UNIVERSITY OF SOUTHAMPTON

FACULTY OF PHYSICAL SCIENCE AND ENGINEERING

Electronics and Computer Science

Self-Organising Assembly using Swarm Robots

by

Niken Syafitri

Thesis for the degree of Doctor of Philosophy

March 2018

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF PHYSICAL SCIENCE AND ENGINEERING
Electronics and Computer Science

Doctor of Philosophy

SELF-ORGANISING ASSEMBLY USING SWARM ROBOTS

by Niken Syafitri

When a swarm of small robots is required to operate in an unstructured and unknown environment, adaptability is required, particularly in the case of traversing a void (typically a crevasse or gap between two stable platforms). Adaptability is required when robots face a void of an unknown size so that they can safely cross. By applying adaptability, the robustness of the swarm can be maintained either by crossing a void or safely retreating after failing to cross. However, the relatively simple characteristics of swarm robots limit their performance in a complex task when they work individually, but it can be accomplished when they work collaboratively.

Solving the problem arised, a strategy of self-organising assembly has been proposed supported by simulation and practical experiments. Algorithms with simple rules have been developed that allows the robot swarm to reach the target area from an initial zone separated by the void, where the size and location of the void are unknown. The configuration for the dynamic structure have been proposed built by the robot swarm to bridging two separated zones. To measure the performance of the swarm, metrics have been identified within the simulation environment. A number of 3D printed platforms have been designed and evaluated to investigate the emerging physical problems including the required docking system, platform shape, approaches to actuation and communication system.

Several contributions have been achieved, in particular the approach that proposes a dynamic structure with simple rules that is constructed by a robotic swarm to cross the void, where the size and location of the void are unknown. This is in accordance with the concept of self-organise and self-assembly. The solution developed is scalable, with any number of robots can be added into the swarm. Any robot can also be the leader who initiates the construction or detects the target. When the size of the swarm does not satisfy with the size of the void, the swarm will retreat to the initial zone to prevent the robots lost. To support the rigid dynamic structure, a number of docking mechanisms are proposed.

Contents

List of Figures

List of Tables

Declaration of Authorship

I, Niken Syafitri , declare that the thesis entitled *Self-Organising Assembly using Swarm Robots* and the work presented in the thesis are both my own, and have been generated by me as the result of my own original research. I confirm that:

- this work was done wholly or mainly while in candidature for a research degree at this University;
- where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- where I have consulted the published work of others, this is always clearly attributed;
- where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
- the OctBot was designed by GDP students (Aziz et al., 2015), and its coupling by a summer intern, J. D. Mazlan;
- parts of this work has been published as IEEE IROS 2015 paper and has been accepted as IEEE SMC 2017 paper (Syafitri et al., 2015, 2017)

Signed:..

Date:..

Acknowledgements

I would like to express my sincere gratitude to the supervisory team, Dr. Richard M. Crowder and Dr. Paul H. Chappell, for supporting me in research and writing this thesis, for their patience and every discussion throughout the study, and for accepting and trusting me as their student. And I would also like to thank the examiners, Dr. Christopher T. Freeman and Prof. Hongnian Yu, for discussion and every suggestion given.

I would like to thank the Directorate General of Resources for Science, Technology and Higher Education, the Ministry of Research, Technology and Higher Education of the Republic of Indonesia for the funding. And I would also like to thank the National Institute of Technology, Bandung, Indonesia, for the support and the approval of studying abroad.

I am also grateful to the supportive wellbeing team of University of Southampton during my difficult times. I would like to thank Dr. Andrew J. Chipperfield, the Agents, Interaction and Complexity (AIC) Research Group of Electronics and Computer Science (ECS) Department in the Faculty of Physical Science and Engineering, Prof. Philippa A. S. Reed, the Faculty of Engineering and the Environment, and the University of Southampton for the support of transferring faculty process.

I would also like to thank my friends for their friendship, especially Brittany, Amr, Rosa Karnita and family, Saikiran, Dr. Whysnianti Basuki and Pak Gunawan Wibisono, as my best friends, helpers, and my new family; Adil, Beining, and Lenka, for every discussion of swarm robotics; my fellow doctoral students in AIC and WAIS Research Groups; Mbak Lisa and Auralius of UZH and ETH Zurich; my hallmates at St Margaret's House; the ghibah team and all members of Indonesian Society in Southampton.

Last but not the least, I would like to thank my family, Mama, Papa, Adik, Buliek Didiet, Oom Djono and Pak Eddy, for supporting me spiritually throughout the study, writing this thesis and my life in general.

Nomenclature

Initial zone	The zone which robots explore from the start until they detect the void; it is located at the left-hand side of the void
Target zone	The opposite of initial zone, which is separated by the void at which the target is located and is located at the right-hand side of the void
Seed robot	The first robot that detects the void and initiates the construction
Free robot	Robots at the initial zone that haven't been recruited to assemble the structure
A small void	A void that can be crossed by only a simple line structure of assembled swarm robots, the width of which is from one up to five robot void width
A large void	A void that requires a more complex structure of assembled swarm robots to be crossed, the width of which is larger than five robot void width
R_s	The seed robot, which is a robot that detects the void for the first time and initiates the construction
R_p	Robots whose positions are at the initial zone
R_g	Robots whose positions are over the void
R_o	Robots whose positions are at the opposite zone
n_s	Sum of R_s robots in the structure
n_p	Sum of R_p robots in the structure
n_g	Sum of R_g robots in the structure
n_o	Sum of R_o robots in the structure
N	Total number of robots in the structure
N_{min}	Minimum total number of robots in the structure to cross a certain width of void
L	The length of the structure of swarm robots
T_t	Sum of sampling time for full simulation
T_s	Sum of sampling time during exploration at the initial zone
T_o	Sum of sampling time during exploration at the target zone

T_b	Sum of sampling time during exploration at the initial zone after robots have retreated
T_a	Sum of sampling time during assembly process in construction
T_d	Sum of sampling time during disassembly process after robots have retreated

Chapter 1

Introduction

Social insects can perform complex tasks when they work collaboratively, yet which are impossible if an insect works as an individual. For example, ants can cross wide voids and pull leaf edges together to form nests; bees and wasps build their nests; and termites build their nests with remarkable heights and complex rooms, corridors and vents (Bonabeau et al., 1999). The capability of social insects has inspired swarm robotic research where robot swarms replace the insects.

Swarm robotics has been defined as the study of how to design a large number of relatively simple mobile robots that can work collectively to carry out a specific task (Şahin, 2005). Its application relies on a number of advantages, such as robustness, flexibility, and scalability (Martinoli, 2001). In addition, its characteristics should have the following features that differentiate swarm robotics from other multi-robot systems (Beni, 2005; Şahin, 2005):

- The robots should be autonomous and mobile.
- The swarm of robots has a decentralised control or coordination mechanism and lack of synchronicity.
- A robot swarm should be a large number of robots and, while the swarm could consist of a small number of robots, a swarm robotic system aims at scalability; it can work even if the number of robots changes.
- The robots have limited sensing and communication abilities.
- The swarm of robots should consist of relatively homogeneous robots. It can consist of a few different types of robots, but the number of robots of each type should be large.
- Robots can engage in collective behaviour through the local interactions among robots and between robots and the environment.

Swarm robotic systems have many possible applications, including exploration, surveillance, search and rescue, humanitarian demining, intrusion tracking, cleaning, inspection and transportation of large objects (Brambilla et al., 2013) in which they will be deployed in unstructured and largely unknown environments. In contrast, the majority of research based on swarm robotic systems operates within structured environments. The difference between a laboratory based excersice and a real world application can be illustrated by considering the following search and rescue. Following any major disaster, either natural or man-made, dynamic and uncertain conditions result. The search and rescue operation for survivors becomes a priority yet difficult because rubble obstructs the effort of the operation team to reach the victims. However, the safety of both rescuers and victims has to be considered in order to reduce the risk of potential loss of life.

A rescue operation is a time critical action. Besides the main aim to save lives, the objectives are to reduce dangerous tasks, minimise physical demands such as rescuers become tired or exhausted and negative psychological aspects such as anxiety levels, critical incident stress (CIS), depression and post-traumatic stress disorder (PTSD). As the rescue timing is critical in terms of life-or-death, obstacles (ruins or voids) might delay the rescue as the rescue team has to remove or traverse them first before reaching the victims. As a consequence, it will affect the completion of the other task (Ramchurn et al., 2015), namely saving the lives of survivors.

Statheropoulos et al. (2015) have stated a requirement to deploy small tools that work in confined areas in a rescue operation in order to reduce the weight and dimensions of tools and to retain the stability of buildings and ruins. Another significant aspect is the fast and wide-scale deployment of inexpensive tools to speed up the operation. These requirements aim to maximise the chance of survival and raise the possibility of swarm robots, which are light, small, inexpensive and can be deployed fast and over a wide area, to operate in a rescue operation. Therefore, swarm robots are able to play the important role in a search and rescue operation as they have to be in small size yet simple, but can be deployed in large scale.

There is a specific case in the real search and rescue operation when swarm robots have to bridging two separated area to reach the victims, and several approaches have been proposed to solve this case. However, when a swarm is required to traverse an obstacle between two separated zones, the majority of the proposed approaches have used preconditioned obstacles which, instead of overcoming the challenge, bring about other problems, such as the inability of the swarm to traverse a changeable or unknown obstacle size. A number of research groups have proposed heterogeneity as an approach to solve the unstructured environment problem (Kernbach et al., 2011; Dorigo et al., 2013). However, heterogeneity is costly, as it requires different types of robots and each type should present in large numbers.

