is related to sample skewness. Specifically, on experiments 1 and 3 the values of $\bar{bat}_R$ clustered close to the median value and distorted the Gaussian distribution, which changes the standard deviation values computed.

Table 5.2 summarizes the result of the simulation trials, focusing on the average battery usage after multiple iterations. Notice that, the average battery usage is smaller when the proposed approach is used. A smaller battery usage implies that the robots that are assigned to structures have, on average, more power to keep operating and execute the self-assembly algorithms.

5.5 Conclusion

This chapter discussed the assignment problem for a swarm robotic structure parts. An order of which the robots move to each structure being assembled has to be collectively computed. The need for an assigning strategy comes from the fact that robots must arrive at the structure location in sequence, one after the other. Consequently, some robots have to stay in the waiting state longer than others.

The assignment problem was modeled as an integer program with binary decision variables that indicated either a robot was assigned to a structure or not. This task assignment problem fits the particular case of single-assignment in the literature, where decommissioning is not possible and each robot is assigned only once. An auction-based assignment solution was proposed and the selection of robots used a minimum spanning tree computed from the communication topology. The tree helped to identify the robots in the swarm that could be removed without breaking significant network connections, stopping communication between robots. Also, the auction process was used to identify the most suitable robot to be assigned among those available. In it, a score function was proposed to evaluate candidate robots that combined distance to the structure and battery level of the robot.

It was first showed that network communication is guaranteed if only leaves of the spanning tree were selected and assigned. Moreover, in the results section, simulation trials showed that the strategy proposed is more efficient than a baseline approach. The auction-based assignment was more energy-efficient than the random assignment strategy, leading the robots to, on average, save battery power at the end of the ordering stage.

After performing the ordering procedure described, we assume that robots have been correctly assigned to a structure in the environment. Also, robots move to the structure position, $\boldsymbol{p}^{\mathsf{s}}$, while carrying the structure blueprint, B^{s}. Note that, by assigning agents one by one, the ordering stage was capable of defining a sequence of which robots to arrive at the structure location. After arrival, robots must start *placing* algorithms, which enables them to search for empty positions in the structure using the blueprint as a reference. Placing related algorithms are the topic of discussion in the next chapter.

6 Self-assembly of three-dimensional structures

The correct placing of parts in the structure is a crucial stage of the assembly problem. This chapter focuses on placement strategies for structure parts that are autonomous self-actuated robots. Each robot is capable of sensing, communicating, and actuating on the environment. A common challenge on self-assembly is to define specific behaviours and rules of interaction between robots and with the environment, such that the structures are correctly built [13, 10].

Systems based on local interactions tend to be more robust to noises and failures [46, 43, 9, 11]. These adverse conditions can be either locally identified and corrected or automatically suppressed by the adding redundancy to the system (increasing the number of parts).

In this chapter, self-assembly algorithms are proposed with rules based on local interactions. The algorithms use the structure blueprint with the desired shape (described in Section 3.5) to guide the movement of robots. Also, the graph representation is used to derive the necessary proofs of convergence of the assembly process.

This chapter focuses on the assembly procedure of a single structure. Recall that, one or more structures could be assembled simultaneously in the assembly location. Also, as result of the assignment procedure, described in Chapter 5, robots are sent to different structures as needed. Thus, if multiple structures are being assembled at the same time, each one is assumed to be an independent process. Each instance contains its own set of robots operating based on a particular blueprint. In addition, for every structure, the assembly is executed in layers from bottom to the top and, on each layer, blocks sequentially place themselves starting from a seed position.

The contributions of this chapter are related to the three-dimensional layered assembly using autonomous parts. The assembly algorithms presented should be programmed at each robot and encapsulate a set of behaviours to guide the robots' motion. The behaviours are triggered at an specific order, enabling a single robot to place itself without global knowledge of the structure's state. Moreover, mathematical proofs are derived based on the graph representation of the structure, showing that the self-assembly process leads to the construction of a desired shape.

Simulated results show the efficacy of the self-assembly procedure by comparing the traveled distanced of robots with best and worst paths to be taken by the robots. Also, network controllability is used to investigate the influence of the correct selection of seed positions on the assembly process. Finally, experimental results demonstrate that the algorithm is applicable to real-world scenarios.

6.1 Problem Formulation

This section shows the fundaments of the self-assembly problem where robots are the structure parts. Moreover, the focus is on blueprint-based assembly and the characteristics of the construction process that can be derived from it. The blueprint notation, its associated graph, and some of its properties were previously defined on Section 3.5.

For simplicity, let us consider that the placing process is discrete, defined by time steps $t = 1, ..., t_f$. Note that t_f defines the instant when all blocks have been placed and the autonomous construction finishes. The assembly starts when the first robot places itself in a seed position, in the bottom layer ($\boldsymbol{\rho}_{s_1}$). As a result of the procedure on Chapter 5, robots arrive adjacent to $\boldsymbol{\rho}_{s_1}$, sequentially. Also, the seed position is always the first position that a robot will occupy in a layer (enabling directed growth, as discussed in Section 2.2). Robots scan possible block positions, searching the best location to occupy in the structure (under some metric). Given the sensing range, a robot can only scan a position when it is physically adjacent to it. Fig. 6.1 shows that a robot is able to detect objects or empty spaces that are adjacent to it on its four lateral sides (red arrows represent occupied positions and green arrows unoccupied.).

The block-shaped structure parts and the discrete nature of the positions in the blueprint result in the placing process taking place in a grid-like environment. Moreover, the movement of a robot is limited to 1 grid space at every time step. Thus, the movement between any two positions in the grid should be performed sequentially, one unit at a time.

The assembly is layered, from bottom to top. Thus, a robot $r \in R$ should

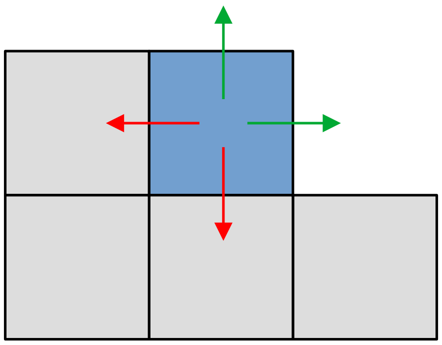


Figure 6.1: Local sensing of a robot with a top view perspective

search for unoccupied spaces in $P_l = P_1$ first, and sequentially move to P_{l+1}. The layered assembly ensures that the assembly of lower layers start before upper ones.

Robots carry the structure blueprint, i.e., P, and the seed locations, S. A robot uses the structure blueprint as a map to *search* for unoccupied positions to place itself. A *search* is a process where a robot $r \in R$ evaluates positions $\boldsymbol{\rho} \in P$, looking for a currently unoccupied position in the real-world (no other block placed). The term *search* is used because the actual location of an unoccupied spot in the structure is not known when the robot starts the assembly procedure. This also implies that the robots do not have full knowledge of their location in space as elaborated in the next property.

Property 10. *In the placing stage, a robot does not have global localization information (position in the environment); it can, however, compute its own position and orientation with respect to a seed robot.*

A robot $r \in R$ can compute its position, $\boldsymbol{\rho}_r = \begin{bmatrix} x_r & y_r & z_r \end{bmatrix}^T$ in O_B, by counting its grid movements with respect to $\boldsymbol{\rho}_{s_1}$. Following Properties 5 and 6, each robot is capable of local sensing its surroundings, internal computation and communication with a nearby robots. In real robots, embedded sensors could be used to identify movement and compute robots' position, e.g., inertial sensors, odometers, and cameras among others. As expected, these sensors will yield noisy and imprecise information. However, since robots move using neighbouring placed blocks as reference, (Section 6.2), this information can be used to deal with the noise generated by the sensors, e.g, using state observers and sensor fusion techniques [93, 94, 95]. Also, to manage the communication with multiple robots may be unfeasible due to communication errors, signal jamming, limited bandwidth, among other issues. Thus, a design choice is to consider only close range communication and on specif cases (discussed in Section 6.2.1). However a solution based on local communication only is advantageous given that it is still applicable on the cases where global communication is possible.

Finally, recall that Property 8 states that a structure receives as many robots as there are empty positions in to be filled. These robots are assigned by the auctioneer as a response to a part request made by a structure being assembled. Note that, as stated in Property 9, seed robots remain active and, as shown in 6.2.1, they control the number of robots per layer. Thus, it is assumed that bottom-layer seed robot, s_1, requests new parts to the auctioneer. New requests are made after a recently arrived robot communicates with the seed and starts its assembly behaviours, moving away from s_1.

Remark 6. *Note that even though the parts are defined as cubes, other shapes could be easily implemented, with the limitation that all parts are convex and homogeneous. For example, hexagonal prisms could be used in the construction. The differences would be:*

- *the blueprint would be defined in a 3D hexagonal lattice instead of the cubic lattice of Fig. 3.6;*
- *in the graph $G = (V, E)$, each vertex $v \in V$ would have potentially six neighbours instead of four (on each one of the sides);*
- *and the algorithms in Section 6.2 would have to be modified to deal with the new parts; the changes would increase the complexity of the algorithms and make the presentation more intricate, therefore, they are omitted.*

6.2 Self-Assembly

In this section, the behaviours that lead to the assembly of the desired structure defined by a blueprint B are presented. The self-assembly process is executed at each layer of the structure, and layers are assembled sequentially from the bottom to the top. To place themselves, robots follow a process that can be split in three parts: *i*) find the layer currently being assembled; *ii*) circle blocks already placed until an unoccupied desired position is found; and *iii*) move toward the seed position of the layer and stop at the closest unoccupied position.

The self-assembly procedure starts by considering that all robots arrive in the structure from an external position adjacent to $\boldsymbol{\rho}_{s_1}$. To explain the robot's behaviour, let us look at a single robot $r \in R$ that is starting its assembly behaviour. After arriving at a position adjacent to $\boldsymbol{\rho}_{s_1}$, r must move *around* the structure to find an internal position to occupy (part *ii* of the process). By moving *around* those already placed (i.e. moving adjacent), aiming to reach an available structure position, the robot is performing the *circling behaviour*. Note that, the circling behaviour implies a direction of movement, clockwise or counterclockwise when seen from the top, around placed blocks. Recall that r is able to sense its sides, determining if there are objects around it. Thus, to find an unoccupied position, r must move to an adjacent position and then scan for a block already placed.

Consider now that while circling the structure at layer l, r eventually finds a position $\boldsymbol{\rho} \in P_l$ that is unoccupied. Then, r moves to $\boldsymbol{\rho}$ and must decide if it will continue moving or stop and land. Direct growth from the seed position is obtained by sequentially placing all blocks as close as possible from the seed (part *iii* of the process). Therefore, r will evaluate neighbouring positions and iteratively move as close as possible to the seed position.

However, if there are already $|P_l|$ robots in a layer l, then the newly arrived robot r would not find an unoccupied location at level l and should search the next layer, $l+1$. We then introduce the main function of the seed robot (part *i* of the process) — being the layer's *gatekeeper*. The seed is responsible to decide if an incoming robot should stay in a given layer or move to the next. After communicating with seed at layer l, a robot r that was not allowed to circle that layer will then move to layer $l+1$. In the same fashion, r then communicates with the seed of that layer, at position $\boldsymbol{\rho}_{s_{l+1}}$.

Note that the design choice of storing local information in the seed robot decreases the system robustness. That is, there is no way for a seed to detect if a robot has failed after they have already communicated. Thus, if a robot fails to place itself, the seed will not allow extra robots in the layer and the structure will not be completely assembled. There are strategies to mitigate such issues, e.g., extending the communication between robots in the layer until a valid placement is detected. Such discussion is left for future work. However, the seed robots act to speed up the assembly process. This is accomplished because the seed robots add extra information to the autonomous assembly system. This characteristic will be explored in Section 6.3.

6.2.1 Robots' behaviours

Note that all behaviours are repeated by each robot $r \in R$, and for all layers. Also recall that movement is performed discretely in time and in space, *i.e.*, a robot is capable of moving one grid space at every time step.

The main program of the robots is shown in Algorithm 4. In the algorithm, there are three functions that represent different behaviours. Also, one behaviour is applied after the other. The result of each function is used as the input (a start condition) of the next behaviour. After executing all behaviours, the program finishes and the robot turns off.

Algorithm 4 Robots program

```
1: $\boldsymbol{\rho}_{s_l} \leftarrow$ IDENTIFYING_LAYER()                                ▷ Alg. 5
        /* A layer being assemble was found, start circling the block at position $\boldsymbol{\rho}_{s_l}$ */
2: $\boldsymbol{\rho}_r \leftarrow$ CIRCLING($\boldsymbol{\rho}_{s_l}$)                                      ▷ Alg. 7
        /* Robot currently located at position $\boldsymbol{\rho}_r$ in the structure, move close to the layer's seed */
3: MOVING_INTERNALLY($\boldsymbol{\rho}_r$)                                         ▷ Alg. 8
4: *Shutdown*
```

Algorithm 4 starts by calling the **first** behaviour, which must identify the layer being currently assembled. The correct layer is the bottom-most one that has unfilled structure positions. To determine if a block is at the correct layer, an arriving robot $r \in R$ communicates with the seed robot of the layer (positioned at $\boldsymbol{\rho}_{s_l}$) by sending a message m_r^l. This message serves to check the status of the seed robot, similar to a *ping* in a network. Robot r then waits for a response from the seed robot, m_l^r. The possible responses from the seed robot are:

$$\mathrm{m}_l^r = \begin{cases} YES \\ NO \\ no\ response \end{cases} \tag{6.1}$$

Depending on m_l^r, robot r shows different behaviours:

- m_l^r = YES, r circles the current layer;
- m_l^r = NO, r moves to the next layer (layer $l + 1$);
- *no response*, the position is unoccupied and robot r becomes the seed for layer l.

Such logic is implemented in Algorithm 5. The algorithm returns the position of the seed of the layer that is currently being assembled. In the case that the seed position is still empty, the robot moves to the seed location and it starts the seed algorithm. Notice that in the case that the robot becomes a seed, the execution does not return to the main program.

Robots at seed positions remain active, waiting for the message from new blocks to count how many robots are currently in that layer, acting as the *gatekeeper* of the layer.