To answer the challenge of an unstructured environment, a number of research groups have proposed various approaches, such as climbing obstacles with a certain height (Kuyucu et al., 2013; Levi et al., 2014; Li et al., 2015; Vonásek et al., 2015) or avoiding holes or traversing a void (O'Grady et al., 2010, 2012; Trianni et al., 2014; Bruni et al., 2015), by assembling swarm robots to form a structure or robotic organism. However, these proposed approaches employ the strategy of assembling all robots in an initial zone to form a structure. After the structure is formed, the whole structure climbs, avoids or crosses the obstacle(s). After another defined area has been reached, the structure disassembles. Unfortunately, this method has significant problems when the size of the obstacle changes or if it is unknown. In addition, if the swarm size changes, there is a probability that the structure cannot traverse the obstacle to reach the other area.

Taking inspiration from nature, especially ant colonies, the adaptivity to overcome an unstructured and unknown environment can be achieved through such a remarkable mechanism. For example, how they bridge two separated zones by collaboratively making such a structure, either vertical or horizontal, which could not be achieved by the work of individuals, as shown in Figure 1.1.

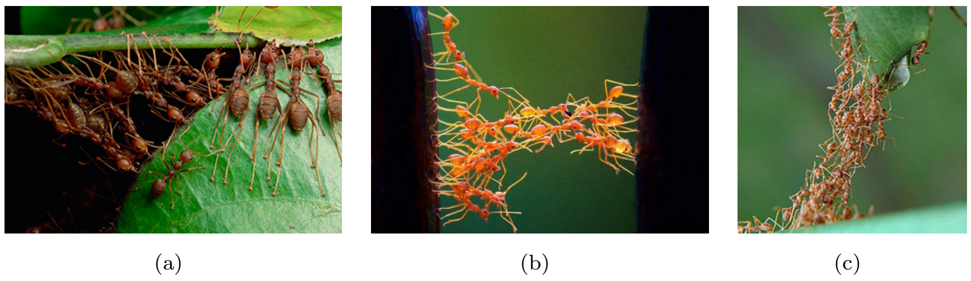


Figure 1.1: Ants forming a bridge. (a) Ants grip two leaves with gradient distance (Gaines, 2013); (b) ants form a bridge between two places with the same distance (Brennan, 2012); and (c) ants form a bridge of two places with different heights (Correll, 2011).

According to this case, while Cucu et al. (2015) proposed a vertical self-assembled structure that mimics the pyramidal tower structure of ants to reach higher area, as in Figure 1.1(c), this research considers a method whereby swarm robots build a structure to reach another area horizontally, as in Figure 1.1(b). In addition, another difference compared to other approaches is that the structure is built by making the processes of assembling, crossing the obstacle and disassembling operate in parallel. To be precise, in our approach, a robot swarm operates in an arena that has two zones separated by a void. At the initial process, the swarm robots are deployed on the initial zone at the left side of the arena. They wander in the arena and have to traverse the void to reach the other side where the target is located. After the target is found, the operation finishes. Inspired by ant colonies, in fact they retain memory or specialisation during the construction of the structure (Avril et al., 2016). However, considering the efficiency of simulation time and computation memory, keeping memory and specialisation for self-assembly arrangement is not applied in this research.

1.1 Research Contribution

According to the proposed approaches, there are a number of related contributions, including the supportive contributions, such as the hardware design and the simulation environment. To be specific, the contributions of this thesis are as follows:

1. **Developing a Matlab-based simulation environment**. As the existing robotic simulation environments are not supportive enough for swarm robotic systems, such as their purpose is for a single or two robots, or they are high definition resolution multi-robot simulations, or they are too complex to operate, or it is difficult to define the robot shape and features, a simulation environment using Matlab has been developed to execute the proposed methods. It is purposely a swarm robot simulation where any number of robots can be added into the swarm towards the scalability and users can choose whether there are a number of obstacles on the arena, or choose the robot shape between square and octagon in 2D.
2. **The dynamic simple line structure approach for small void crossing**. The proposed simple line structure is purposely for a swarm consisting of a small number of robots operating in an arena with two separated zones. The approach is robust as various sizes of swarm can traverse a range of void sizes to reach the opposite zone. If there is a lack of robot numbers resulting in an unstable structure, the swarm can retreat to the initial zone to prevent robots falling into the void. Here an analogy can be taken; the swarm behaviour is similar to an ant colony in searching the food source or new location for the nest.
3. **The dynamic complex structure approach for large void crossing**. The proposed complex structure is purposely for a large size of swarm, accordingly in the order of thousands. Thus, the approach is only how to traverse any size of voids without the risk of retreating back to the initial zone. An analogy can be considered in that the swarm behaviour is similar to the whole of an ant colony moving to a new nest location or to get food from the source.
4. **Designing and printing the robot model**. As a structure consists of self-assembled swarm of robots, aspects of robot design have been considered as the base for the simulation, such as the robot shape, the robot features and the robot docking mechanism. It has been resulted in a Cube robot design in which the shape is based on a cube, relies on a number of relatively simple sensors, the movement uses Ackerman steering, but can be modified using differential-drive steering and

the docking mechanism employs grooved pins and a latched engagement mechanism. OctBot (Aziz et al., 2015), another robot design, has been implemented to the physical platform and has also been considered for use. It has an octagonal shape with a relatively simple sensor and moves using differential-drive steering. The docking mechanism uses a ring and capture mechanism. The measured physical aspects of these models are considered and included in the simulation.

1.2 Structure of the Thesis

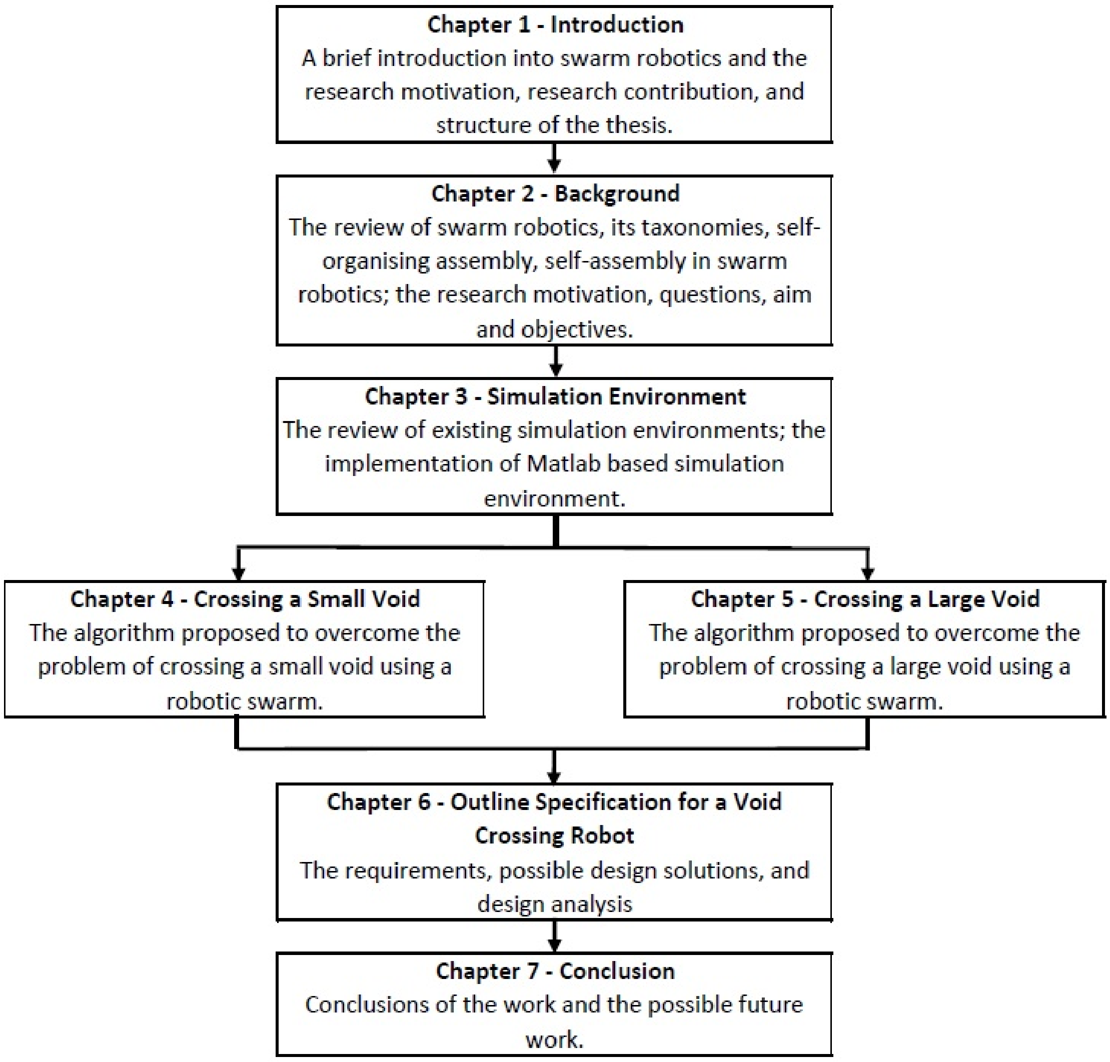

Chapter 1 - Introduction
A brief introduction into swarm robotics and the research motivation, research contribution, and structure of the thesis.
Chapter 2 - Background
The review of swarm robotics, its taxonomies, self-organising assembly, self-assembly in swarm robotics; the research motivation, questions, aim and objectives.
Chapter 3 - Simulation Environment
The review of existing simulation environments; the implementation of Matlab based simulation environment.
Chapter 4 - Crossing a Small Void
The algorithm proposed to overcome the problem of crossing a small void using a robotic swarm.
Chapter 5 - Crossing a Large Void
The algorithm proposed to overcome the problem of crossing a large void using a robotic swarm.
Chapter 6 - Outline Specification for a Void Crossing Robot
The requirements, possible design solutions, and design analysis
Chapter 7 - Conclusion
Conclusions of the work and the possible future work.