Algorithm 5 Identifying layer

```
1:  function IDENTIFYING_LAYER( )
2:      $l \leftarrow 1$                                   ▷ The robot starts adjacent to $\boldsymbol{\rho}_{s_1}$
3:      while Robot ON do
4:          Send message to layer's seed, $\mathrm{m}_r^l$=circle layer?
5:          Start timeout timer
6:          Wait for message from seed, $\mathrm{m}_l^r$
7:          if Message received before timeout then
8:              if $\mathrm{m}_l^r = YES$ then
9:                  Return $\boldsymbol{\rho}_{s_l}$                  ▷ End this behaviour
10:             else                                         ▷ $\mathrm{m}_l^r = NO$
11:                 $MOVE.EXT(r, \boldsymbol{\rho}_{s_{l+1}})$
12:                 $l \leftarrow l + 1$, update current layer variable
13:             end if
14:         else                       ▷ No response, no block at seed position
15:             $MOVE(r, \boldsymbol{\rho}_{s_l})$
16:             Start Algorithm 6 on $r$                  ▷ Become seed robot
17:         end if
18:     end while
19: end function
```

To determine if a new block should circle the current layer or move forward, a seed must answer the following question: is the number of blocks in the current layer (circling or stopped) already enough to fill all desired positions, $|P_l|$? This logic is implemented in Algorithm 6. Every new robot that arrives in a layer communicates with the seed to inquire if it should circle or move to the next layer (line 8) using the messages described above (equation (6.1)). A seed uses this communication to compute (lines $9-11$) $n^l_{robots} = n^l_{robots} + 1$ where n^l_{robots} is the number of robots currently at layer l. If $n^l_{robots} = |P_l|$, a layer has already enough robots, moving or placed, to be fully assembled (check in line 9). Thus, when n^l_{robots} reaches its desired value, the seed starts rejecting new robots (line 13). There is a particular case when the robot is a bottom-layer seed (s_1). This robot is responsible for requesting new blocks to the structure by sending a request to the auctioneer in the swarm. To do that, the robot keeps track of n_{robots}, and requests a new block until the there are as many robots as position in the structure (lines $3-5$). The message sent to the auctioneer is the structure position, $m^a_{\mathsf{s}} = \boldsymbol{p}^{\mathsf{s}}$.

Fig. 6.2 illustrates the relevance of the seeds from a graph perspective. Recall that each layer form a grid graph G_l but layers are not connected

Algorithm 6 Seed program

```
1: Initialize counters $n^l_{robots} \leftarrow 1$, $n_{robots} \leftarrow 1$,
2: while TRUE do
3:     if $l = 1$ and $n_{robots} < |P|$ then                    ▷ Bottom-layer seed
4:         Send message $m^a_{\mathbf{s}}$ to the auctioneer,          ▷ Request a new block
5:         $n_{robots} \leftarrow n_{robots} + 1$       ▷ Increase counter of robots in structure
6:     end if
7:     Wait $\mathrm{m}^l_r$ message from a new block $r$.
8:     if $\mathrm{m}^l_r$ message received then
9:         if $n^l_{robots} < |P_l|$ then
10:            $\mathrm{m}^r_l \leftarrow YES$                        ▷ Should stay in this layer
11:            $n^l_{robots} \leftarrow n^l_{robots} + 1$                         ▷ Increase counter
12:        else
13:            $\mathrm{m}^r_l \leftarrow NO$                         ▷ Reject block in this layer
14:        end if
15:        Send $\mathrm{m}^r_l$ to $r$
16:    end if
17: end while
```

with each other. By acting as a gatekeeper and redirecting robots to upper layers, seeds create shortcuts taken by the blocks to reach the layer being assembled. This reduces the travelled distance for a robot when comparing to the alternative of having all robots exploring every layer from bottom to top until an empty position is found. Seed positions can be seen as *local hubs*, and how "connected" the seed is to the rest of the layer affects the number of movements robot perform when assembling a layer. Section 6.3 shows that the assembly process can be more effective by using the correct seed positions.

Algorithm 5 uses two movement functions:

- Function $MOVE(r, \boldsymbol{\rho})$ relocates the robot $r \in R$ at position $\boldsymbol{\rho}_r$ to an adjacent position $\boldsymbol{\rho}$ in the grid,

$$MOVE(r, \boldsymbol{\rho}) : \boldsymbol{\rho}_r \leftarrow \boldsymbol{\rho}. \tag{6.2}$$

- Function $MOVE.EXT(r, \boldsymbol{\rho}_{s_{l+1}})$ moves the robot $r \in R$ to a position adjacent to the seed at $\boldsymbol{\rho}_{s_{l+1}}$ by sequentially applying $MOVE(r, \boldsymbol{\rho})$ (moving one grid space at a time). Movement is executed in one axis at a time and the order to be followed is: *i*) move on the Z_B axis (change height), *ii*) move horizontally (X_B and Y_B axes) until an external and adjacent position to the seed has been achieved. Note that

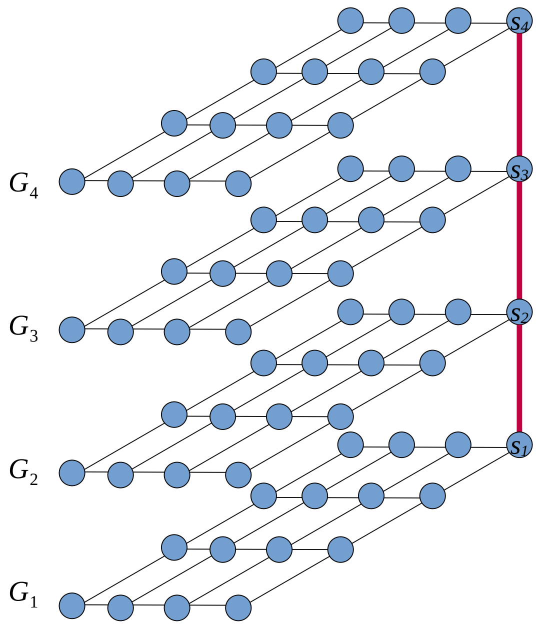


Figure 6.2: Graph view of a 4×4 cube with seed locations highlighted. Each layer G_l in the graph contains only the edges in *black*. *Red* edges between seeds represent the shortcuts taken during assembly, when a robot moves to upper layers.

$MOVE.EXT(r, \boldsymbol{\rho}_{s_{l+1}}) : \bar{P} \to \bar{P}$, i.e. only move through and always finish at an *external* position.

Fig. 6.3 shows a robot that just arrived at a layer that is completely assembled. In the figure, green rectangles show the empty positions in the structure, where a block should be placed. After communicating with the seed robot at $\boldsymbol{\rho}_{s_1}$, it decides to move adjacent to the second seed location, $\boldsymbol{\rho}_{s_2}$. Note that, after the robot moves adjacent to $\boldsymbol{\rho}_{s_2}$, it sends the ping message m_r^l, but it will not obtain any response. Therefore, following Algorithm 5, the robot assumes the seed position $\boldsymbol{\rho}_{s_2}$ and becomes a seed robot at layer $l = 2$. Recall that robots do not have information about the state of all positions in the structure, but only those adjacent to it at the moment. Thus, empty (green) and occupied (grey) positions are highlighted to help the reader visualise the partial status of the assembly, but that does not reflect the current information a robot has.

If robot r does not become a seed for a layer, after the layer that is currently being assembled is found ($\mathrm{m}_l^r = YES$), the execution returns to

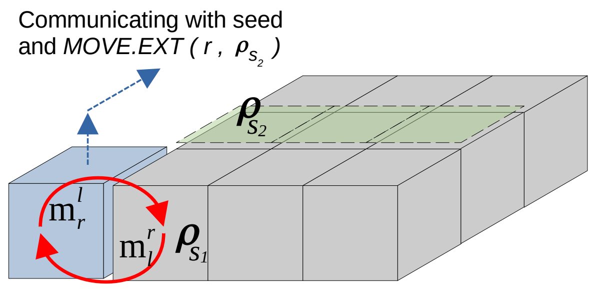


Figure 6.3: A newly arrived block inquires the seed if it should circle the current layer. Once the layer is already full, the seed denies it and the robot will apply $MOVE.EXT(r, \boldsymbol{\rho}_{s_2})$ to go to the next layer's seed.

Algorithm 4 and the main program starts the **second** behaviour by calling CIRCLING($\boldsymbol{\rho}_{s_l}$). The circling behaviour is used to guarantee that the execution of the assembly is deterministic: given a blueprint, circling will guarantee that parts are always assembled in the same order. Furthermore, note that the circling behaviour works in the same way if the structure is symmetric or asymmetric. However, when the structure is asymmetric, the choice of seed locations has a higher impact on how fast the structure is assembled.

Using the circling behaviour robot $r \in R$ moves around a block located at $\boldsymbol{\rho}_{s_l}$ by attempting to traverse the eight positions that encircles it. Using a generic location $\boldsymbol{\rho}_c$, let us define the cyclic order of $\boldsymbol{\rho}_c$ as the list containing these eight positions,

$$\begin{aligned} CO(\boldsymbol{\rho}_c) = (\quad & \boldsymbol{\rho}_c + [\ \ 0 - 1\ 0]^T, \quad \boldsymbol{\rho}_c + [-1 - 1\ 0]^T, \\ & \boldsymbol{\rho}_c + [-1 \quad 0\ 0]^T, \quad \boldsymbol{\rho}_c + [-1 \quad 1\ 0]^T, \\ & \boldsymbol{\rho}_c + [\ \ 0 \quad 1\ 0]^T, \quad \boldsymbol{\rho}_c + [\ \ 1 \quad 1\ 0]^T, \\ & \boldsymbol{\rho}_c + [\ \ 1 \quad 0\ 0]^T, \quad \boldsymbol{\rho}_c + [\ \ 1 - 1\ 0]^T\ ). \end{aligned} \tag{6.3}$$

Note that each pairwise adjacent elements in the cyclic order are one unit apart in the grid. Furthermore, for any two positions, $\boldsymbol{\rho}_i$ and $\boldsymbol{\rho}_j$, if $\boldsymbol{\rho}_i$ is in $CO(\boldsymbol{\rho}_j)$, then $\boldsymbol{\rho}_j$ must be in $CO(\boldsymbol{\rho}_i)$ and these positions are called *cyclically adjacent*. Also, note that a list such as the one in (6.3) can be created for other block formats, such as a hexagon. In that case, though, the list would be

$$\begin{aligned} CO(\boldsymbol{\rho}_c) = \quad (\quad & \boldsymbol{\rho}_c + [-1 \quad 0 \quad 0]^T, \boldsymbol{\rho}_c + [-\tfrac{1}{2} \quad \tfrac{\sqrt{3}}{2}\ 0]^T, \\ & \boldsymbol{\rho}_c + [\ \tfrac{1}{2} \quad \tfrac{\sqrt{3}}{2} \quad 0]^T, \boldsymbol{\rho}_c + [\ 1 \quad 0 \quad 0]^T, \\ & \boldsymbol{\rho}_c + [\ \tfrac{1}{2} \ - \tfrac{\sqrt{3}}{2} \quad 0]^T, \boldsymbol{\rho}_c + [-\tfrac{1}{2} - \tfrac{\sqrt{3}}{2}\ 0]^T). \end{aligned}$$

Considering the cubic case in (6.3), assume that r is circling a position $\boldsymbol{\rho}_c$ and is currently located at a position $\boldsymbol{\rho}_r \in CO(\boldsymbol{\rho}_c)$. To compute the position to move next (let us call it $\boldsymbol{\rho}_i$), r uses Algorithm 7.

Algorithm 7 Circling

```
function CIRCLING($\boldsymbol{\rho}_c$)                    ▷ $\boldsymbol{\rho}_c$ = position to be circled
    $CL \leftarrow START.LIST(\boldsymbol{\rho}_c, \boldsymbol{\rho}_r)$
    while Robot ON do
        $\boldsymbol{\rho}_i \leftarrow NEXT.POS(CL, \boldsymbol{\rho}_r)$
        if $\boldsymbol{\rho}_i \in P_l$ then                           ▷ Internal position
            Scan $\boldsymbol{\rho}_i$
            if $\boldsymbol{\rho}_i$ is occupied then
                $\boldsymbol{\rho}_c \leftarrow \boldsymbol{\rho}_i$, change circled block to $\boldsymbol{\rho}_i$
                $CL \leftarrow START.LIST(\boldsymbol{\rho}_c, \boldsymbol{\rho}_r)$
            else                                           ▷ $\boldsymbol{\rho}_i$ is empty
                $MOVE(r, \boldsymbol{\rho}_i)$
                Return $\boldsymbol{\rho}_i$
            end if
        else                                              ▷ External position
            $MOVE(r, \boldsymbol{\rho}_i)$
        end if
    end while
end function
```

The robot starts by creating a circular linked list [96], CL (lines 2, 9). The $START.LIST(\boldsymbol{\rho}_c, \boldsymbol{\rho}_r)$ function creates the list by, first, computing the positions to move, $CO(\boldsymbol{\rho}_c)$, using $\boldsymbol{\rho}_c$ and (6.3). Then, an empty circular list, CL, is created and the current robot position, $\boldsymbol{\rho}_r$, is added as the head of the list. Finally, starting from the robot position, $\boldsymbol{\rho}_r \in CO(\boldsymbol{\rho}_c)$, elements of $CO(\boldsymbol{\rho}_c)$ are sequentially added at the end of CL.

The robot proceeds by selecting the goal position, $\boldsymbol{\rho}_i$, by using the function $NEXT.POS(CL, \boldsymbol{\rho}_r)$ (line 4). It returns the next element in CL, linked to the current position of the robot $\boldsymbol{\rho}_r$.

Note that a robot must store in memory its actual position, $\boldsymbol{\rho}_r$, and the circular list, CL. With them, the robot can obtain the next position to move. In Algorithms 4 and 5, $\boldsymbol{\rho}_c$ is initialized as the seed position of the layer, $\boldsymbol{\rho}_{s_l}$. Also, as mentioned in Section 6.1, the robot computes $\boldsymbol{\rho}_r$ by counting its movement in the grid since the start of operation.