Figure 1.2: The structure of the thesis with the relation between chapters. The small void crossing is up to five robot wide and the large void crossing is six to ten robot wide, but the strategy can be applied for up to thirty robot wide.

The remainder of the thesis is organised as follows:

In Chapter 2, a brief explanation and review are delivered. In reviewing the definition and characteristics of swarm robotics, it has been considered that swarm robots in the proposed methods have a number of key features according to the swarm robot characteristics. The reviewed taxonomies become the base to deliver the proposed approaches as well as the process of designing the methods. The following section discusses the self-organising assembly, including definitions, general types of self-assembly and the existing self-assembly in swarm robotics. As self-assembled structures need to consider the practical system, this is reviewed in the next section. The previous sections delivered to the research motivation and research questions, aim and objectives.

As the proposed methods are required to be simulated, a swarm robots simulation environment has been developed. After comparing a number of existing simulation environments, in the next section of Chapter 3 the features of the simulation environment are discussed to show other users how to use and modify it. This implementation provides the first research contribution.

All void crossing approaches consist of two main tasks: distributed individual level tasks and collective behaviour tasks, which are explorations and building the dynamic structures. To implement the real building self-assembled structure approaches, there are two main prior steps developing the simulation. The first step is according to the main purpose of the approach, how a swarm is able to traverse any size of void, which is implemented in case A. Because this approach will not be necessary with the scalability, then algorithm in case B has been developed. Here any size of swarm is able to traverse a void, but it has *a priori* knowledge of the void size. The final approach is to combine those two steps into the simple line structure to produce advanced algorithm in cases C and D, where any size of swarm can traverse a range of voids. A limited number of robots are added into the swarm according to the fixed initial robot positions to find the frequency seed robot number, the accomplishment time, and the success task. The swarm size can be changed into a scalable number by setting the random initial robot positions to find the accomplishment time, the success task and the relation between the swarm size and the accomplishment time. All these approaches and results are discussed in Chapter 4 and provide the second research contribution.

To bridge a large void by significantly increased numbers of swarm robots (hundreds or thousands), a complex structure is proposed. A further analysis has been done as the base on which to build a dynamic complex structure. Using this approach, swarm robots are able to traverse a void that is separated by two zones. Including results obtained, this approach is discussed in Chapter 5 and provides the third research contribution.

To strengthen the approach, robot models have been designed and printed. The design considers the swarm robot characteristics, manoeuvrability and the connection mechanism between two robots, at the front-rear sides of them, as discussed in Chapter 6. It also discusses the robot model to form the complex structure. It is the developed version of the robot model for the simple line structure by adding the connection mechanism for right-left and top-down sides, so that the structure can be constructed in 3D form. This work provides the fourth contribution.

Chapter 7 presents the conclusions to the work together with a discussion on possible future work that will lead to real-time implementation of certain aspects of this work.

An IEEE paper (Syafitri et al., 2015) has been published on IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2015. The conference was held on 28th September to 3rd October 2015 in Hamburg, Germany, following the presented poster. ISBN: 978-1-4799-9993-4. And an IEEE paper (Syafitri et al., 2017) has been accepted on the 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC2017) that was held on 5th to 8th October 2017 in Banff, Canada, but regretly it was not published because of an issue.

Chapter 2

Background

This chapter considers the aspects of swarm robotics, including its definition and characteristics, to determine key features of the proposed methods of swarm robots. A number of taxonomies are discussed to classify to which method or behaviour the swarm robots belong. The next review concerns self-organising assembly, discussing the definitions, the general types of self-assembly and existing self-assembly in swarm robotic systems. Because self-assembly in robotics depends on the robot body shape and connection mechanism, the practical systems section covers the review of those problems. All of the reviews are considered to formulate the research motivation, research questions and objectives.

2.1 Swarm Robotics

Bonabeau et al. (1999) stated that an insect in a social insect colony seems to have the capability to work on its own. Furthermore, if such insects are collected in a colony, they become an organised group. It seems that each activity in a social insect colony does not require any supervision, because all individuals work independently. When working collaboratively, they perform remarkable feats, although they are not capable of doing the same thing when they work individually. This scheme has inspired swarm robotics research to involve a large number of robots that work collaboratively.

Swarm robotics is a type of multi-robot system, but it has specific characteristics. To differentiate between swarm robots and other multi-robot systems, the definition and attributes of swarm robotics need to be considered in detail as some key features have to be strictly applied.

2.1.1 Definition and Characteristics

Swarm robotics has been defined as "*a novel approach to the coordination of large numbers of robots*" and as "*the study of how large number of relatively simple physically embodied agents can be designed such that a desired collective behaviour emerges from the local interactions among agents and between the agents and the environment*" (Şahin, 2005).

The characteristics of a robotic swarm that are found in swarms of social insects have been identified by Beni (2005) and Şahin (2005). There are several aspects that need to be clearly defined to differentiate swarm robotics from other multi-robot research areas:

1. A swarm robotic system should be autonomous and mobile, hence a robotic swarm does not have an external controller.
2. A swarm robotic system is a multi-robot system. However, a multi-robot system can work under centralised control (Winfield, 2012). Therefore, swarm robotic systems have decentralised control. It was also noted that the same set of built-in behaviours have to be embedded into swarm robots and there is no hierarchy in a swarm, which means a leader cannot be predetermined.
3. The size of a robot swarm should be defined clearly as the realistic size of a social insect swarm is $10^2 - 10^6$. Şahin (2005) mentioned that most research would accept group sizes of ten to twenty robots as swarms, but the system aims for scalability. However, in most swarm robotic research, only a small number of research projects achieve sizes of ten to twenty; most of them use fewer than ten robots. Therefore, the most acceptable term for swarm size is the aim for scalability.
4. The robotic swarms consist of relatively simple mobile robots. The word simple means robots have difficulty in completing a complex task alone, but would find it easier if they cooperated with one another. Robots should also have limited sensing and communication capability.
5. The swarm of robots should consist of relatively homogeneous robots. It can consist of a few different types of robot, but the number of robots in each type should be large.
6. Emergence is one of the important things that differentiates swarm robotic systems from other collective robotic systems. This is the set of behaviour patterns that emerge from the interactions between robots in a swarm (Pfeifer and Bongard, 2007).

Following the characteristics of swarm robotics, the swarm should have the following key features: autonomous mobility, limited sensing and communication abilities, simplicity, decentralised control or coordination mechanisms, homogeneity and scalability.

Before discussing the taxonomy of swarm robotics in the next section, there are three advantages of a robotic system that implements the swarm concept (Martinoli, 2001):

1. Robustness: the system should be able to continue to operate when there are failures in the individuals or disturbances in the environment.
2. Flexibility: the system should be able to adapt to the dynamics of the number of robots and the change of tasks.
3. Scalability: the system should be able to operate with different numbers of robots, from a few to hundreds or thousands.

2.1.2 Taxonomy

There are a number of surveys in swarm robotic research to classify and analyse the approach in swarm robotic research. Beni (2005) has grouped the research of swarm robotics into: i) pattern formation; ii) investigation of individual robots in a swarm to carry out a specific task; iii) investigation of collective behaviour through provision of a task that should be operated on cooperatively by swarm robots; and iv) generic robotic tasks. However, the classification is only focused on the tasks that should be carried out by swarm robotic systems. Bayindir and Şahin (2007) have divided swarm robotic research into five main axes, as can be seen in Figure 2.1. The proposed taxonomy is quite complete, although some collective behaviour problems are not included in the review.

A classification of applications in swarm robotics based on mode of communication has been presented by Penders (2007). However, further examples of existing research projects in swarm robotics related to the communication mode have not been delivered. Mohan and Ponnambalam (2009) have proposed nine research domains of swarm robotic research: biological inspiration, communication, control approach, mapping and localisation, object transportation and manipulation, reconfigurable robotics, motion coordination, learning, and task allocation. The challenges faced in each domain with related research are presented and reviewed. Barca and Sekercioglu (2013) have classified swarm robotic research based on research challenges that are faced, such as communication and control scheme, self-organising behaviours, scalability and robustness, connected formations, path planning and obstacle avoidance functions, object transport and manipulation functions, and the energy problem. Although each problem is clearly reviewed, it is quite difficult to define the taxonomy. Brambilla et al. (2013) have proposed two taxonomies to classify swarm robotic research into the used methods and the collective behaviours, as can be seen in Figure 2.2. It is a complete review and future directions of swarm robotic research are suggested so that other researchers can directly investigate the suggested topics.