After robot $r \in R$ at $\boldsymbol{\rho}_r \in CO(\boldsymbol{\rho}_c)$ computes the goal position, $\boldsymbol{\rho}_i \in CO(\boldsymbol{\rho}_c)$, three possible cases unfold:

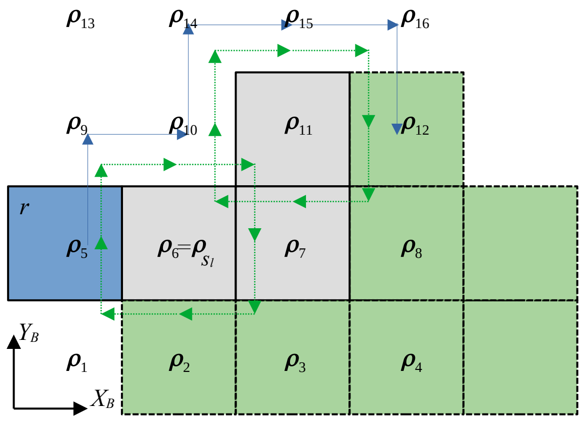


Figure 6.4: Top view: A robot (blue) circling the already placed blocks (grey) in the layer until it reaches an empty position (green). The green arrows describe $CO(\boldsymbol{\rho}_{s_l})$ and $CO(\boldsymbol{\rho}_{11})$. The position to be circled, $\boldsymbol{\rho}_c$, starts as $\boldsymbol{\rho}_{s_l}$ and changes to $\boldsymbol{\rho}_{11}$ as the robot reaches $\boldsymbol{\rho}_{10}$.

1. The goal position $\boldsymbol{\rho}_i$ is external ($\boldsymbol{\rho}_i \in \bar{P}$): r moves from $\boldsymbol{\rho}_r$ to $\boldsymbol{\rho}_i$ (lines $14 - 15$).
2. The goal position $\boldsymbol{\rho}_i$ is an internal position (line 5):
   a) The position $\boldsymbol{\rho}_i$ is occupied: then, robot r updates the position it is circling by setting $\boldsymbol{\rho}_c \leftarrow \boldsymbol{\rho}_i$ (lines $7 - 9$) and recomputes CL.
   b) The position $\boldsymbol{\rho}_i$ is unoccupied: robot r moves to $\boldsymbol{\rho}_i$, it ends the circling behaviour, the execution returns to Algorithm 4, and the robot starts the moving internal behaviour (lines $10 - 12$).

To better illustrate the behaviour, Fig. 6.4 shows a clockwise movement. The cyclic order of two positions, $\boldsymbol{\rho}_{s_l}$ and $\boldsymbol{\rho}_{11}$ are highlighted as green arrows. The robot's first position is adjacent to the seed position. Let us assume that in this case, the robot access the structure by position $\boldsymbol{\rho}_5$, i.e., $\boldsymbol{\rho}_r = \boldsymbol{\rho}_5$ and the robot is circling $\boldsymbol{\rho}_c = \boldsymbol{\rho}_{s_l}$. In this case, using (6.3), robot builds the list of positions to move and the circular list as,

$$\begin{aligned} CO(\boldsymbol{\rho}_{s_l}) &= (\boldsymbol{\rho}_3, \boldsymbol{\rho}_2, \boldsymbol{\rho}_1, \boldsymbol{\rho}_5, \boldsymbol{\rho}_9, \boldsymbol{\rho}_{10}, \boldsymbol{\rho}_{11}, \boldsymbol{\rho}_7) \\ CL &= ((\boldsymbol{\rho}_5)^{head}, \boldsymbol{\rho}_9, \boldsymbol{\rho}_{10}, \boldsymbol{\rho}_{11}, \boldsymbol{\rho}_7, \boldsymbol{\rho}_3, \boldsymbol{\rho}_2, \boldsymbol{\rho}_1). \end{aligned} \tag{6.4}$$

Note that the head of the circular list is set as $\boldsymbol{\rho}_5$. Following the list, the robot computes the goal position (line 4) as **second** element, i.e., $\boldsymbol{\rho}_9$.

The robot scans $\boldsymbol{\rho}_9$ and assess if it is empty. Since it is, it moves to that position. On the next iteration, $\boldsymbol{\rho}_r = \boldsymbol{\rho}_9$, and the goal position becomes the **third** element in the list CL, i.e, $\boldsymbol{\rho}_{10}$. Again, the robot scans $\boldsymbol{\rho}_{10}$ and moves to it because it is empty. At $\boldsymbol{\rho}_r = \boldsymbol{\rho}_{10}$, the robot will try to continue following CL by selecting the **fourth** element in the list, $\boldsymbol{\rho}_{11}$. But, as $\boldsymbol{\rho}_{11}$ is occupied, the robot cannot move into it. Thus, the block being circled ($\boldsymbol{\rho}_c$) changes from $\boldsymbol{\rho}_{s_l}$ to $\boldsymbol{\rho}_{11}$. From that point, the robot uses (6.3) and computes CL based on $CO(\boldsymbol{\rho}_{11})$ as

$$CL = ((\boldsymbol{\rho}_{10})^{head}, \boldsymbol{\rho}_{14}, \boldsymbol{\rho}_{15}, \boldsymbol{\rho}_{16}, \boldsymbol{\rho}_{12}, \boldsymbol{\rho}_{8}, \boldsymbol{\rho}_{7}, \boldsymbol{\rho}_{s_l}). \tag{6.5}$$

Since $\boldsymbol{\rho}_r = \boldsymbol{\rho}_{10}$, then $\boldsymbol{\rho}_{10}$ is set as the head of list. The goal position, following the list, is the **second** element, $\boldsymbol{\rho}_{14}$ and the robot moves through $\boldsymbol{\rho}_{14}$, $\boldsymbol{\rho}_{15}$, $\boldsymbol{\rho}_{16}$, $\boldsymbol{\rho}_{12}$. As $\boldsymbol{\rho}_{12}$ is an empty internal position, the robot ends its circling behaviour here.

As it will be shown in Section 6.2.2, it is not expected that a robot moves through all the positions in the list until it reaches the first position again. If that was the case, it would mean that the robot was not able to find an empty perimeter position to place itself and change its behaviour, which is not possible. Moreover, equation (6.3) was taken "left-to-right" to construct (6.4) and (6.5), and therefore such lists describe a clockwise movement. If (6.3) was taken "right-to-left" the robot would describe a counterclockwise.

However, if different robots were allowed to move clockwise and counterclockwise when assembling the same layer, a special case could happen where robots coming from both directions try to move to the same location at the same time. In that case, both robots (say r_1 at $\boldsymbol{\rho}_{r_1} = \boldsymbol{\rho} + [1\ 0\ 0]^T$ and r_2 at $\boldsymbol{\rho}_{r_2} = \boldsymbol{\rho} + [0\ 1\ 0]^T$) would have identified position $\boldsymbol{\rho}$ as empty at time t but would be unaware of each other. Thus, both robots would try to move to $\boldsymbol{\rho}$ at time $t+1$. Therefore for all robots, the list CL based on $CO(\boldsymbol{\rho}_c)$ is always constructed the same way, and only motion on the same direction is allowed, avoiding collisions.

Once a robot has circled the layer and found an unoccupied internal position, $\boldsymbol{\rho} \in P_l$, it moves to it and it is considered to be *inside* the structure. From *inside*, the circling behaviour ends and the execution returns to Algorithm 4. The robot then starts the **third** behaviour, implemented in Algorithm 8, which guides it to move as close as possible to the layer's seed, $\boldsymbol{\rho}_{s_l}$. From that point on, r only moves through other internal positions.

Let us consider that the robot is currently located at position $\boldsymbol{\rho}_r$. First, the robot identifies neighbouring positions in the same layer (l) that are part of the structure (line 3)

$$\begin{aligned} N(\boldsymbol{\rho}_r) = \{ \quad & \boldsymbol{\rho}_r + \begin{bmatrix}-1 & 0 & 0\end{bmatrix}^T, \boldsymbol{\rho}_r + \begin{bmatrix}1 & 0 & 0\end{bmatrix}^T, \\ & \boldsymbol{\rho}_r + \begin{bmatrix}0 & -1 & 0\end{bmatrix}^T, \boldsymbol{\rho}_r + \begin{bmatrix}0 & 1 & 0\end{bmatrix}^T \} \cap P_l. \end{aligned} \tag{6.6}$$

Algorithm 8 Moving internally

```
1: function MOVING_INTERNALLY($\boldsymbol{\rho}_r$)
       /*Robot currently located at $\boldsymbol{\rho}_r$*/
2:     while Robot ON do
3:         Compute internal adjacent positions, $N(\boldsymbol{\rho}_r)$
4:         Scan and remove occupied pos., $N(\boldsymbol{\rho}_r)^{empty}$
5:         Compute dist($\boldsymbol{\rho}_e, \boldsymbol{\rho}_{s_l}$), $\forall \boldsymbol{\rho}_e \in N(\boldsymbol{\rho}_r)^{empty}$
6:         if $\exists \boldsymbol{\rho}_e \in N(\boldsymbol{\rho}_r)^{empty}$ s.t. dist($\boldsymbol{\rho}_e, \boldsymbol{\rho}_{s_l}$) < dist($\boldsymbol{\rho}_r, \boldsymbol{\rho}_{s_l}$) then
7:             $MOVE(r, \boldsymbol{\rho}_e)$
8:         else
9:             Scan the position below $r$
10:            if Ground or an already placed robot then
11:                Return
12:            end if
13:        end if
14:    end while
15: end function
```

Second, the robot scans its neighbouring positions searching for empty locations, obtaining $N(\boldsymbol{\rho}_r)^{empty} \subseteq N(\boldsymbol{\rho}_r)$ (line 4). Finally, the robot moves to an unoccupied position $\boldsymbol{\rho}_e \in N(\boldsymbol{\rho}_r)^{empty}$ if and only if $\text{dist}(\boldsymbol{\rho}_e, \boldsymbol{\rho}_{s_l}) < \text{dist}(\boldsymbol{\rho}_r, \boldsymbol{\rho}_{s_l})$. That is, if the position $\boldsymbol{\rho}_e$ is closer to $\boldsymbol{\rho}_{s_l}$ (seed) than $\boldsymbol{\rho}_r$ (lines $6-7$).

If there is no other position to move to, a robot stops and shuts down if it is not a seed robot. The seed robot limits the number of robots that have "entered" a layer, but there is no control if these robots have already found a position and stopped. Thus, a layer l may start being assembled before layer $l-1$ has finished (all robots placed). In some special cases, it may occur that a robot r at layer l, $\boldsymbol{\rho}_r = [x_r\ y_r\ z_l]$, has stopped moving, but the position below it, $\boldsymbol{\rho}_i = [x_r\ y_r\ z_{l-1}]$, *i*) is still **empty**, because robots at $l-1$ have to take a longer path than those at layer l or *ii*) is **temporarily occupied**, because the robot at $\boldsymbol{\rho}_i$ at time t will continue moving and relocate at time $t+1$. The robot r scans the position below at least two consecutive times to guarantee that the position below is permanently occupied and the block has settled. This behaviour is shown in Fig. 6.5. The blue arrow shows the direction of r's motion so it reduces its distance from the seed ($\text{dist}(\boldsymbol{\rho}_3, \boldsymbol{\rho}_{s_l}) < \text{dist}(\boldsymbol{\rho}_1, \boldsymbol{\rho}_{s_l})$). After moving to $\boldsymbol{\rho}_3$, the robot will not find another empty position that is closer to $\boldsymbol{\rho}_{s_l}$ ($\boldsymbol{\rho}_4$ is occupied, therefore, not part of $N(\boldsymbol{\rho}_3)^{empty}$). Then the robot stops and shuts down. As before, in the figure, green are unoccupied

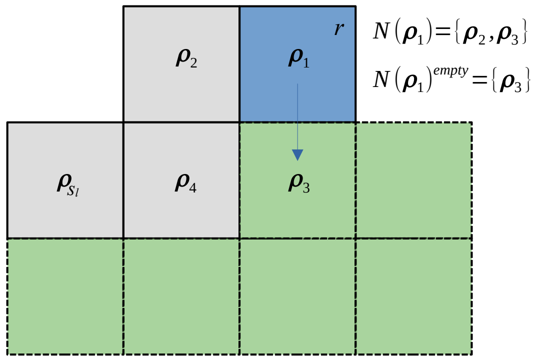

$\boldsymbol{\rho}_2$
$\boldsymbol{\rho}_1$
r
$N(\boldsymbol{\rho}_1)=\{\boldsymbol{\rho}_2,\boldsymbol{\rho}_3\}$
$N(\boldsymbol{\rho}_1)^{empty}=\{\boldsymbol{\rho}_3\}$
$\boldsymbol{\rho}_{S_l}$
$\boldsymbol{\rho}_4$
$\boldsymbol{\rho}_3$

Figure 6.5: A robot that is at an internal position evaluates neighbouring positions $N(\boldsymbol{\rho}_r)^{empty}$ to determine if and where to move.

positions, grey are occupied positions, and the robot is the blue square.

Remark 7. *Note that, under the properties defined in Sections 3.5 and 6.1, the behaviours described in the algorithms in this section are **deterministic** for any arbitrary initial condition. Therefore, the construction of a specific structure will always take the same number of steps and will always result in the same sequence of movements. As a result, there is no stochasticity in the assembly of structures.*

6.2.2 Self assembly algorithm proofs

In this section, the graph G introduced in Section 3.5 is used to prove that the proposed algorithms lead to the assembly of the structures in the blueprints. Lemma 3 defines that every layer receives sufficient number of robots to occupy all empty positions. In Lemma 4, all empty perimeter vertices are reached and occupied by robots. Lemma 5 defines that a layer must first be assembled for all its robots to finish their algorithms. Finally, Theorem 2 states that all robots indeed complete their program assembling the structure.

Lemma 3. *Every layer $1 \leq l \leq L$ receives $|V_l|$ robots to build the layer.*

Proof. By following Algorithm 4 and 5, the first robot that enters any layer l positions itself onto the seed vertex s_l. Each of the following $|V_l| - 1$ robot coming to layer l are assigned to circle the layer, after which any remaining robot is directed to move to layer $l+1$. Due to Property 8, there are sufficient number of robots to occupy each vertex in the structure. Thus each layer $1 \leq l \leq L$ receives a total of $|V_l|$ robots to build the layer. $\square$

Lemma 4. *If a layer has an empty perimeter vertex and a circling robot, then following Algorithm 7 a circling robot shall enter an empty perimeter vertex in that layer.*

Proof. Recall that Property 1 states that each layer of the structure can be represented by a connected graph. Let us first assume that layer l is completely empty. Therefore, based on Algorithms 4 and 5, an arbitrary robot r will occupy the seed vertex s_l, which is a perimeter vertex. What is left to demonstrate is that this is true for any other unoccupied perimeter vertex $v \in V_l$.