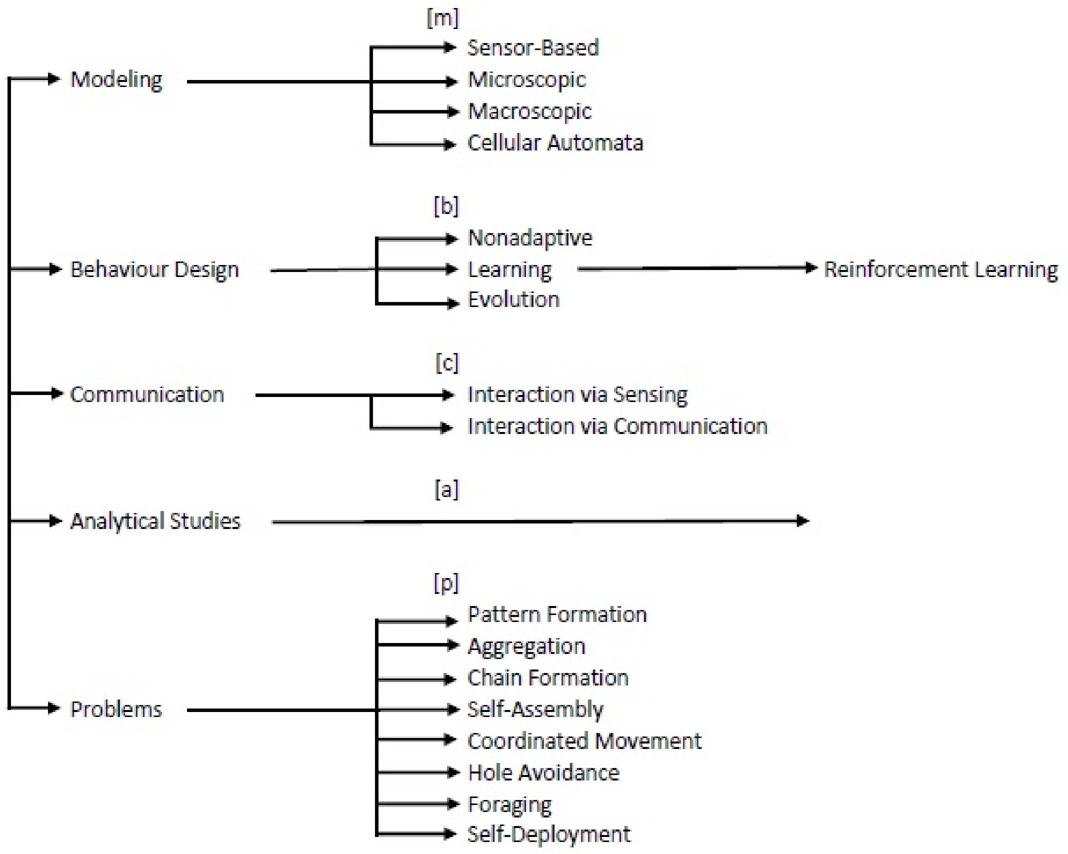


Figure 2.1: A taxonomy of swarm robotic research proposed by Bayindir and Şahin (2007). It consists of five axes: (m) modelling, (b) behaviour design, (c) communication, (a) analytical studies and (p) problems that need to be solved by robot swarms.

Navarro and Matia (2013) have classified swarm robotic research into two different experimental types: i) experimental platforms which include robotic platforms and simulators, although not all existing platforms are discussed, and ii) experimental basic behaviours and tasks, although most of the discussed types can be classified into complex behaviours and tasks. They are aggregation, dispersion, pattern formation, collective movement, task allocation, source search, collective transport of objects, and collective mapping. Patil et al. (2013) have classified the hardware of swarm robots based on modularity and architecture, although not all of the existing swarm robotic systems have been reviewed. The modularity of swarm robots is classified into three groups: i) lattice-based; ii) chain-based; and iii) mobile-based robot architectures. The hardware architecture has been reviewed based on robot's subsystems, such as: i) sensor platform; ii) actuation and locomotion platform; iii) controller and communication module platform; and iv) power option. Based on two classifications, both advantages and disadvantages of swarm robots can be reviewed.

Based on the aforementioned categorisations and comparing the completeness of the

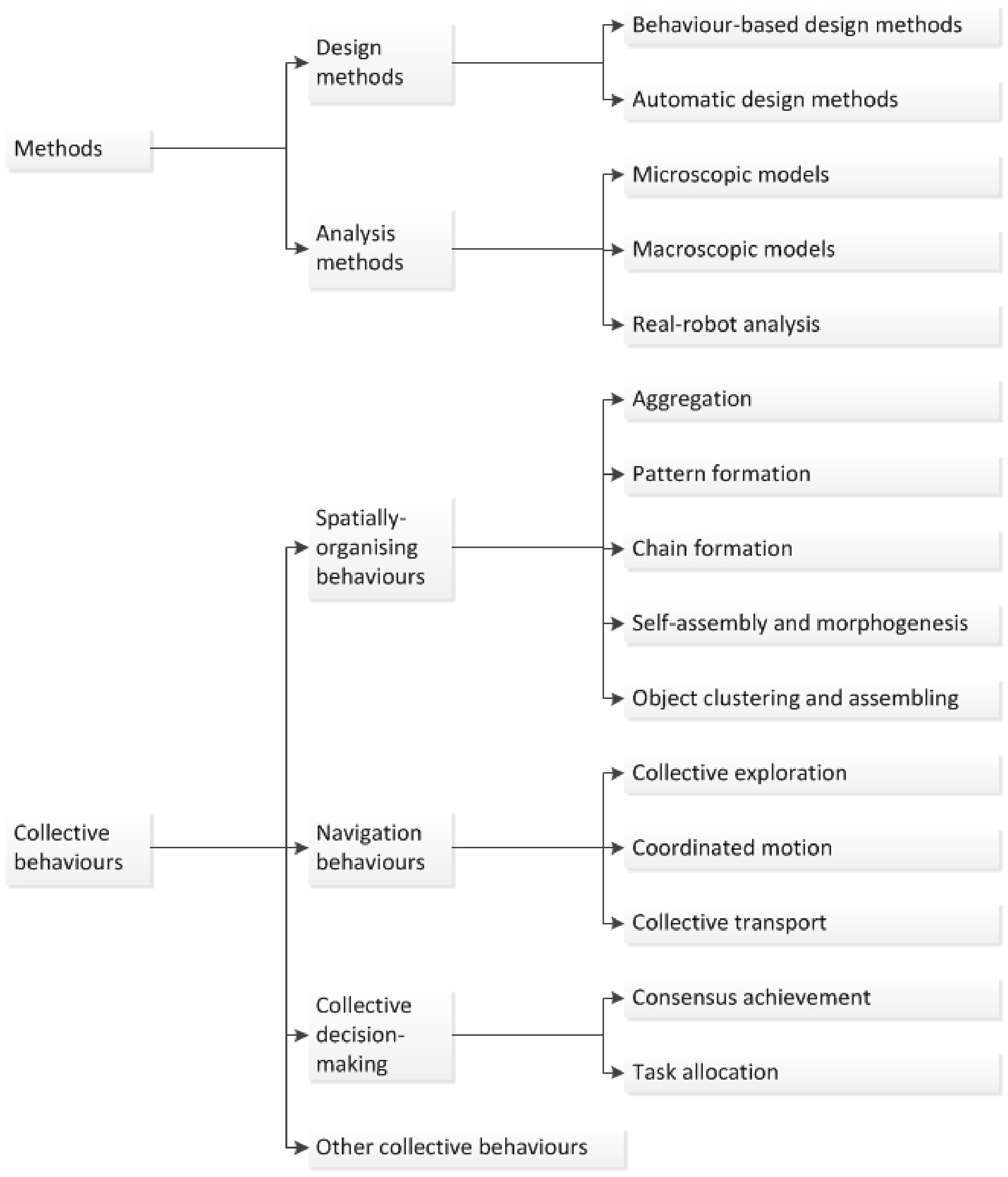


Figure 2.2: Two taxonomies proposed to classify swarm robotic research (Brambilla et al., 2013). The first taxonomy is classified based on applied methods, and the second taxonomy is classified based on implemented collective behaviours that should be carried out by robot swarms.

review to cover all existing swarm robot research, this research adopts the taxonomy proposed by Brambilla et al. (2013). Within the two taxonomies, the applied method is behaviour-based design because it takes inspiration from the behaviour of an ant colony bridging a void to reach inaccessible areas. Applying the property-driven design proposed by Brambilla et al. (2012), as can be seen in Figure 2.3, only three phases have been taken into designing the swarm robot model because the approach has not yet been implemented on real robots. It has been considered that deciding the metrics to use belongs to the validate properties phase, which are the void distance to cover and the accomplishment time related to the swarm size. Improving the model phase has been done by considering the structure, robot model and void and arena sizes, repeatedly, while simulations have been implemented.

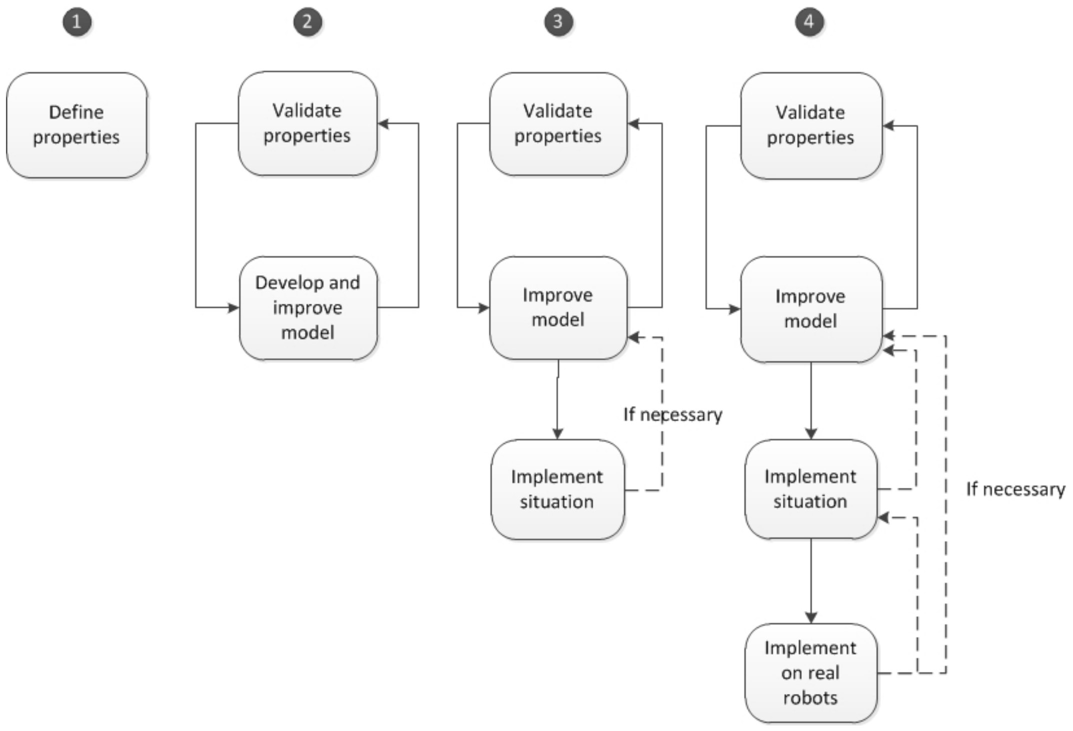


Figure 2.3: Different phases of property-driven design proposed by Brambilla et al. (2012).

The collective behaviours carried out by robots in the swarm are self-assembly and morphogenesis, which relate to the spatially-organising behaviour. It is proposed that the simulated swarm robots connect physically to each other by following a particular pattern to self-assemble forming a structure to finish the main task. A further discussion of self-assembly in general and in robotics follows in the next section.

2.2 Self-Organising Assembly

The swarm robotic system in this research is mostly related to the self-assembly and morphogenesis behaviour, although the main task is to find the target in the environment.

Applying this behaviour is widely based on several reasons that are in line with swarm robotics such as: simplifies control, eases tasks, increases evolvability, affords new behaviours, supports self-exploration, creates new research questions and supports scalability (Bongard, 2014). This behaviour also functions for formation, replication, growth, reconfiguration, sub-module repair and task-oriented growth (Groβ and Dorigo, 2008); and brings benefits of increasing interaction between members of the swarm, keeping specific distances, increasing stability or pulling power while tackling specific tasks such as all-terrain navigation, hole avoidance, void crossing or passing a narrow bridge (Brambilla et al., 2013). While most existing approaches proposed making the organism or the structure first before it wanders in the environment, it will brings an unadaptability when environment changes or is unknown. As there is a related behaviour of an ant colony and the parallel process of exploration and self-assembly is a must for adaptability, the void crossing problem is taken in this research as the parallel process can be seen clearly. Thus, the definition of self-assembly has to be discussed.

2.2.1 Definitions

As defined by Groβ and Dorigo (2008), self-assembly is "*a process by which preexisting components (separate or distinct parts of a disordered structure) autonomously organise into patterns or structures without human intervention*". Although the use of self-assembly is well applied in swarm robotics, it should be noted that, in many applications, the human (operator) is still involved to arrange the pattern or structure when the form of tasks change, even though they were using the term self-assembly. In Brambilla et al. (2013), self-assembly is defined as the process of connecting robots physically without stating human intervention. As stated by Sendova-Franks and Franks (1998), the definitions of self-assembly and self-organisation in biology are two different terms and they can be combined into a self-organising assembly whereby it is possible to make a system effectively simpler, more robust and more secure against failure. In self-assembly, it involves the process of all individuals that are physically connected to each other to take part in the final structure. In self-organisation, it entails a mechanism to build the structure at a higher level through the use of simple rules, without giving an explicit rule as to how an exact individual connects to the exact other one.

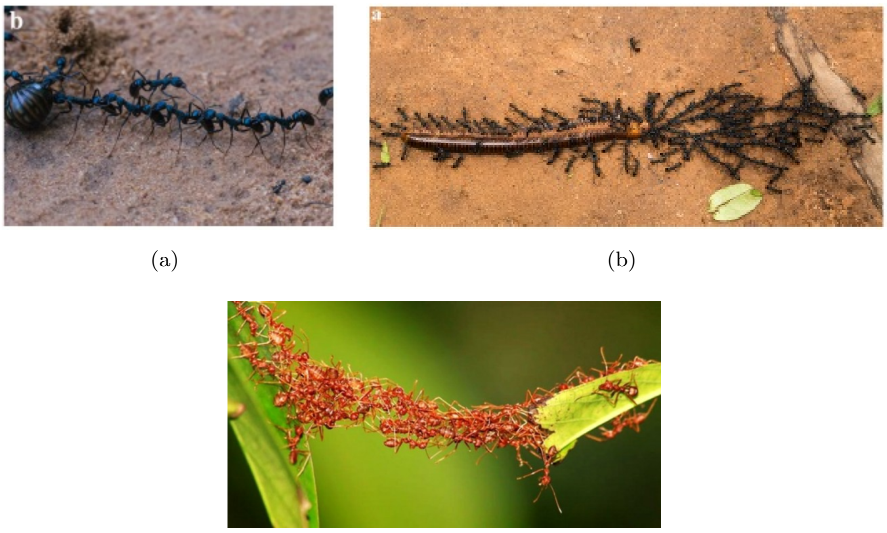
The proposed methods in this research are considered to involve both self-assembly and self-organisation, whereby the simulated swarm robots can assemble to each other forming a structure through a simple rule. Considering the swarm robotics and self-organisation characteristics, all robots have the same opportunity to place themselves in the structure and to connect to any other robots regardless of their array. Thus, considering the physical implementation, if a robot or even a few fail in the middle of an operation, the swarm can still accomplish the operation through the structure formation and the environment exploration.

Because self-assembly is performed both in biology and robotics, the discussion of self-assembly types based on those major fields is delivered in the following sub-section. Swarm robots in this research are viewed from both fields.

As an addition, when constructing a structure (e.g. a raft during a flood), a colony of *Formica selysi* perform specialisation and have memory of their position in the structure, but factors of the position consistency and the reason for this behaviour are still unknown (Avril et al., 2016). If this behaviour is applied into the swarm robotic system performing self-assembly to traverse more than one void, it affords the advantage of the efficiency of time when the distance of the following voids is close to the previous ones. However, if the distance between voids is far, it not only affects the longer accomplishment time, because swarm robots wander randomly in the environment and the structure needs to wait for particular robots to assemble, but it also needs more memory space in each robot to keep memorising its position. This approach also results in a slow simulation, as it will be shown in a case in Chapter 4. Considering the disadvantages and the case where there is only one void, it has been decided that, after traversing the void, swarm robots erase their memory of their position in the structure.

2.2.2 Types of Self-Assembly

A classification of structures formed through self-assembly was proposed by Anderson et al. (2002) in which seventeen structures are categorised into three main structures: i) chain (1D), where the formed structure shapes are only long single lines; ii) mesh (2D), where insects assemble forming structures in many types of irregular polygons; iii) or cluster (3D), where the formed structure shapes are irregular prisms. Examples of 1D to 3D social insect structures can be seen in Figure 2.4. Based on this classification, the simple line structure of the proposed methods can be categorised as a dynamic chain – pulling chain (1D), where a single robot cannot reach the separated opposite zone and, therefore, the swarm will form a moving chain and finish the task by bridging the void. As for the large void structure, the structure is categorised as a bridge structure (3D), but dynamic in a non-random arrangement. Both structures are considered as complex structures as individuals explore the environment randomly, but they are well-arranged in the structure. The final structures are also dynamic and adaptable to the change of the swarm and the void sizes.



(a) (b) (c)

Figure 2.4: Social insect structures (ant colony) that shape in: (a) 1D structure, a pulling chain to bring a small prey to the nest, the formed structure is only a line; and (b) 2D structure, a pulling chain to take a larger prey to the nest, where two or more lines connect (Peeters and Greef, 2015); and (c) 3D structure, a bridge to connect two leaves, where ants construct the structure in 3D (Alonso, 2015).

Self-assembly classification in robotics can be related to modular robotics as the connection mechanism is usually equipped into the robot modules in order to assemble each other, thereby forming a complex structure. Swarm robots performing self-assembly can be viewed as a robot module, but, individually, they can act on their own. Furthermore, the combination of modular robotics and self-assembly might bring an advantage of exploring strategy for morphologies and emergent functionalities (Pfeifer et al., 2007).

A classification of artificial systems performing self-assembly at the macroscopic scale has been proposed by Groβ and Dorigo (2008), in which the existing modular robotic systems were classified based on four categories: i) physical and electrical design characteristics, swarm robots in this research are homogeneous, using both passive and active connection mechanisms, using local interaction between them, and designed to be equipped by self-propelled components; ii) outcome and analysis of self-assembly experimentation, where the complexity of the situation applied in this research is the unknown size of the void to be traversed by swarm robots, although they are simulated in a 2D environment; iii) process control, whereby, in this research, all robots can be the seed and they have no *a priori* knowledge of their relative positions; and iv) functionality, where the self-assembly approach in this research has a purpose, growth, where robots form growing entities to bridge a void unnavigable for individuals.

A classification based on geometry of the systems was proposed by Gilpin and Rus (2010) where modular robot systems are categorised into chain, lattice, truss, and variable shape systems. Swarm robots in this research occupy a chain-type system as the robot arrangement in the structure can be single- or multibranched linkages.

As swarm robots in this research have been classified into a number of taxonomies, the comparison of the approach to others has to be reviewed, as discussed in the following sub-section.

2.2.3 Self-Assembly in Swarm Robotics

Kuyucu et al. (2013) have taken the case when a robot swarm operates in a structured environment and has to explore it. The environment is split into several sections using low and high obstacles (walls). As the main task is to explore the environment, the walls existence limits the swarm movement and coverage. To overcome this problem, a method using pheromones has been proposed to allow the fifteen robots to assemble and form a snake-like structure to climb the obstacles. The structure could climb and pass low obstacles, but it was not able to climb high obstacles, as can be seen in Figure 2.5. This problem was solved by forming three snake-like structures at the same time to avoid the assembly breaks. However, if the pheromone to form the assembly was too high, the structure could not disassemble into swarm robots again. From the published results, it was concluded that this approach does not have the adaptivity to adjust with the height change of the walls.