Let us assume that an arbitrary robot r, following Algorithm 7, is circling a layer l. For ease of exposition, let us assume that all perimeter vertices are occupied. As a consequence of Algorithms 4 and 5, robot r starts from an external position that is adjacent to s_l. Let us call this external vertex $\bar{v}_r^1 \in \bar{V}_l$ meaning that it is the vertex occupied by robot r at time step 1. Then, using Algorithm 7, r defines its current circling vertex $v_c \leftarrow s_l$ and moves clockwise around v_c until it reaches the next perimeter vertex (say v_i) that is cyclically adjacent to v_c. Since v_i is occupied by assumption, r updates its current circled vertex $v_c \leftarrow v_i$. Thus the state of r can be expressed with a tuple $(v_c, \bar{v}_r^t)$ where v_c is the current perimeter vertex that r is circling clockwise, and $\bar{v}_r^t \in \bar{V}_l$ is the vertex where r is located at time step t. Let the sequence of tuples describing r's states be $\mathcal{S}_r = ((v_{c_1} = s_l, \bar{v}_r^1), (v_{c_2}, \bar{v}_r^2), (v_{c_3}, \bar{v}_r^3), \ldots, (v_{c_k}, \bar{v}_r^k))$. Since there are only finitely many perimeter vertices, at some point the sequence shall repeat itself so k is selected such that $(v_{c_{k+1}}, \bar{v}_r^{k+1}) = (v_{c_1}, \bar{v}_r^1)$.

Notice that the sequence of external vertices r traverses form a cycle $\bar{C} = (\bar{v}_r^1, \bar{v}_r^2, \ldots, \bar{v}_r^k)$. Since s_l is necessarily a perimeter vertex (Property 3), it is in the interior of $\bar{C}$. Also, since $\bar{C} \subset \bar{V}_l$, due to Property 1 and the Jordan-curve theorem [97], no interior vertices can be on the outside of this cycle. Finally, due to Property 2 and since r circles clockwise coming adjacent to each perimeter vertex cyclically adjacent to the previous one, every perimeter vertex must occur in $\mathcal{S}_r$. Thus, as long as there is an empty perimeter vertex, Algorithm 4 finds the first empty perimeter vertex in the sequence $\mathcal{S}_r$ and takes its position. $\square$

Lemma 5. *When the robots' algorithms terminate, for any layer $1 \leq l \leq L$, all internal vertices of the grid graph G_l get occupied by robots.*

Proof. Lemma 3 shows that for each layer, the required number of robots enter the layer to search for empty positions. Two robots cannot occupy the same vertex simultaneously. If there is an empty perimeter vertex, there must be a robot circling the layer, and, due to Lemma 4, this empty perimeter vertex will eventually be occupied by a robot.

Assume that the algorithm terminates without completing the building of a layer of the structure. Since it was already demonstrated that perimeter vertices get occupied, all the empty vertices must be non-perimeter vertices. Let us assume v_i is such an empty non-perimeter vertex whose distance to the seed vertex is the *greatest*. Let v_j be a vertex among v_i's four neighbours such that $dist(f(v_j), f(s_l)) > dist(f(v_i), f(s_l))$. Then, v_j must be either (i) a non-perimeter vertex which as per assumption is already occupied, or (ii) a perimeter vertex and hence due to Lemma 3 and Lemma 4 must get occupied by a robot. In either case this robot r in v_j would have moved to v_i, making the current state of Algorithm 4 not the terminating state. □

Theorem 2. *Algorithm 4 terminates forming the structure specified in the blueprint B.*

Proof. At the start of the algorithm, the behaviour state of all robots is set to 'find layer'. Then every arbitrary robot r sequentially visits the layers from bottom to top, until it finds a layer with more internal vertices than there are robots in it. Thus due to Property 8, r finds such a layer and changes to the 'circling' behaviour.

Since there can be at most as many robots in a layer as there are empty internal vertices in the layer, due to Lemma 4, after a sequence of circling moves, r occupies a perimeter vertex and changes its behaviour to 'internal'. Whenever a robot r with behaviour state 'internal' moves, its distance to the seed vertex reduces by one. Since this value cannot reduce indefinitely, after a finite number of steps r reaches its optimal position. Due to Lemma 5, the robots occupy every internal position of every layer after a finite number of steps. Thus due to Property 4, every robot must have ground or another robot immediately below it and hence all robots reach the shut down state terminating the algorithm. Since every internal position in the blueprint is now occupied and no robot is in an external position, the structure has been built. □

6.3 Results

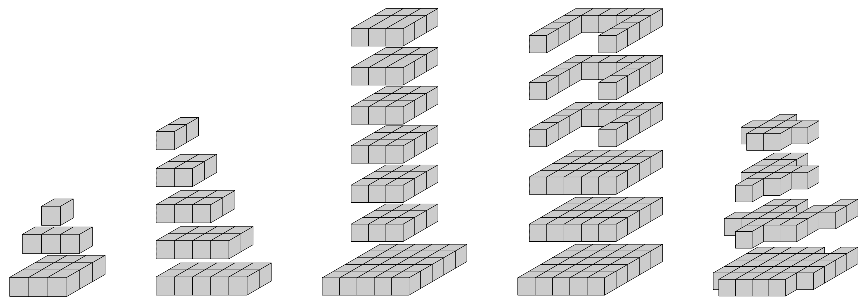
In this section, the effectiveness of the proposed approach is analyzed by looking at distance travelled by the robots during assembly, comparing it to the worst case (when robots take the longest path from start to stop) and the best case (shortest path). Furthermore, by analysing the layer's graph, a relationship can be found between the distance travelled by the robots and the position of the seed in the layer.

The remainder of the section is organized as follows. Section 6.3.1 describes the simulation set up and the structures used in simulation. Section 6.3.2 shows the parameters used to analyse the algorithms' efficiency. In Section 6.3.3 the proposed approach performance is evaluated. In Section 6.3.4, graph-based metrics are proposed to quantify the impact of seed selection in the assembly performance. Finally, Section 6.3.7 shows the application of the proposed approach in real-world.

6.3.1 Simulation setup

The proposed solution was tested in structures with different sizes and shapes, as shown in Fig. 6.6. The structures varied from 13 to 98 blocks, with up to 7 layers. Notice that, one restriction imposed by Property 4 is that every block in the structure must be either supported by another block below, or on the ground. That reduces the number of possible structures that can be directly assembled using the proposed algorithms. Note though that a large number of structures of interest abide to this restriction. Furthermore, according to Property 1 and Property 2, structures need to be connected, i.e., no holes are allowed in a layer. However, many complex structures can be broken down in smaller and simpler ones [20] and several executions of the proposed approach (in parallel or sequentially) can be performed to build more complex structures. For example, walls and pillars of a house could be built separately and form the final structure after all have been assembled.

One implicit parameter is the time interval Δt at which robots arrive adjacent to $\boldsymbol{\rho}_{s_1}$. Smaller this interval, bigger the number of robots moving simultaneously during assembly. Recall that new robots arrive as per demand from bottom-layer seed (Algorithm 6, line 4). After requested, a robot from the swarm in the waiting state is assigned by auctioneer and sent to the structure (see Section 5.2). Thus, the time needed for the auction process and movement of the selected robot impacts Δt directly. There is, however, a minimum interval at which robots could arrive being computed as the time for: *i*) a recently arrived robot to communicate with the seed at $\boldsymbol{\rho}_{s_1}$ (2 time



(a) pyramid (b) stairs (c) pole (d) chair (e) asymmetrical

Figure 6.6: Exploded view of the structures assembled highlighting each layer.

steps); and *ii*) move 1 grid space away from the arrival position (1 time step). Note that short arrival intervals can result in other issues during assembly, e.g., having robots in upper layers waiting for robots at layers below to stop moving. On the other hand, the assembly algorithms are not affected by changes on Δt, e.g., the interval between the arrivals of the third and fourth robots could be longer than the interval between the fourth and the fifth robots. In the simulations performed, Δt was arbitrarily defined as 5 time steps always.

6.3.2 Guidelines for performance analysis

The distance travelled by the robots during assembly was used as a metric to evaluate the performance of the proposed approach. In simulation, the distance travelled is equal to the number of times the function $MOVE(r, \boldsymbol{\rho})$ is executed by each robot. As mentioned in Section 6.2, this function results in the robot moving 1 grid space per call.

The proposed approach was compared with both the longest and the shortest path a block could take from the position where it starts to its final destination:

- Longest path (**LP**) - if there is no communication between robots, robots starting adjacent to $\boldsymbol{\rho}_{s_1}$ would have to circle every layer of the structure to identify if it was already assembled. The circling is performed sequentially, layer by layer, until an empty position is found.
- Shortest path (**SP**) - if there were no other blocks placed in the structure, the shortest distance a robot would need to navigate to reach its final position is simply the Manhattan distance from origin to destination.

This path is computed using the robot's start location (adjacent to $\boldsymbol{\rho}_{s_1}$) and its final location ($\boldsymbol{\rho}_{r_f}$), **SP**= $\text{dist}(\boldsymbol{\rho}_{r_f}, \boldsymbol{\rho}_{s_1}) + 1$. Notice that **SP** assumes an unrealistic path, for a robot cannot move through other blocks in the assembly to reach its goal position. However, this metric is still useful because it defines a lower bound for the movements a robot has to perform before placing itself.

6.3.3 Simulation trials

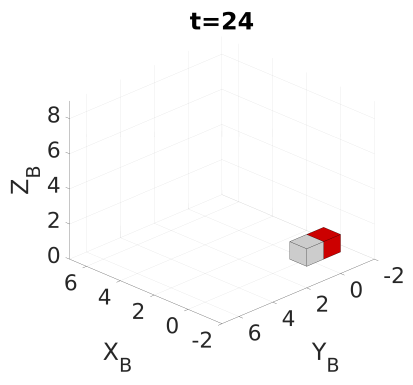
Fig. 6.7 shows the multiple stages of the self-assembly of the chair, 6.6d. In the figure, *red* robots are recently arrived robots, that are communicating with the seed at $\boldsymbol{\rho}_{s_1}$. Also, the *magenta* robots are currently circling the structure. Finally, *grey* colored robots are already positioned blocks that have shutdown. The seed positions were arbitrarily selected as $\boldsymbol{\rho}_{s_l} = \begin{bmatrix} 0 & 0 & l \end{bmatrix}^T$ (seeds form a column on top of the origin of O_B). The assembly demonstrates the proposed self-assembly algorithms working for layers with different sizes and shapes.

In the simulation trials, the number of movements of each robot $r \in R$ participating in the assembly, m_r, was measured. Table 6.1 shows the average number of movements for all robots $\mu_m = (\sum m_r)/|R|$, the standard deviation σ_m, and the maximum number of movements $max(m_r)$ by a robot in the assembly. Note that the robots' motions are always of 1 unit, thus, this measure corresponds to the distance traveled by the robots. To select seed positions, first $\boldsymbol{\rho}_{s_1}$ is arbitrarily defined as one of the perimeter vertices in the bottom layer. Then, for the subsequent layers, $\boldsymbol{\rho}_{s_{l+1}}$ in layer $l+1$ is selected as the closest position to $\boldsymbol{\rho}_{s_l}$ in layer l.

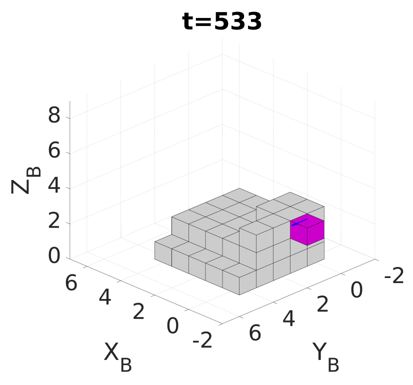
Fig. 6.8 shows the distance traveled by each robot to assemble the chair (Fig. 6.6d). It shows that by taking **LP**, upper layer robots have to move more to reach their final destination. Note that, with the proposed approach, the travelled distance does not grow as fast as the **LP** approach. In fact, for the second layer and above, the proposed approach is similar to the shortest possible distance that the robots can travel, **SP**. The good performance of the proposed approach is due to the fact that seeds create shortcuts in the assembly and help robots to avoid unnecessary movement.

6.3.4 Seed selection and assembly performance

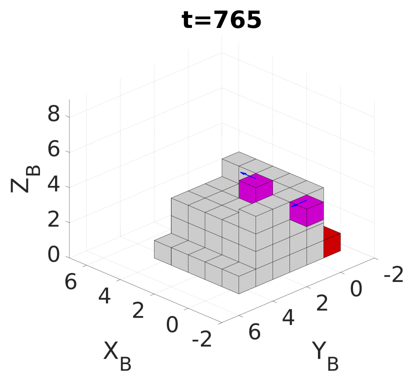
In this section, it is explored if the performance of the proposed assembly strategy is influenced by the location of seed positions in the structure. Recall that robots change from layer l to layer $l+1$ by moving from a position adjacent to $\boldsymbol{\rho}_{s_l}$ to one adjacent to $\boldsymbol{\rho}_{s_{l+1}}$. In that sense, the positions of seeds define the



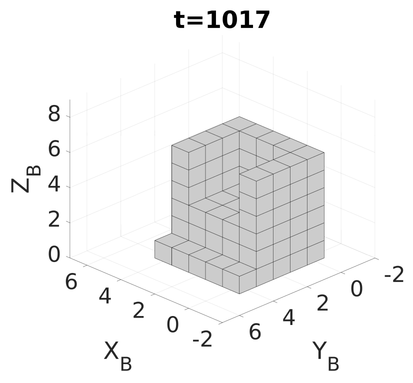
(a)



(b)



(c)



(d)

Figure 6.7: The assembly of the chair with robots moving clockwise.

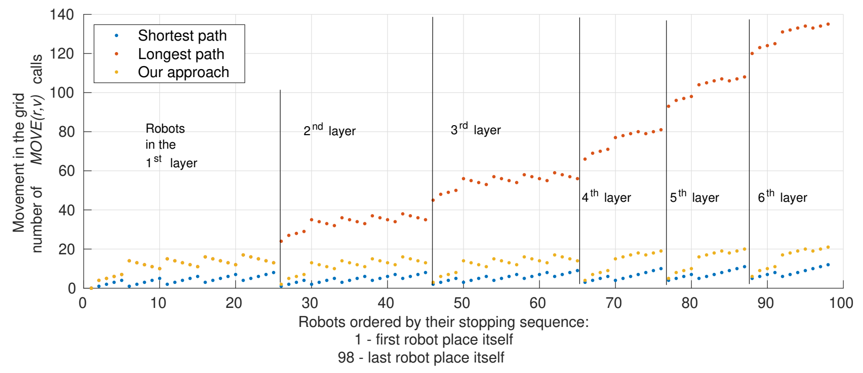


Figure 6.8: The distance traveled by each robot assembling the chair.

Table 6.1: Movements performed by robots during assembly

Structures		**SP**	**LP**	Proposed approach
pyramid	μ_m	2.4	11.7	6.6
	σ_m	1.3	7.7	3.5
	$max(m_r)$	4	27	11
stairs	μ_m	3.2	29.1	7.1
	σ_m	1.4	20.1	3.1
	$max(m_r)$	5	68	12
pole	μ_m	6.4	55.5	11.6
	σ_m	2.6	39.1	3.7
	$max(m_r)$	12	122	18
chair	μ_m	5.4	55.4	12.7
	σ_m	2.4	39.1	4.5
	$max(m_r)$	12	135	21
asymmetrical	μ_m	5.5	39.0	12
	σ_m	2.0	29.21	5.6
	$max(m_r)$	10	97	23

path the robots take when moving between layers. Using the structure graph, it was evaluated if seed positions could be selected such that the distance travelled by the robots is minimized.

As defined in Section 6.1, graphs representing different layers, G_l, were not connected between them. Given that seed positions define a path to be taken, the structure graph, $G[V]$, is constructed by connecting the seed of layer s_l to the layers above and below, s_{l+1}, s_{l-1} (see Fig. 6.2).

Two metrics were used to analyse the structure graph:

- Closeness - this is a graph centrality metric that outputs the average distance between a given vertex and all other vertices. Closeness is computed as

$$c(v_i) = \frac{1}{\sum\limits_{j=1,j\neq i}^{N} d_{sp}(v_i, v_j)} \tag{6.7}$$

where the distance $d_{sp}(v_i, v_j)$ is the cost of traversing the graph using the shortest path between vertices v_i and v_j. In the graph, all edges in the same layer have cost equal to 1. Thus, $d_{sp}(v_i, v_i) = \text{dist}(\boldsymbol{\rho}_i, \boldsymbol{\rho}_j)$ and dist was defined in equation 3.7. Also, the cost for edges connecting two seeds, say s_l and s_{l+1}, is $\text{dist}(\boldsymbol{\rho}_{s_l}, \boldsymbol{\rho}_{s_{l+1}})$. Therefore, for vertices on different layers $v_i \in G_i$ and $v_j \in G_j$, $1 \leq i \leq j \leq L$,

$$d_{sp}(v_i, v_j) = \text{dist}(\boldsymbol{\rho}_i, \boldsymbol{\rho}_{s_i}) + \sum_{k=i}^{j-1} \text{dist}(\boldsymbol{\rho}_{s_k}, \boldsymbol{\rho}_{s_{k+1}}) + \text{dist}(\boldsymbol{\rho}_{s_{j+1}}, \boldsymbol{\rho}_j) \quad (6.8)$$

- The Network Controllability Gramian (NCG) - this metric is used to analyse the control effort to change the state of all nodes in a network. Also, the trace of the NCG, $trace(W_j)$, is inversely related to the average energy to control the network [81]. The hypothesis suggested is that such energy-related metric can provide some understanding on how the selection of different seed positions impact the structure graph, and therefore, the assembly. To compute the NCG, first the structure's adjacency matrix, $\mathbf{A} = [a_{ij}]$, is computed. For vertices connected by an edge and within the same layer, v_i and v_j, $a_{ij} = a_{ji} = 1$. Vertices that are not connected yield $a_{ij} = a_{ji} = 0$. The connection between seeds is represented as $a_{s_l,s_{l+1}} = a_{s_{l+1},s_l} = 1/dist(\boldsymbol{\rho}_{s_l}, \boldsymbol{\rho}_{s_{l+1}})$. Given the adjacency matrix, $\mathbf{A}$, one may compute $\mathbf{W}_j$ (the NCG) by following equations (4.12)-(4.14) (see Section 4.2.2). When computing $\mathbf{W}_j$, commonly $3 < \alpha < 10$ and $100 < H < 1000$ for the structures analysed. Finally, nodes in the network are sequentially selected (selecting each vertex as input) to obtain an average controllability value, τ, for the topology,

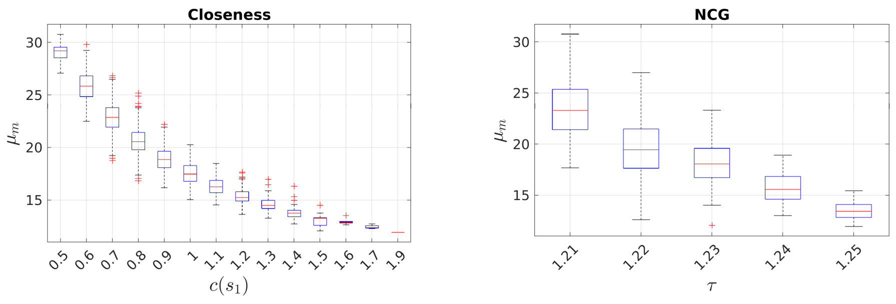
$$\tau = \frac{1}{N} \sum_{j=1}^{N} trace(\mathbf{W}_j). \quad (6.9)$$

The assembly of two structures was analysed using the metrics proposed, the *chair* in Fig. 6.6d and the *asymmetrical structure* in Fig. 6.6e.

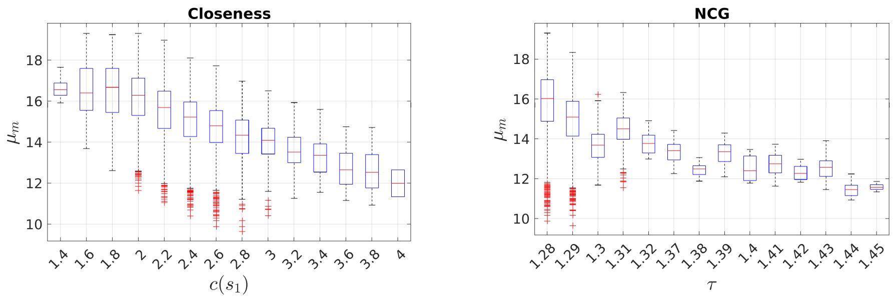
The list of seed locations is denoted by $S = \{\boldsymbol{\rho}_{s_1}, \ldots, \boldsymbol{\rho}_{s_L}\}$. The *chair* has a total of six layers ($L = 6$) and the four corners of each layer were considered as possible seed locations for a total of $4^6 = 4096$ combinations. Note that every combination forms a set S_β, $1 \leq \beta \leq 4096$. The asymmetrical structure has four layers ($L = 4$), and all perimeter vertices were considered as possible seeds for a total of $16 \times 15 \times 10 \times 6 = 14,400$ combinations and $1 \leq \beta \leq 14,400$.

For every combination, S_β, the assembly was executed twice: first with all robots moving in the clockwise direction, and second with all robots moving counterclockwise. Therefore, for each S_β, it is computed $\mu_m = (\mu_{cw} + \mu_{ccw})/2$ to account for assemblies on both directions.

Fig. 6.9 shows the correlation between the graph's topology (given S_β) and the efficiency of the proposed strategy. The two metrics, average controllability, τ, and closeness of the bottom-most seed vertex, $c(s_1)$ are compared with



(a) Chair, Fig. 6.6d



(b) Asymmetrical, Fig. 6.6e

Figure 6.9: The relationship between the closeness and the NCG metrics with the average distance travelled by robots when executing the proposed assembly strategy.

the average travelled distance by the robots, μ_m. The figures show that the higher the $c(s_1)$, the shorter the average distance to be traveled by robots in the assembly. Also the higher the average controllability (lower control energy required for a given topology), the lower the number of movements performed by robots.

In the figure, it is also possible to notice that there is a correlation between the regularity of the curves and symmetricity of the structure. The chair, which has highly symmetrical layers, presented a curve that roughly followed a decreasing exponential, whereas in the asymmetrical structure, such curve seems noisier (bigger standard deviation) and roughly linear. However, the investigation of the correlation between structure shapes and the metrics' curves is a topic of future research. Finally, before starting an assembly, a designer can calculate the closeness or NCG to choose the best locations for the seeds. This has the potential to reduce the assembly process to manageable time scales.

6.3.5 Assembly of more complex structures

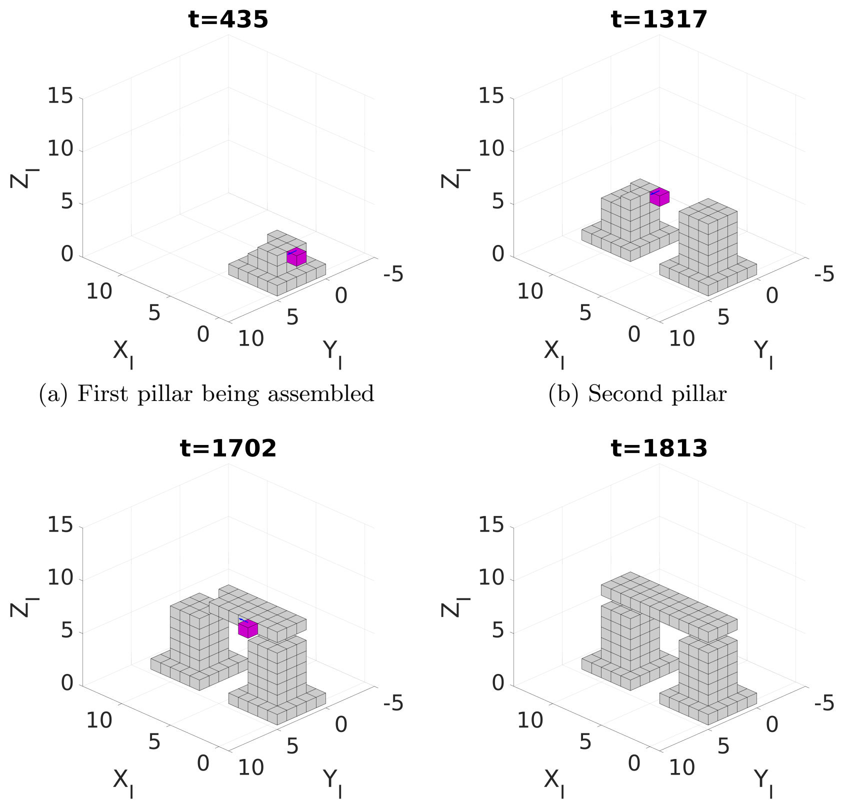
The set of possible structures that can be constructed by a single run of the proposed self-assembly strategy is limited to structures where a block is either on the ground or on top of another (properties in Section 6.1). However, given the decentralized nature of the system, more complex structures can be assembled by executing multiple runs of the approach presented.

Figure 6.10 shows an example where a bridge is assembled by executing the self-assembly solution more than once. One execution of the algorithms is necessary to assemble each pillar, and another for the top. Each pillar of the bridge is assembled by a different set of robots. The robots carry the same blueprint, but the position of the bottom seed, s_1, is different. Recall that the blueprint carries relative positions for the structure, with respect to s_1. Therefore the location of s_1 in space determines the location of the whole structure. The two pillars are assembled sequentially in the simulation, followed by the top part, for ease of presentation. However, note that as a result of auction-based assignment process (discussed in Chapter 5), the structures could be assembled simultaneously. First, the bottom-layer seed robot of each structure requests new blocks as needed (see Algorithm 6). Second, the auction process selects robots from the swarm to send to each structure.

To assemble the top of the bridge, Property 4 is relaxed to allow robots to stop on top of others or at a position on the first level (s_1 level). The top part is assembled one level higher than the other blocks, with robots docking in midair. After assembled, the top part can be lowered, landing on top of the pillars. Note that this assumes that the robots are capable of docking in midair and perform cooperative flight, similar to [42]. To enable cooperative flight, Property 9 could be relaxed to allow communication and for the robots to remain in operation (instead of shutting down). The study of connected flight if left as future work.

6.3.6 Validation using embedded sensors for navigation

A simulation was developed to demonstrate that a robot with only embedded sensors is capable to perform the self-assembly algorithms proposed. On the robotics simulator V-Rep 3.5, each robot was embedded with an on-board camera facing down, a range sensor to detect height, and an inertial measurement unit (IMU) for attitude control. Camera data were sent through the Robot Operating System (ROS) to an auxiliary Python script (a ROS node), which processed the image detecting other robots in space and estimating the



(a) First pillar being assembled (b) Second pillar

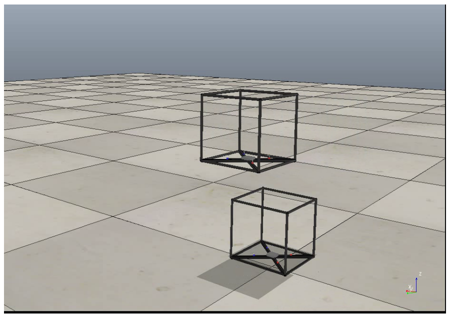
(c) Assembly of the top (d) End of execution

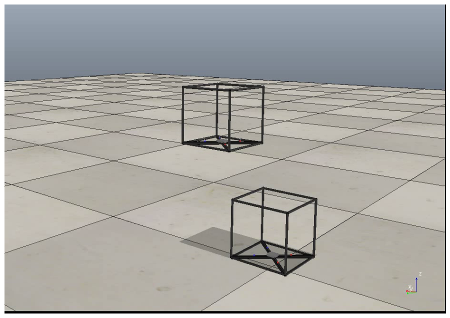
Figure 6.10: The assembly of a bridge through multiple executions of the proposed self-assembly approach.

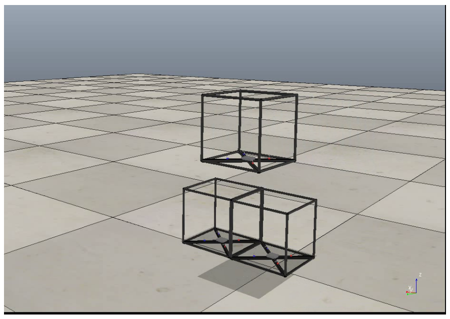
robot's motion. To approximate the simulation with real-world results, estimated noise values for the micro UAV Bitcraze Crazyflie [98] were added to sensors' outputs. Note that, the same UAV is used in the experiments (Section 6.3.7).