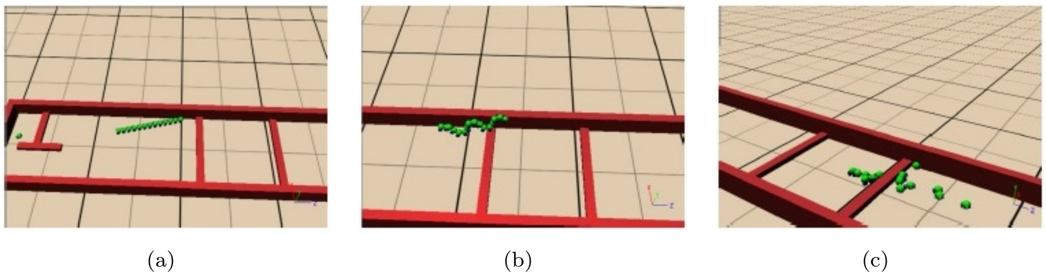


Figure 2.5: A snakebot (Kuyucu et al., 2013). (a) Swarm robots assembled to a snake form when they met an obstacle. (b) The snakebot could climb a low obstacle, but (c) it was stuck when it had to pass a high obstacle.

In the Swarm-Bots project led by IRIDIA, Belgium, collective behaviour of swarm robotics, especially self-assembly, has been being investigated using s-bots. Self-assembly has been proposed to solve the following problems: all-terrain navigation, hole avoidance, hill climbing, rescue operations and crossing a void (Mondada et al., 2004). In Trianni and Dorigo (2006), the hole avoidance case has been taken by locating a square of four s-bots in the centre of the square arena. The assembled robots had to avoid all arena edges. The development of how swarm robots self-assemble using evolutionary methods, what morphologies can be formed and the rules of self-assembly have been investigated in Ampatzis et al. (2009), O'Grady et al. (2009), Trianni and Nolfi (2009) and O'Grady et al. (2012). In Trianni and Nolfi (2010), the proposed self-assembly method has been used in a case study for bridging or avoiding a void. Nine s-bots in a square formation were able to cross a 10cm wide void, but they avoided the 20cm void because it was too wide to traverse. In O'Grady et al. (2010), several formations of both manual and self-assembled s-bots were used to manage the structure for climbing a hill and crossing a void in which they had to reach broken s-bots which needed to be rescued. In O'Grady et al. (2012), two case studies were considered to consider the ability of self-assembled s-bots to overcome the emergent problem in the previous research. In the first case, four robots crossed a 22cm void by forming a line structure, where the width of the void was chosen because it was passable by four robots in a line morphology, as seen in Figure 2.6(a). Then, in the second case, robots had to cross two pipes. To keep balance while traversing the pipes, two robots had to move side-by-side and grip each other, as seen in Figure 2.6(b). In Trianni et al. (2014), the developed evolutionary method was implemented so that self-assembled s-bots avoided the hole. From the proposed methods and the obtained results, the height and steepness of the hill to be climbed and the width of the void are still conditioned and, additionally, the structure cannot self-adapt to the change in dimensions.

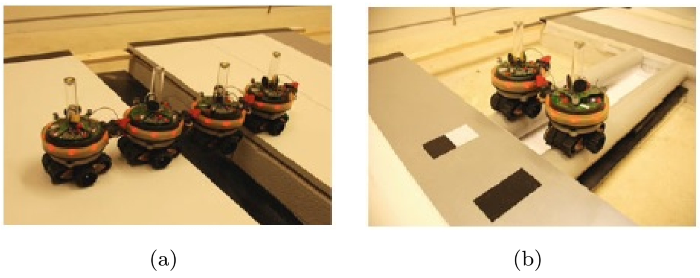
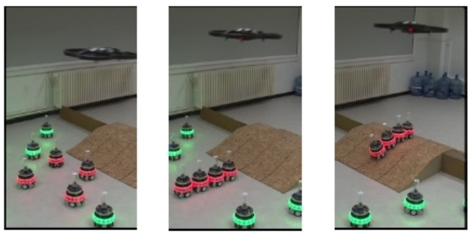


Figure 2.6: Swarm-Bots project led by IRIDIA, where (a) swarm robots had to cross a 22cm void by forming a line structure and (b) they had to cross two pipes by gripping each other to maintain the stability (O'Grady et al., 2012).

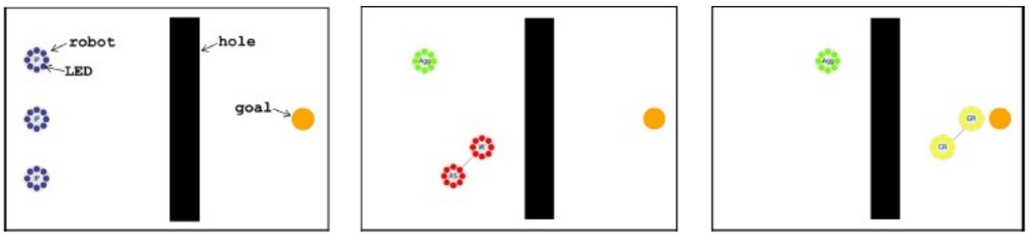
To increase the ability of s-bots and the efficiency of energy, the IRIDIA research group has developed marXbots (or foot-bots) (Bonani et al., 2010b). In Mathews et al. (2011), a method to change the morphologies of assembled marXbots while they were moving was considered. In the proposed future case study, they have to cross a gully in the real world. In Mathews et al. (2012, 2015), a swarm of marXbots was combined with several aerial robots to climb a hill, as shown in Figure 2.7. Exploiting the communication method, the aerial robots observed the environment and informed the marXbots if there was a hill in front of the swarm. Then, they gave instructions to several of the closest marXbots to self-assemble, forming a structure which would move to the targeted location. In these approaches, the change of height, steepness and width of hill and void can be adapted by the swarm through morphology changes and robot heterogeneity. However, with these proposed methods, the robotic systems were more characteristic of multi-robot systems rather than swarm robot systems. It applied hierarchy in the system. The employed aerial robots were giving instructions about which ground swarm robots to assemble and what type of morphology to be formed. Furthermore, the robotic swarm system needs more space, especially the height of the environment, as the aerial robots need the space to fly. This design will have problems in a destroyed environment where there is a possibility that the height of the environment is low due to ruined objects.



(a) (b) (c)

Figure 2.7: A swarm of marXbots (Mathews et al., 2012) had to explore the environment. (a) The aerial robot informed marXbots that there was a hill in front of them. (b) The aerial robot gave an instruction to the closest marXbots to assemble, forming a structure based on the hill's characteristic. (c) The assembled marXbots climbed the hill.

In a similar case study proposed by Bruni et al. (2015), simulated s-bots were used to test the method of assembly readiness using LEDs. Three s-bots were deployed in one side of an arena which was split in two by a long square hole. A target was positioned in the opposite area. When a robot met the void, it enabled itself to be grabbed by other free robots. When it was grabbed, the assembled s-bots moved towards the target by traversing the hole, as seen in Figure 2.8. However, with only two assembled s-bots, this approach did not consider the physical aspects of s-bots that can cause the structure to fall into the hole.



(a) (b) (c)

Figure 2.8: A swarm of s-bots had to cross a void, but the physical aspects, such as the relation between the void width, the strength of connection and the number of assembled robots, were not considered (Bruni et al., 2015). (a) Three s-bots had to reach the goal that was at the opposite zone. (b) A robot detected the void and grabbed another robot in order to cross the void. (c) Two assembled s-bots successfully found the target.

In the Symbrion and Replicator projects, the challenge to deploy a number of swarm robots in an environment for a long time has been accepted. To answer this challenge, the consortium has proposed an approach predicated on the idea that the heterogeneous swarm robots can self-assemble, forming snake-like and X-like organisms to find and reach power sources behind the walls, as can be seen in Figure 2.9 (Kornienko et al., 2007; Levi et al., 2014). The hardware, software, mechanical and morphology challenges have been discussed, reviewed and answered in Kernbach et al. (2008, 2009, 2010, 2011). The proposed organisms, their movement and behaviour, and further morphology have been proposed in Winkler and Worn (2009), Saska et al. (2011), Liu and Winfield (2012), Meister and Gutenkunst (2012), Meister et al. (2013), and Vonásek et al. (2015), where the varied terrain has been added into the environment. The organisms have to explore the environment and sometimes they have to traverse a void. In general, the built organisms were formed on the planar surface, then lifted themselves up and started moving within the environment to reach the target location. Therefore, like other approaches, the organisms were built first and subsequently the organisms in their whole state moved to the targeted location, then they dispersed into swarm robots after the organisms recharged their power. Moreover, once the organisms have moved, they cannot add other robots to the organism.

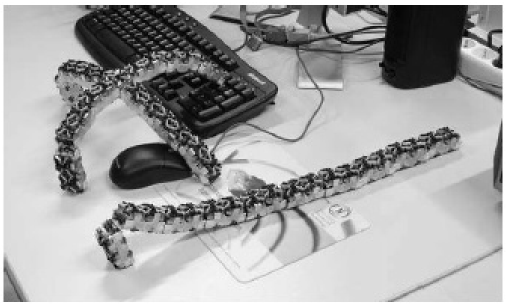


Figure 2.9: The concept of an assembled heterogeneous swarm robots from the Symbrion and Replicator projects to find power sources at other locations (Kornienko et al., 2007).

Sambots are a combination of swarm robotics and modular robotics, to self-assemble and form several structures, such as snake-like and X-like shapes, to perform varied tasks such as autonomous docking, quadruped configuration and caterpillar locomotion (Wei et al., 2010) and climb high obstacles (Li et al., 2015, 2016), and which use an evolutionary approach to describe the topology of the structures. This enabled swarm robots to self-assemble before they climbed the obstacles and passed them, and robots could adapt with the change of the obstacle height. However, in the physical experiments, the seed was selected by a human (operator) and the target configuration pre-designed so that it was not fully autonomous and not applying the full characteristics of swarm robot systems. Furthermore, before climbing the obstacle, the structure was built and, when the structure could not climb the obstacle, they had to retreat and disassemble to later form another morphology.