Figure 6.11 and Figure 6.12 show the simulation where two robots sequentially arrive adjacent to the s_1 seed robot, already landed. Each moving robot's camera is used as an object detection sensor, showing if a neighbouring position is empty or not. Robots start adjacent to the seed (already landed) and compute their velocities, $\dot{p}_r$, using the Lucas-Kanade optical flow technique [99]. A video of a typical run of the simulation can be found at `https://youtu.be/gZYQwoK7EZs`.

The simulation shows that it is possible for a robot to navigate and ex-


(a) A robot executing the circling behaviour (Algorithm 7)


(b) Robot moving internally (Algorithm 8)


(c) Second robot moving from adjacent to the seed (Algorithm 7)

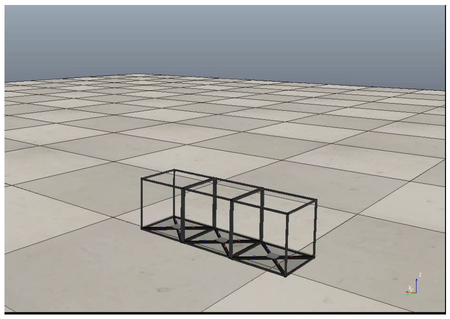
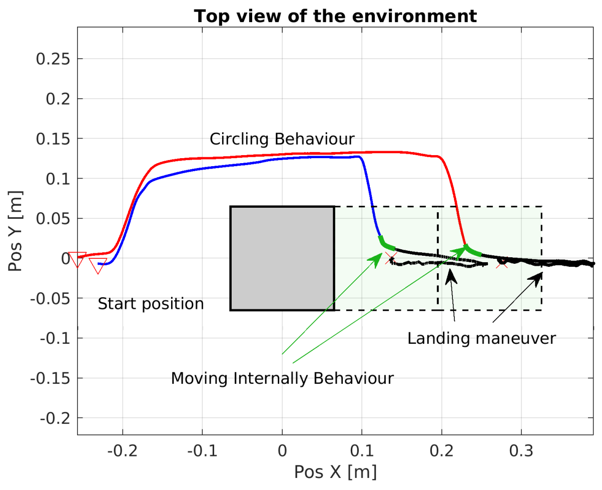
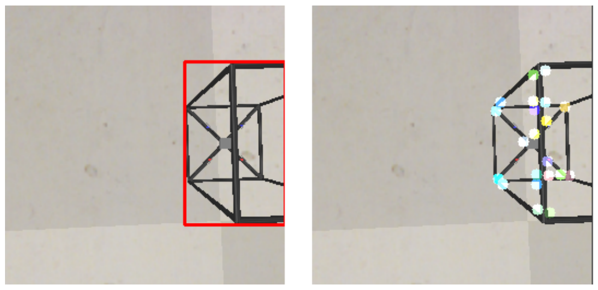

(d) All robots landed

Figure 6.11: Robots performing navigation and self-assembly using embedded sensors.

ecute the proposed self-assembly approach relying only on embedded sensors. Robots were able to correctly compute their location in the structure, find empty positions and land, executing all behaviours in the proposed self-assembly solution. For ease of demonstration, Figure 6.11 shows the assembly of a single-layered structure, however, as shown in the sections above, the self-assembly approach can be extended to structures with more layers. Figure 6.12a shows the trajectory described by each one of the robots and the behaviour that was being executed at each part. Moreover, Figure 6.12b shows the output of the object detection and the optical flow algorithms, which are used to detect other robots in space and compute the vehicles' velocity.



(a) Top view of the Trajectory



(b) Output of camera processing algorithms. Left: object detection, right: optical flow

Figure 6.12: Robots performing navigation and self-assembly using embedded sensors.

6.3.7 Experimental results

The proposed algorithms were implemented in real-world robots: small quadrotors placed inside carbon-fibre light weight cubic frames (Fig. 3.3) that act as the structure parts. The length of each side of the block is 12 cm and it weighs less than 50 g. The use of multicopters enable the robot to perform three-dimensional movement, respecting Property 5.

Position, velocity, and orientation of the vehicles are captured by a VICON camera system and sent to each robot. Such information is then used by the embedded controllers during flight. Notice that the proposed self-assembly solution is applicable to any robot with local sensing information, as shown in Section 6.3.6. Therefore, using the camera system (a centralized sensor approach) to capture the states of robots is a limitation of the robotic platform used and it does not invalidate Property 10 of the proposed self-assembly solution. Moreover, data obtained from the camera system is used to simulate the embedded sensors (status of neighbouring positions), explained in Section 6.1.

To avoid possible collisions and air turbulence, real robots fly approximately 36 cm (length of three blocks) higher than the computed position in the self-assembly algorithms. Such displacement is considered only at the robots' flight controllers and do not affect the self-assembly behaviours. To the self-assembly algorithms, the robots move side-by-side of other blocks and in a grid-like environment, as done in simulation.

Fig. 6.13 shows the assembly process of the stairs with three levels and six robots ($L = 3,\ |R| = 6$) and Fig. 6.14 shows stairs with three levels and twelve robots ($L = 3,\ |R| = 12$). The figures shows photos of the experiment whilst videos of the assembly can be found at: *a)* 6 blocks, `https://youtu.be/WJ4kyx_MOxs` and *b)* 12 blocks, `https://youtu.be/l1Q2W28R6DU`. The assembly starts with a robot located at $\boldsymbol{\rho}_{s_1}$. Other vehicles are sequentially added to the process. After a new robot is started, first it moves adjacent to $\boldsymbol{\rho}_{s_1}$, and then it starts the self-assembly behaviours.

The closeness metric proposed previously was used to define the seeds of each layer, S_β. For both structures, every block is a perimeter vertex. Thus, a total of 6 combinations where evaluated for the 6-block stairs and 48 combinations for 12-block stairs. The combination selected was $S_\beta = \{ \boldsymbol{\rho}_{s_1} = [0\ 0\ 0]^T\ ,\ \boldsymbol{\rho}_{s_2} = [0\ 0\ 1]^T\ ,\ \boldsymbol{\rho}_{s_3} = [0\ 0\ 2]^T\}$, given that it returned the highest metric values, $c(s_1) = 0.125$ for the 6-block stairs and $c(s_1) = 0.045$ for the 12-block stairs. Finally, a video showing in detail the docking procedure, with robots approaching and attaching to each other can be found at `https://youtu.be/-uk45pwKqig`.

As showed, the self-assembly strategy proposed led to structure assembly under real-world conditions. Moreover, the structures tested did not present any issues related to weight and balance distribution. This provides a reasonable indication of the proposed approach's applicability in realistic scenarios.

6.4 Conclusion

In this chapter, a novel approach for self-assembly of structures was proposed. The blueprint-based assembly strategy relies on local information and it is shown to work on three-dimensional environments. The proposed approach follows a series of local behaviours that regulate interactions between robots.

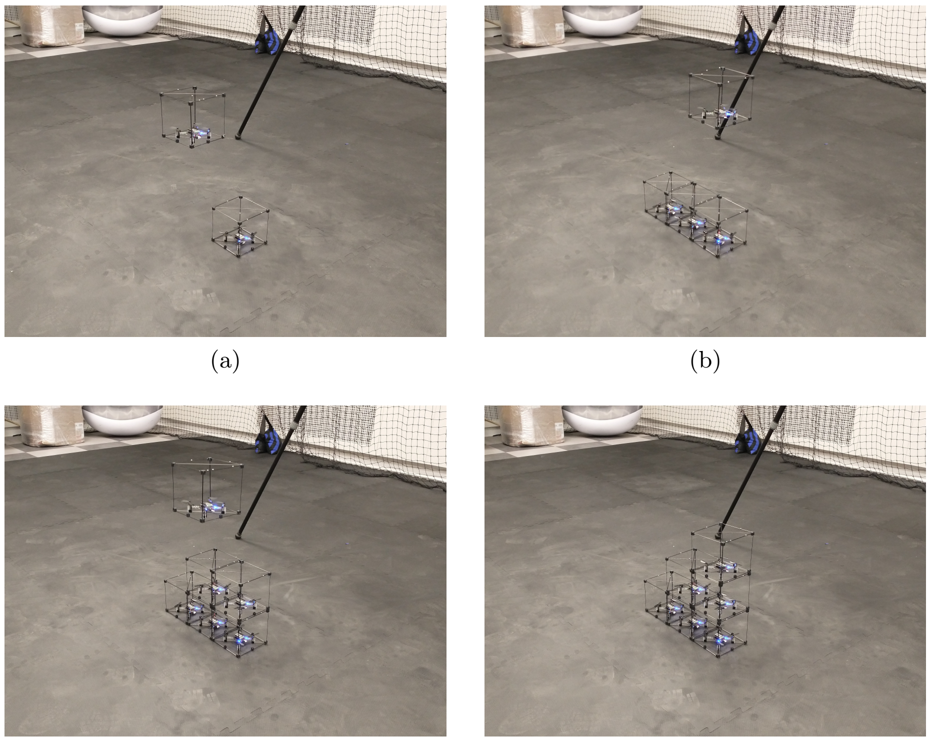

(a) (b) (c) (d)

Figure 6.13: Assembly of stairs with six robots.

The behaviours proposed guide the robots' search for an empty position in the structure. The self-assembly approach finishes when all robots have found a location to stop and the structure has been assembled.

In Section 6.3, it is shown that by executing the proposed algorithms, the robots efficiently place themselves, taking a path similar to the shortest possible path. Also, using network analysis tools, it was demonstrated that the effort to build a structure (distance travelled by all robots) is related to how connected the structure graph topology is. Therefore, a careful selection of seed positions can improve the self-assembly efficiency and possibly decrease assembly time. A future topic of investigation is the impact of structure asymmetry in the metrics chosen to select the seeds.

Finally, it is shown that the proposed approach is applicable to real life construction, using real robots. The construction process was correctly performed for two structures with different number of robots. The robots used

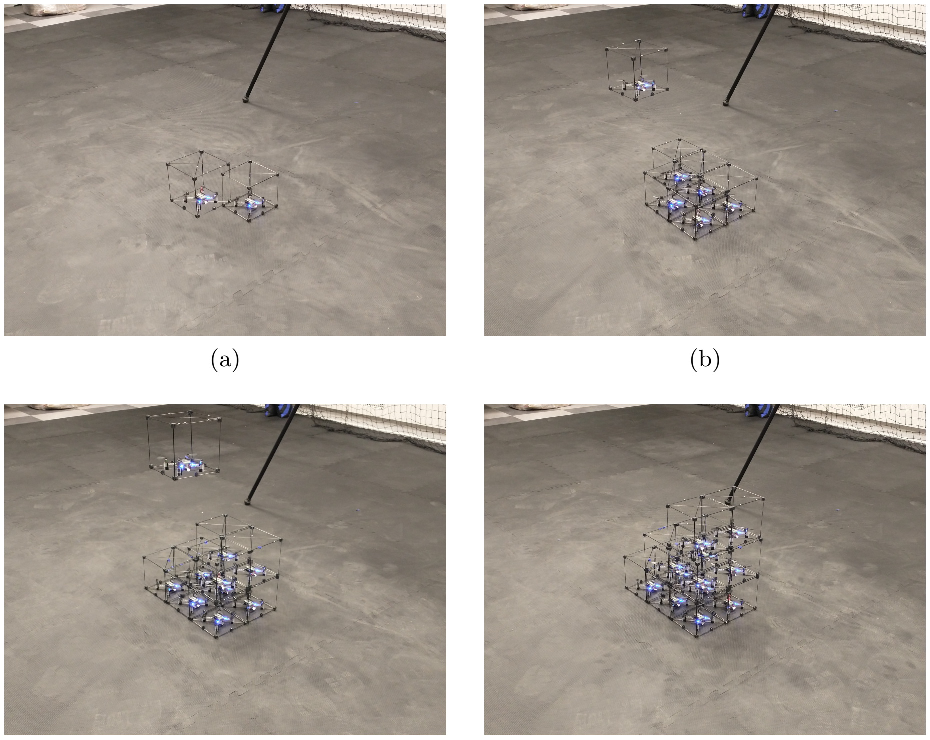

(a) (b) (c) (d)

Figure 6.14: Assembly of stairs with twelve robots.

were aerial quadrotors inside cubic frames. Due to limitations in the number of robots available, the structures used in simulations could not be tested with real robots.

7 Conclusion

This chapter concludes the work presented. In this thesis, we discuss the autonomous construction of three-dimensional structures. Specifically, the focus is on the self-assembly problem where robots are themselves structure parts. We accomplish self-assembly by proposing a new high-level architecture tailored to our problem. We then integrate movement, assignment, and placement solutions into this architecture in a way that encourages efficient inter-robot cooperation and minimizes energy consumption. Note that the self-assembly of structures is a diverse area of research. Its challenges and characteristics change depending on the robotic platform used, the environment in which the assembly is taking place, or even the structure representation. The broadness of the autonomous construction area, which include various robotic platforms [10, 11, 8], structure shapes [11, 36, 35], and assembly rules [100, 10], leverage the self-assembly problem to be used as a framework to discuss different aspects of multi-robotic systems.

In Chapter 3 we start by proposing a systematization of the self-assembly problem, i.e., the description of the overall architecture with the flow of information in the process. With the elements of the assembly system properly defined, our analysis shows that further contributions could be made in the planning and control stages, as well as in the structure representation. The assembly process is then organized into three distinct stages. First, parts move from the deployment location to the assembly location while maintaining cohesion and moving as a group. Second, an assembly order is computed among robots in the group. Such computation defines a sequence in which robots move to a structure being assembled. Finally, robots search for a position in the structure being assembled to place themselves.

We can highlight the contributions on this thesis as:

- The use of a connectivity metric to identify network fragmentation (Chapter 4);
- The concept of indivisible subgroups (minimal groups) and the proposal of a cohesion control term to control these groups (Chapter 4);

- A pin selection method based on network controllability (Chapter 4);
- The computation of the robot order using auction-based task assignment and graph methods (Chapter 5);
- A blueprint-based layered assembly of three-dimensional structures that implemented directed growth on each layer (Chapter 6);
- The set of behaviours to be programmed at each robot that enabled the assembly process (Chapter 6);
- A structure design method that uses network controllability to define the positions of the seeds (Chapter 6).

The remainder of this chapter discusses the stages of the assembly process and details the contributions mentioned above. Lastly, recommendations for future work are presented.

7.1 Moving of parts in space

The first step of the robotic assembly is conceived as the movement of robots from deployment position to the assembly location. To represent objects, other robots, or even structures in space, circular obstacles were added to the environment. Network control techniques and swarm analysis are used to design the control laws to be programmed on the robots. Also, planning algorithms are used to find a path in the environment to be taken by agents on the swarm to take the robots to the assembly location.

Our contributions start by proposing the *connectivity metric*, m_i, (Section 4.2.1) which is to determine if robots in the swarm are moving away from each other, and network connections are about to be broken. We use such metric to actively rewire network connections and split the swarm into smaller groups, reducing group fragmentation. We also propose the definition of *minimal groups* which could not be divided and are controlled by applying a *cohesion control term*, defined in Section 4.2.4. The cohesion control term was able to correctly avoid fragmentation of minimal groups even in the event of multiple obstacle encounters. Moreover, we propose a controllability-based method to choose the control node of a group in the swarm (Section 4.2.2). The use of an energy-related metric for a swarm of self-assembly parts showed to be more effective than a random selection. As a result, the swarm moved faster between waypoints, with less control signal input.

Results from Chapter 4 show that properly defined network control techniques can guide the robots through space by applying local control stimuli. Two advantages of the decentralized control solution proposed can be highlighted. First, our control strategy does not depend on the robot's model, being applicable to various robotic platforms. Second, the control strategy and the pin selection method are robust to different network topologies, being applicable to different arrangements of robots in space.

Another positive aspect of the flocking controller as proposed in this thesis is how it accommodates swarms with larger number of robots. The use of pinning control technique reduces the communication bandwidth needed to control the group. Also, obstacle encounters and robot-to-robot interactions are dealt with at each agent without the necessity of information from other swarm members.

7.2 Ordering and task assignment

The second step of the assembly is responsible to compute a sequence among the robots to move to the structure. That is modelled as a task assignment problem, where each requisition of a new part (coming from a structure being assembled) is viewed as a task to be assigned to one of the robots in the swarm.

In Chapter 5 a combination between graph analysis and auction process to solve the assignment problem is used. We propose the computation of a Minimum Spanning Tree to identify the robots in the network that could be removed without compromising the communication between robots in the swarm. Also, we define the messages that should be exchanged by robots to execute the auction process and how the information exchanged is processed. We propose the algorithms to perform the auction and analyzed their behaviour (Section 5.3), proving that they converge to the correct completion by assigning all robots to structures.

The energy consumed by robots while executing the ordering stage was used to compare the performance of our strategy with a baseline approach. Simulated results showed that, on average, the auction-based assignment strategy required less power from robots to wait (before assignment) and move to a structure (after assignment). In a realistic scenario, a higher energy level when proceeding to the next stage is advantageous once it increases the chance of each robot completing the assembly task. On the other hand, in the proposed approach, the time to assign all agents grows (linearly) as the number of agents grow, which could be problematic for larger swarm sizes. We discuss in Section 7.4 possible strategies to speed up the assignment problem.

7.3 Placement using local information

The final stage of the self-assembly problem is the insertion of the autonomous part into an empty position in the structure. For that, techniques are proposed where robots used only local information (based on embedded sensing) with no global knowledge about the assembly status.

In Chapter 6, we propose a self-assembly procedure for a three-dimensional layered assembly. We introduce the behaviours to be programmed for each robot, namely *identifying layer*, *circling* and *moving internally* (Section 6.2). We show that a single robot is capable of placing itself using local information by sequentially triggering these behaviours. We also define the messages that circulate between robots and the information needed by each one of the behaviours. Moreover, we present mathematical proofs that the self-assembly process leads to the construction of a desired shape (Section 6.2.2).

The behavioural rules guide the robots to search for the best position in the structure such that at every layer, the structure grew from a single position, called the seed position. Simulated results show the efficacy of the self-assembly procedure (Section 6.3). It was demonstrated that such algorithms are able to assemble the structure with robots moving close to the minimum trajectory possible. We also analyze how the selection of seed positions influences the assembly problem by using the Network Controllability Gramian and the closeness centrality (Section 6.3.4). The proper selection of seeds was demonstrated to reduce the assembly time by reducing robot motion.

7.4 Future work

This thesis contributes to different areas within the autonomous construction problem. However, not all problems in the field were solved within this work. Below we discuss possible research on each one of the investigated areas.

A key aspect of self-assembly construction is robustness to failures or noises in the system [20]. There is a trade-off between the capability of dealing with failures and the existence of global information in the system. Centralized controllers or global sensors may enable assembly coordination but introduce a single point of failure. In this thesis, we recognize that pins (moving stage), auctioneer (ordering stage), and seeds (placing stage) are single points of failure. Commonly, the solution to these issues come in the form of decentralized methods [17, 66, 73, 10] or redundancy in the number of robots [11] and could be studied in the future.

Besides the points of failure described above, the system is also nonreactive to individual robot failure. The actions of individual robots are not monitored by the ensemble and therefore the system can not react if one of them fails. For each one of the self-assembly stages, individual robotic failure can be handled differently. For example, during the moving and ordering stages, a faulty robot could be easily replaced with the proper adjustments of the algorithms and system properties to allow redundancy. On the other hand, in the planning stage, seed robots need to identify that a robot failed to place itself to adjust their internal counters (robots in the layer) accordingly. Therefore, algorithms need to be redesigned and the communication extended (with already placed robots remaining active as in [11]).

Another possible extension of this research involves improving path planning efficiency in the moving stage. For example, learning [87] and control [101] can be applied to optimize the planning algorithm. Usually, these strategies only consider the dynamics of a single robot. Instead, the positive results obtained in this work (Sections 4.3.2 and 6.3.4) lead us to believe that network control methods (e.g., Controllability Gramian [52], Laplacian [56]) could be used to optimize path planning strategies for a group of robots.

In the ordering stage, the number of decision agents could be increased with the assignment process being decentralized. In this way, it is expected that multiple requests for new structure parts could be handled simultaneously, instead of the serial and centralized approach taken in this thesis. That could increase power efficiency during assignment and decrease the total assignment time. MDP and consensus algorithms are used in the literature to deal with decentralized decisions [73, 68]. However, further research can be carried on the application of decentralized and learning techniques to the ordering stage of self-assembly problems.

In the placing stage, more information could be added to improve the search process for an empty position. Already placed blocks could remain active and keep local information about the assembly status. For example, already placed robots could keep the information about empty positions in their surroundings and guide moving robots to empty positions faster. A gradient-based search is shown to effectively place parts in the two-dimensional structure [11] using local communication between agents. In the context of this work, local communication can enable, for example, robots to start their circling behaviour from any layer (instead of the bottom layer). The search for the empty position could apply gradient-based methods and would be guided by already placed robots.

The *circling* and *moving internally* behaviours (Section 6.2) constrain the movement of the robot to the horizontal plane. However, the fact that aerial vehicles can operate at different heights could be exploited. In Section 6.3.5, a complex structure was assembled by dividing the process into multiple substructures. Note that the top of the bridge was assembled above others, in midair. That is an example of how layers don't necessarily need to be assembled at the same place nor sequentially. Given that robotic parts could perform coordinated connected flight [42], layers can be simultaneously assembled in different places being stacked on top of each other after completed.

Finally, in this section we pointed out some future research topics, given the advances of this work and the current literature. Future research can include the study of redundancy to increase robustness to failures, the use of decentralized methods for assignment and information storage, and the redesign of assembly rules to optimize assembly efficiency.

Bibliography

[1] K. M. Cabral, S. N. Givigi, S. R. B. dos Santos, and P. T. Jardine, "Design of a self-assembly system of three-dimensional structures using autonomous construction blocks," in *2019 IEEE International Systems Conference (SysCon)*, pp. 1–8, 2019.

[2] A. Warszawski and R. Navon, "Implementation of Robotics in Building: Current Status and Future Prospects," *Journal of Construction Engineering and Management*, vol. 124, pp. 31–41, Jan. 1998.

[3] J. Wawerla, G. Sukhatme, and M. Mataric, "Collective construction with multiple robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, pp. 2696–2701, 2002.

[4] A. Bolger, M. Faulkner, D. Stein, L. White, S. Yun, and D. Rus, "Experiments in decentralized robot construction with tool delivery and assembly robots," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5085–5092, Oct. 2010.

[5] J. Werfel, K. Petersen, and R. Nagpal, "Designing Collective Behavior in a Termite-Inspired Robot Construction Team," *Science*, vol. 343, pp. 754–758, Feb. 2014.

[6] S. R. B. dos Santos, C. L. Nascimento, and S. N. Givigi, "Planning and learning for cooperative construction task with quadrotors," in *2014 IEEE International Systems Conference Proceedings*, pp. 57–64, 2014.

[7] J. Willmann, F. Augugliaro, T. Cadalbert, R. D'Andrea, F. Gramazio, and M. Kohler, "Aerial Robotic Construction towards a New Field of Architectural Research," *International Journal of Architectural Computing*, vol. 10, pp. 439–459, Sep. 2012.

[8] J. Seo, M. Yim, and V. Kumar, "Assembly sequence planning for constructing planar structures with rectangular modules," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5477–5482, IEEE, May 2016.

[9] J. Paulos, N. Eckenstein, T. Tosun, J. Seo, J. Davey, J. Greco, V. Kumar, and M. Yim, "Automated Self-Assembly of Large Maritime Structures by a Team of Robotic Boats," *IEEE Transactions on Automation Science and Engineering*, vol. 12, pp. 958–968, Jul. 2015.

[10] E. Klavins, "Programmable self-assembly," *IEEE Control Systems Magazine*, vol. 27, no. 4, pp. 43–56, 2007.

[11] M. Rubenstein, A. Cornejo, and R. Nagpal, "Programmable self-assembly in a thousand-robot swarm," *Science*, vol. 345, no. 6198, pp. 795–799, 2014.

[12] M. P. B. Magueta, S. R. B. dos Santos, F. A. M. Cappabianco, and S. N. Givigi, "Designing collective behavior for construction of containment structures using actuated blocks," in *2020 IEEE International Systems Conference (SysCon)*, pp. 1–8, 2020.

[13] Z. Zeravcic and M. P. Brenner, "Spontaneous emergence of catalytic cycles with colloidal spheres," *Proceedings of the National Academy of Sciences*, vol. 114, no. 17, pp. 4342–4347, 2017.

[14] W. Yu, G. Chen, and J. Lü, "On pinning synchronization of complex dynamical networks," *Automatica*, vol. 45, pp. 429–435, Feb. 2009.

[15] J. Cortés, S. Martínez, and F. Bullo, "Robust rendezvous for mobile autonomous agents via proximity graphs in arbitrary dimensions," *IEEE Transactions on Automatic Control*, vol. 51, no. 8, pp. 1289–1298, 2006.

[16] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proceedings of the IEEE*, vol. 95, pp. 215–233, Jan. 2007.

[17] J. Cortés and M. Egerstedt, "Coordinated control of multi-robot systems: A survey," *SICE Journal of Control, Measurement, and System Integration*, vol. 10, no. 6, pp. 495–503, 2017.

[18] G. Notomista and M. Egerstedt, "Constraint-driven coordinated control of multi-robot systems," in *2019 American Control Conference (ACC)*, pp. 1990–1996, 2019.

[19] J. Seo, *Grasping and Assembling with Modular Robots*. PhD thesis, Publicly Accessible Penn Dissertations, 2014.

[20] K. H. Petersen, N. Napp, R. Stuart-Smith, D. Rus, and M. Kovac, "A review of collective robotic construction," *Science Robotics*, vol. 4, no. 28, p. eaau8479, 2019.

[21] K. M. Cabral, S. N. Givigi, and P. T. Jardine, “Autonomous assembly of structures using pinning control and formation algorithms,” in *2020 IEEE International Systems Conference (SysCon)*, pp. 1–7, 2020.

[22] K. M. Cabral, S. N. Givigi, and P. T. Jardine, “Obstacle avoidance of swarms using pinning control,” *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 9628–9635, 2020. 21st IFAC World Congress.

[23] K. M. Cabral, T. Kaykobad, J.-A. Delamer, P. T. Jardine, and S. N. Givigi, “Design of a decentralized strategy for layered self-assembly of 3d structures using robotic blocks,” *Revision Requested*, 2022.

[24] S. R. B. dos Santos, S. N. Givigi, and C. L. Nascimento, “Autonomous construction of multiple structures using learning automata: Description and experimental validation,” *IEEE Systems Journal*, vol. 9, no. 4, pp. 1376–1387, 2015.

[25] S. R. dos Santos, S. N. Givigi, C. L. Nascimento, J. M. Fernandes, L. Buonocore, and A. A. Neto, “Iterative decentralized planning for collective construction tasks with quadrotors,” *Journal of Intelligent & Robotic Systems*, vol. 90, no. 1, pp. 217–234, 2018.

[26] Y. Zheng, M. Allwright, W. Zhu, M. Kassawat, Z. Han, and M. Dorigo, “Swarm construction coordinated through the building material,” in *Benelux Conference on Artificial Intelligence*, pp. 188–202, Springer, 2020.

[27] S. R. B. dos Santos, D. O. Dantas, S. N. Givigi, L. Buonocore, A. A. Neto, and C. L. Nascimento, “A stochastic learning approach for construction of brick structures with a ground robot,” *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 5654–5659, 2017. 20th IFAC World Congress.

[28] M. Dogar, R. A. Knepper, A. Spielberg, C. Choi, H. I. Christensen, and D. Rus, “Multi-scale assembly with robot teams,” *The International Journal of Robotics Research*, vol. 34, pp. 1645–1659, Nov. 2015.

[29] A. Stroupe, T. Huntsberger, A. Okon, H. Aghazarian, and M. Robinson, “Behavior-based multi-robot collaboration for autonomous construction tasks,” in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1495–1500, Aug. 2005.

[30] Q. Lindsey, D. Mellinger, and V. Kumar, “Construction with quadrotor teams,” *Autonomous Robots*, vol. 33, pp. 323–336, Oct. 2012.

[31] A. Jimenez-Cano, J. Martin, G. Heredia, A. Ollero, and R. Cano, “Control of an aerial robot with multi-link arm for assembly tasks,” in *2013*

IEEE International Conference on Robotics and Automation, pp. 4916–4921, 2013.