All the aforementioned proposed approaches explore the methods for how swarm robots self-assemble, forming a structure before they climb walls or cross a void or avoid a hole.

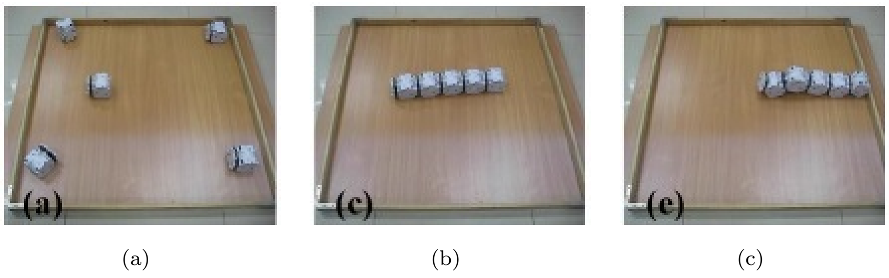


Figure 2.10: Sambots formed a snake-like structure to climb high obstacles (Li et al., 2016). (a) Five Sambots were placed on the arena and a selected Sambot became the seed. (b) Sambots formed a predefined snake configuration. (c) The snake configuration climbed the high obstacle.

The structure will self-disassemble after it has reached the opposite zone. However, the requirement for adaptivity with respect to changes in the obstacle height or the void width is still a problem. Furthermore, this approach will not meet the aim of scalability for swarm robotics. If the number of robots changes, there is a requirement that the system has to learn how to change the morphology to build a bigger or longer structure, or to make several structures, as well as changing the parameters of the system.

However, there is an approach which takes inspiration from an ant colony to build a pyramid structure (Cucu et al., 2015). Using a specific design of mobile robots and implementing a specific algorithm, they can climb over other robots. The approach is still being developed, but it is a promising method for swarm robots to build a structure to reach a higher inaccessible area, whereby, in this research, the aim of building a structure is to reach another location in the horizontal.

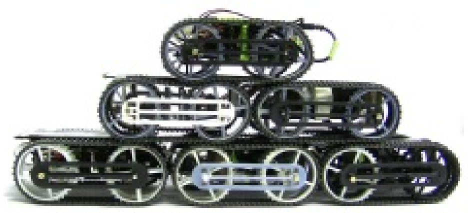


Figure 2.11: A pyramid structure formed by assembled mobile robots inspired by an ant colony to reach a higher location (Cucu et al., 2015). Robots can climb over other robots and construct a pyramid structure, whereby the above layers contain fewer robots than the ones below, so as to maintain the structural stability.

As self-assembly in robotics cannot be separated from the connection mechanism, it is important to review a number of connection mechanism designs to decide which mechanism to be used for a swarm robot system in this research, to support self-assembly and building a structure to traverse a void. The review is discussed in the following section alongside the discussion of communication systems in swarm robotics.

To summarise types of structures in this section, Table 2.1 shows the classification of structures of existing swarm robot projects.

Table 2.1: Types of structure.

	Types of structure		
Robots	**Chain (1D)**	**Mesh (2D)**	**Cluster (3D)**
Snakebots (Kuyucu et al., 2013)	✓		
s-bots (Mondada et al., 2004; Trianni and Dorigo, 2006; Ampatzis et al., 2009; O'Grady et al., 2009; Trianni and Nolfi, 2009; O'Grady et al., 2010; Trianni and Nolfi, 2010; O'Grady et al., 2012; Trianni et al., 2014; Bruni et al., 2015)	✓	✓	
marXbots (foot-bots) (Bonani et al., 2010b; Mathews et al., 2011, 2012, 2015)	✓	✓	
Symbrion and Replicator (Kornienko et al., 2007; Kernbach et al., 2008, 2009; Winkler and Worn, 2009; Kernbach et al., 2010, 2011; Saska et al., 2011; Liu and Winfield, 2012; Meister and Gutenkunst, 2012; Meister et al., 2013; Levi et al., 2014; Vonásek et al., 2015)	✓	✓	
Sambots (Wei et al., 2010; Li et al., 2015, 2016)	✓	✓	
(Cucu et al., 2015)			✓

2.3 Current Robotic Systems for Self-Assembly

2.3.1 Connection Mechanisms

There are several modular robots and swarm robots which use a connection system based on magnets, such as Smart Pebble (Gilpin and Rus, 2010), M-Blocks (Romanishin et al., 2013), and extended e-pucks (Murray et al., 2013). Using an electropermanent (EP) magnet and a 12mm cube, Smart Pebble can connect to other modules by arranging its EP magnetic poles. No module has a battery, so it depends on having to share power with other connected modules in the structure. By pre-programming the desired shape of a 2D-structure and locating a fixed root module, Smart Pebble modules were placed on a vibration table. By exploiting the vibration, they moved and self-assembled to form the structure.

Because EP magnets require high instantaneous current to actuate, Romanishin et al. (2013) have proposed M-Blocks, a 50mm cube module that utilises twenty-four bonding magnets along its edges to connect it and a flywheel to initiate the motion. Through use of a proposed pivoting cube model (PCM), M-Blocks move to other locations by pivoting their edges and forming a rigid 3D-structure. Although the bonding magnets mechanism makes the connection rigid and precise, M-Block modules still receive commands from a central computer to locomote using radio communication. Moreover, a connection system using a magnetic field has a disadvantage. It will be a burden when it works in an area with magnetic or ferromagnetic material (Kutzer et al., 2010). Both Smart Pebbles and M-Blocks can be seen in Figure 2.12.

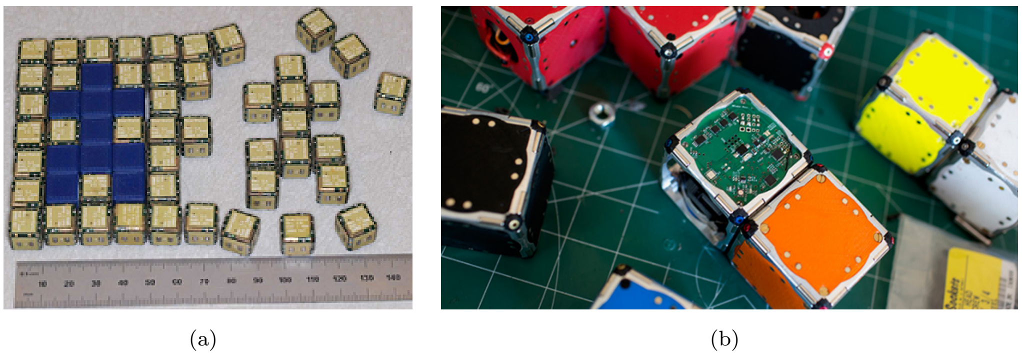


Figure 2.12: Cube modular robots developed by a research group in MIT: (a) Smart Pebbles (Guizzo, 2012) and (b) M-Blocks (MIT, 2013).

A gripper system has been used in s-bots and the connection is quite simple in that it grips other s-bots along the gripped ring (Mondada et al., 2004). However, the connection is not rigid and it causes the structure of assembled robots to be deflected, resulting in them falling when they have to cross a void. There is also a possibility that the gripper slips when it holds another robot. An improvement of the s-bot gripper has been developed in marXbots (Bonani et al., 2010b). It employs a similar concept of gripped ring, but the gripper shape consists of a row of three pins. If the middle pin is moving up in the gripped ring, then the two other pins are moving down, and vice versa. With this mechanism, a marXbot is able to connect to other marXbots in almost 360° and a marXbot can be gripped by two or more marXbots. However, the marXbot connection is not rigid if the structure requires a long row of assembled marXbots. Both s-bot and marXbot can be seen in Figure 2.13.

A connection mechanism using hooks has been proposed by Jorgensen et al. (2004) for ATRON and Wei et al. (2010) for Sambots. The hook principle in ATRONs is quite complex and requires more space for implementation. An ATRON has two half spheres with a rotation mechanism. Each half sphere consists of two male and two female connectors. Each male connector has three hooks where the hook in the middle points is in the opposite direction to the other two hooks, and each female connector has three rods. The use of two hooks in Sambots strengthens the connection between

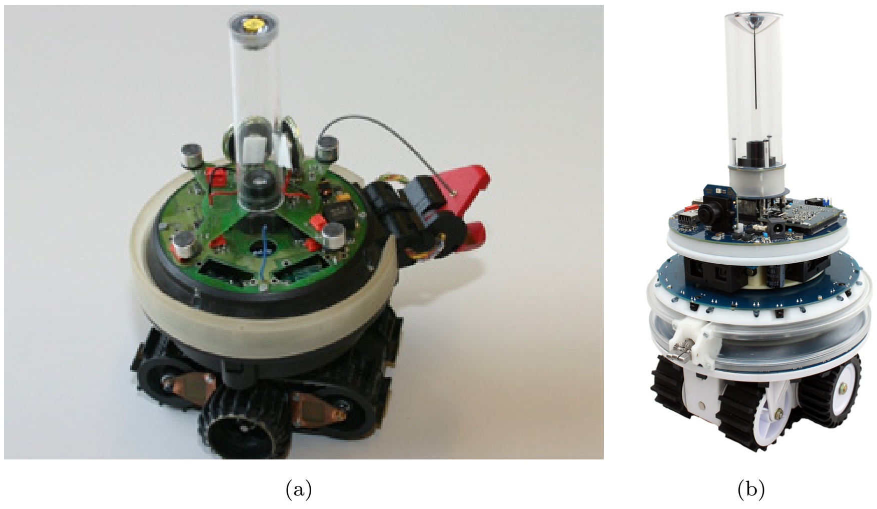
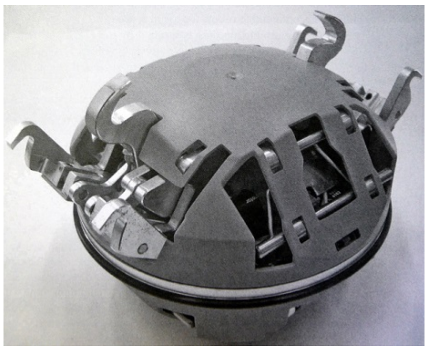


Figure 2.13: (a) s-bot (EXPO21XX, 2014) and (b) marXbot (Bonani et al., 2010a) developed by EPFL, Switzerland.