[32] K. M. Cabral, S. R. B. dos Santos, S. N. Givigi, and C. L. Nascimento, "Design of model predictive control via learning automata for a single uav load transportation," in *2017 Annual IEEE International Systems Conference (SysCon)*, pp. 1–7, 2017.

[33] S. R. dos Santos, S. N. Givigi, C. L. Nascimento, J. M. Fernandes, L. Buonocore, and A. A. Neto, "Iterative decentralized planning for collective construction tasks with quadrotors," *Journal of Intelligent & Robotic Systems*, vol. 90, no. 1-2, pp. 217–234, 2018.

[34] Y. Zhang, Y. Koga, and D. Balkcom, "Interlocking block assembly with robots," *IEEE Transactions on Automation Science and Engineering*, vol. 18, no. 3, pp. 902–916, 2021.

[35] J. Werfel and R. Nagpal, "Three-dimensional construction with mobile robots and modular blocks," *The International Journal of Robotics Research*, vol. 27, no. 3-4, pp. 463–479, 2008.

[36] M. T. Tolley and H. Lipson, "On-line assembly planning for stochastically reconfigurable systems," *The International Journal of Robotics Research*, vol. 30, no. 13, pp. 1566–1584, 2011.

[37] F. Augugliaro, S. Lupashin, M. Hamer, C. Male, M. Hehn, M. W. Mueller, J. S. Willmann, F. Gramazio, M. Kohler, and R. D'Andrea, "The flight assembled architecture installation: Cooperative construction with flying machines," *IEEE Control Systems Magazine*, vol. 34, no. 4, pp. 46–64, 2014.

[38] S. Hormoz and M. P. Brenner, "Design principles for self-assembly with short-range interactions," *Proceedings of the National Academy of Sciences*, vol. 108, no. 13, pp. 5193–5198, 2011.

[39] M. Gauci, R. Nagpal, and M. Rubenstein, *Programmable Self-disassembly for Shape Formation in Large-Scale Robot Collectives*, pp. 573–586. Springer International Publishing, 2018.

[40] K. Stoy and R. Nagpal, "Self-reconfiguration using directed growth," in *In Proc. 7th Int. Symp. on Distributed Autonomous Robotic Systems*, pp. 1–10, 2004.

[41] J. Seo, M. Yim, and V. Kumar, "Assembly planning for planar structures of a brick wall pattern with rectangular modular robots," in *2013 IEEE International Conference on Automation Science and Engineering (CASE)*, pp. 1016–1021, IEEE, Aug. 2013.

[42] D. Saldana, B. Gabrich, G. Li, M. Yim, and V. Kumar, "Modquad: The flying modular structure that self-assembles in midair," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 691–698, 2018.

[43] N. Gandhi, D. Saldana, V. Kumar, and L. T. X. Phan, "Self-reconfiguration in response to faults in modular aerial systems," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2522–2529, 2020.

[44] A. Y. Yazıcıoğlu, M. Egerstedt, and J. S. Shamma, "Formation of robust multi-agent networks through self-organizing random regular graphs," *IEEE Transactions on Network Science and Engineering*, vol. 2, no. 4, pp. 139–151, 2015.

[45] J. Fu, Y. Lv, G. Wen, and X. Yu, "Local measurement based formation navigation of nonholonomic robots with globally bounded inputs and collision avoidance," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 3, pp. 2342–2354, 2021.

[46] M. Coppola, J. Guo, E. Gill, and G. C. de Croon, "Provable self-organizing pattern formation by a swarm of robots with limited knowledge," *Swarm Intelligence*, vol. 13, no. 1, pp. 59–94, 2019.

[47] H. Medellin, J. R. Corney, J. B. C. Davies, T. Lim, and J. M. Ritchie, "Octree-based production of near net shape components," *IEEE Transactions on Automation Science and Engineering*, vol. 5, no. 3, pp. 457–466, 2008.

[48] L. Zhang, Z.-H. Fu, H. Liu, Q. Liu, X. Ji, and H. Qian, "An efficient parallel self-assembly planning algorithm for modular robots in environments with obstacles," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 10038–10044, 2021.

[49] R. Olfati-Saber, "Flocking for multi-agent dynamic systems: algorithms and theory," *IEEE Transactions on Automatic Control*, vol. 51, pp. 401–420, Mar. 2006.

[50] G. Vásárhelyi, C. Virágh, G. Somorjai, T. Nepusz, A. E. Eiben, and T. Vicsek, "Optimized flocking of autonomous drones in confined environments," *Science Robotics*, vol. 3, no. 20, p. eaat3536, 2018.

[51] E. Nozari, F. Pasqualetti, and J. Cortés, "Heterogeneity of central nodes explains the benefits of time-varying control scheduling in complex dynamical networks," *Journal of Complex Networks*, Feb. 2019.

[52] F. Pasqualetti, S. Zampieri, and F. Bullo, "Controllability metrics, limitations and algorithms for complex networks," *IEEE Transactions on Control of Network Systems*, vol. 1, pp. 40–52, Mar. 2014.

[53] D. Wu, J. Feng, and Y. Zhao, "Synchronization for switching networks with multiple time-delay via aperiodically intermittent pinning control," in *2018 Chinese Control And Decision Conference (CCDC)*, pp. 6050–6055, IEEE, Jun. 2018.

[54] G. Wen, P. Wang, X. Yu, W. Yu, and J. Cao, "Pinning synchronization of complex switching networks with a leader of nonzero control inputs," *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–13, 2019.

[55] X. Liu, T. Chen, and W. Lu, "Consensus problem in directed networks of multi-agents via nonlinear protocols," *Physics Letters A*, vol. 373, no. 35, pp. 3122–3127, 2009.

[56] R. Olfati-Saber and R. Murray, "Consensus problems in networks of agents with switching topology and time-delays," *IEEE Transactions on Automatic Control*, vol. 49, pp. 1520–1533, Sep. 2004.

[57] T.-H. Cheng, Z. Kan, J. R. Klotz, J. M. Shea, and W. E. Dixon, "Event-triggered control of multiagent systems for fixed and time-varying network topologies," *IEEE Transactions on Automatic Control*, vol. 62, pp. 5365–5371, Oct. 2017.

[58] K. Laventall and J. Cortés, "Coverage control by multi-robot networks with limited-range anisotropic sensory," *International Journal of Control*, vol. 82, no. 6, pp. 1113–1121, 2009.

[59] X. Wang and H. Su, "Pinning control of complex networked systems: A decade after and beyond," *Annual Reviews in Control*, vol. 38, no. 1, pp. 103–111, 2014.

[60] Zhi-Hong Guan, Zhi-Wei Liu, Gang Feng, and Yan-Wu Wang, "Synchronization of complex dynamical networks with time-varying delays via impulsive distributed control," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, pp. 2182–2195, Aug. 2010.

[61] A. Adaldo, F. Alderisio, D. Liuzza, G. Shi, D. V. Dimarogonas, M. di Bernardo, and K. H. Johansson, "Event-triggered pinning control of switching networks," *IEEE Transactions on Control of Network Systems*, vol. 2, pp. 204–213, Jun. 2015.

[62] X. Wang, X. Li, and J. Lu, "Control and flocking of networked systems via pinning," *IEEE Circuits and Systems Magazine*, vol. 10, pp. 83–91, Aug. 2010.

[63] S.-J. Chung, A. A. Paranjape, P. Dames, S. Shen, and V. Kumar, "A survey on aerial swarm robotics," *IEEE Transactions on Robotics*, vol. 34, pp. 837–855, Aug. 2018.

[64] S. J. Rasmussen and T. Shima, "Tree search algorithm for assigning cooperating uavs to multiple tasks," *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal*, vol. 18, no. 2, pp. 135–153, 2008.

[65] M. Alighanbari and J. P. How, "Decentralized task assignment for unmanned aerial vehicles," in *Proceedings of the 44th IEEE Conference on Decision and Control*, pp. 5668–5673, 2005.

[66] H.-L. Choi, L. Brunet, and J. P. How, "Consensus-based decentralized auctions for robust task allocation," *IEEE transactions on robotics*, vol. 25, pp. 912–926, Aug. 2009.

[67] J. Chen, K. Xiao, K. You, X. Qing, F. Ye, and Q. Sun, "Hierarchical task assignment strategy for heterogeneous multi-uav system in large-scale search and rescue scenarios," *International Journal of Aerospace Engineering*, vol. 2021, 2021.

[68] X. Jia and M. Q.-H. Meng, "A survey and analysis of task allocation algorithms in multi-robot systems," in *2013 IEEE International Conference on Robotics and biomimetics (ROBIO)*, pp. 2280–2285, 2013.

[69] B. P. Gerkey and M. J. Mataric, "Sold!: Auction methods for multirobot coordination," *IEEE transactions on robotics and automation*, vol. 18, pp. 758–768, Oct. 2002.

[70] M. Otte, M. J. Kuhlman, and D. Sofge, "Auctions for multi-robot task allocation in communication limited environments," *Autonomous Robots*, vol. 44, no. 3, pp. 547–584, 2020.

[71] T. Campbell, L. Johnson, and J. P. How, "Multiagent allocation of markov decision process tasks," in *2013 American Control Conference*, pp. 2356–2361, 2013.

[72] S. Proper and P. Tadepalli, "Solving multiagent assignment markov decision processes," in *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pp. 681–688, 2009.

[73] S. Abdallah and V. Lesser, "Modeling task allocation using a decision theoretic model," in *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pp. 719–726, 2005.

[74] P. Plamondon, B. Chaib-draa, and A. R. Benaskeur, "A multiagent task associated mdp (mtamdp) approach to resource allocation.," in *AAAI Spring Symposium: Distributed Plan and Schedule Management*, pp. 89–96, 2006.

[75] R. Liu, M. Seo, B. Yan, and A. Tsourdos, "Decentralized task allocation for multiple uavs with task execution uncertainties," in *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 271–278, 2020.

[76] V. Thangavelu, Y. Liu, M. Saboia, and N. Napp, "Dry stacking for automated construction with irregular objects," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4782–4789, 2018.

[77] J. C. Rubens, "Bitdrones: Design of a tangible drone swarm as a programmable matter interface," Master's thesis, Queen's University, Kingston, Ontario, Canada, 2019.

[78] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Tech. Rep. 98-11, Computer Science Dept., Iowa State University, 1998.

[79] M. Tognon, C. Gabellieri, L. Pallottino, and A. Franchi, "Aerial comanipulation with cables: The role of internal force for equilibria, stability, and passivity," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2577–2583, 2018.

[80] H. Rastgoftar and E. M. Atkins, "Cooperative aerial payload transport guided by an in situ human supervisor," *IEEE Transactions on Control Systems Technology*, vol. 27, no. 4, pp. 1452–1467, 2019.

[81] F. L. Cortesi, T. H. Summers, and J. Lygeros, "Submodularity of energy related controllability metrics," in *53rd IEEE Conference on Decision and Control*, pp. 2883–2888, 2014.

[82] T. H. Summers, F. L. Cortesi, and J. Lygeros, "On submodularity and controllability in complex dynamical networks," *IEEE Transactions on Control of Network Systems*, vol. 3, no. 1, pp. 91–101, 2015.

[83] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[84] A. Stentz, "Optimal and efficient path planning for partially known environments," in *Intelligent unmanned ground vehicles*, pp. 203–220, Springer, 1997.

[85] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, pp. 566–580, Aug. 1996.

[86] K. Naderi, J. Rajamäki, and P. Hämäläinen, "Rt-rrt* a real-time path planning algorithm based on rrt," in *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games*, pp. 113–118, 2015.

[87] L. Palmieri and K. O. Arras, "Distance metric learning for rrt-based motion planning with constant-time inference," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 637–643, 2015.

[88] A. T. Khan, S. Li, S. Kadry, and Y. Nam, "Control framework for trajectory planning of soft manipulator using optimized rrt algorithm," *IEEE Access*, vol. 8, pp. 171730–171743, 2020.

[89] P. Kavathekar and Y. Chen, "Vehicle platooning: A brief survey and categorization," in *ASME 2011 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pp. 829–845, American Society of Mechanical Engineers, 2011.

[90] S. Huang and W. Ren, "Longitudinal control with time delay in platooning," *IEE Proceedings-Control Theory and Applications*, vol. 145, no. 2, pp. 211–217, 1998.

[91] R. L. Graham and P. Hell, "On the history of the minimum spanning tree problem," *Annals of the History of Computing*, vol. 7, no. 1, pp. 43–57, 1985.

[92] S. Pettie and V. Ramachandran, "An optimal minimum spanning tree algorithm," *J. ACM*, vol. 49, p. 16–34, Jan 2002.

[93] Z. Yue, B. Lian, C. Tang, and K. Tong, "A novel adaptive federated filter for GNSS/INS/VO integrated navigation system," *Measurement Science and Technology*, vol. 31, p. 085102, May 2020.

[94] D. Wang, K. O'Keefe, and M. Petovello, "Decentralized cooperative positioning for vehicle-to-vehicle (v2v) application using gps integrated with uwb range," in *Proceedings of the ION 2013 Pacific PNT Meeting*, pp. 793–803, 2013.

[95] A. Gómez-Casasola and H. Rodríguez-Cortés, "Sensor fusion for quadrotor autonomous navigation," in *2020 American Control Conference (ACC)*, pp. 5219–5224, 2020.

[96] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[97] T. C. Hales, "The jordan curve theorem, formally and informally," *The American Mathematical Monthly*, vol. 114, no. 10, pp. 882–894, 2007.

[98] G. Silano, E. Aucone, and L. Iannelli, "Crazys: A software-in-the-loop platform for the crazyflie 2.0 nano-quadcopter," in *2018 26th Mediterranean Conference on Control and Automation (MED)*, pp. 1–6, 2018.

[99] S. Baker and I. Matthews, "Lucas-kanade 20 years on: A unifying framework," *International journal of computer vision*, vol. 56, no. 3, pp. 221–255, 2004.

[100] J. Werfel, "Building Blocks for Multi-robot Construction," in *Distributed Autonomous Robotic Systems 6* (R. Alami, R. Chatila, and H. Asama, eds.), pp. 285–294, Springer Japan, 2007.

[101] S. Primatesta, A. Osman, and A. Rizzo, "Mp-rrt#: a model predictive sampling-based motion planning algorithm for unmanned aircraft systems," *Journal of Intelligent & Robotic Systems*, vol. 103, no. 4, pp. 1–13, 2021.