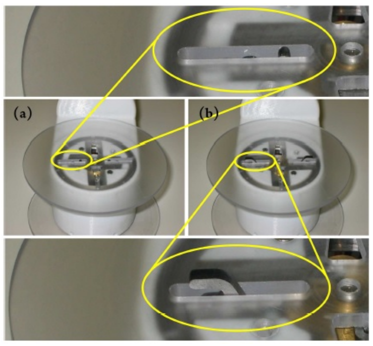
robots, but the curved-pyramid shape of the hooks enables Sambots to dock in a range of misalignments. It creates the possibility of a non-rigid connection. Wolfe et al. (2012) have proposed a mechanism that combines hooks and wheels for an M^3Express platform. Each platform has three wheels that serves as a connector. Although this mechanism reduces the required energy of motors to hook and to move, it still employs magnetic attraction. Moreover, the performance of the docking mechanism is still controlled manually by a human operator and needs further testing. All three robotic systems using hook mechanisms can be seen in Figure 2.14

Unsal et al. (2001) have designed a simple twist-and-lock mechanism for self-reconfigurable modular cube robots called I-Cubes, as can be seen in Figure 2.15. A cross-shaped connector that enters the cross-shaped hole will twist and will be locked by a plate. However, I-Cubes employ two different types of module, links to provide the motion and cubes to provide the stable structure and the position of sensors. This configuration affects the larger robot dimensions and weights and there is a special rule to connect a module to others.

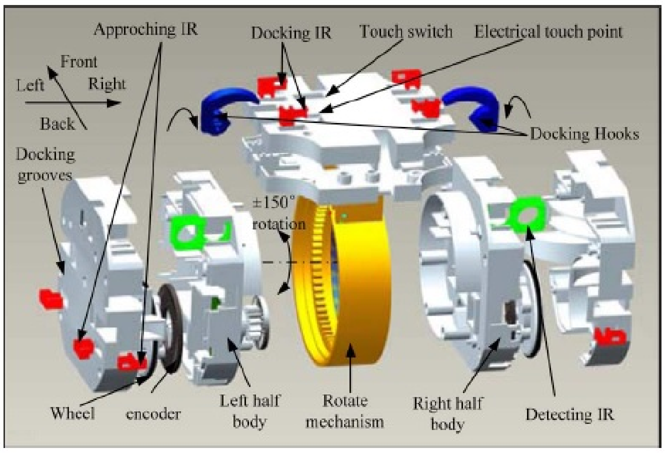
Castano et al. (2000), Yim et al. (2003) and Liedke and Worn (2011) have proposed docking systems using grooved pins and engagement latches, as can be seen in Figure 2.16. The main principle of this docking system is to drive grooved pins into corresponding holes. After the pins have fully entered the holes, a disk with latches will lock the pins. This mechanism is not simple because both grooved pins and engagement latches need to be designed carefully; it consumes more power for the motor driving the disk and holding the pins during connection, and it requires a special communication system to ensure the precise connection. However, the connection in this mechanism is strong and rigid.



(a)

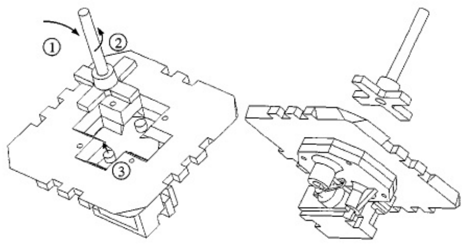


(b)

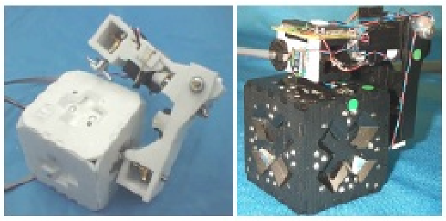


(c)

Figure 2.14: Hooks connection mechanism proposed by (a) Jorgensen et al. (2004) for ATRON, (b) Kutzer et al. (2010), and (c) Wei et al. (2010) for Sambots.



(a)

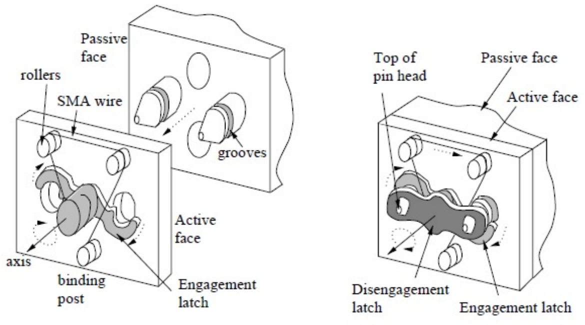


(b)

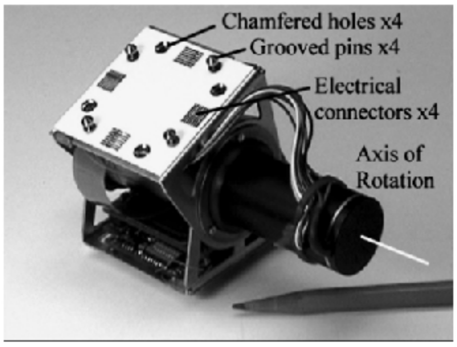
Figure 2.15: Twist-and-lock mechanism (a) used in I-Cubes (Unsal et al., 2001), and (b) previous and latest version of I-Cube modules (Unsal and Khosla, 2001).

2.3.2 Communication Systems

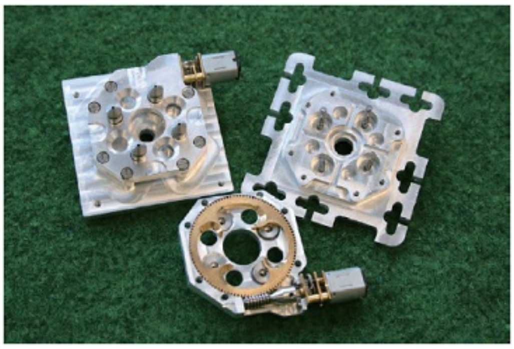
A range of communication systems has been used in swarm robotics, using a one-to-one communication system or one-to-many communication system, whereby the most of the systems are using the second type.



(a)



(b)



(c)

Figure 2.16: Docking systems using grooved pins and engagement latches for (a) CONRO (Castano et al., 2000), (b) Polybot (Yim et al., 2003), and (c) CoBoLD (Liedke and Worn, 2011).

The thousand-scale swarm robots Kilobot uses IR transmitter and receiver embedded in each robot to communicate with each other (Rubenstein et al., 2014). Besides acting as the proximity sensors to measure each other's distance, with these devices a Kilobot can communicate with a few of its neighbours.

e-pucks (Mondada et al., 2009) only occupy Bluetooth for a broadcast communication. A similar communication system is proposed in Kuyucu et al. (2013) by occupying WiFi or ZigBee as radio transmitter/receiver, but the device is proposed to be used as one-to-one and one-to-many communication systems. Different devices are used for s-bots (O'Grady et al., 2012), wherein RGB LEDs and an omnidirectional camera are used, but they can be set for either communication systems.

A number of swarm robotic systems are using several types of communication devices. In Symbrion and Replicator projects (Winkler and Worn, 2009), IR devices are used for one-to-one communication system in individual mode. And, for broadcast communication, ZigBee is equipped in each individual. In organism mode, robots communicate by using an internal bus system. marXbots (Bonani et al., 2010b), a development of s-bots, use RFID, Bluetooth and WiFi for broadcasting. To communicate one-to-one, RGB LEDs, CMOS camera and omnidirectinal camera are used. In Sambot (Wei et al., 2011), ZigBee is used in the swarm state for broadcast communication. In the robotic structure state, CAN bus communication is used.

A one-to-one communication system is particular with the characteristics of limited communication capability. However, a swarm robot needs to communicate to a few of the other robots for a particular purpose, such as calling the closest free robot from a few to become assembled to the structure. Comparing the range of the working distance where, in future work robots should be deployed in the real environment, it is better to use devices that are purposely used as broadcast communication systems. Considering the minimum energy consumption, it is better to use ZigBee as the radio transmitter/receiver.

To summarise connection mechanisms and communication systems of swarm and modular robots in this section, Table 2.2 and Table 2.3 show the classification of existing swarm and modular robot projects. By reviewing a number of aspects of swarm robotics, self-assembly and practical consideration that have motivated this research, the purpose of this research and a number of research objectives are discussed in the following section.

Table 2.2: Types of connection mechanism.

	Types of connection mechanism				
Robots	**Magnet**	**Gripper**	**Hooks**	**Twist-and-lock**	**Pins-and-latches**
Smart Pebble (Gilpin and Rus, 2010)	✓				
M-Blocks (Romanishin et al., 2013)	✓				
extended e-pucks (Murray et al., 2013)	✓				
s-bots (Mondada et al., 2004)		✓			
marXbots (Bonani et al., 2010b)		✓			
ATRON (Jorgensen et al., 2004)			✓		
Sambots (Wei et al., 2010)			✓		
M^3Express (Wolfe et al., 2012)			✓		
I-Cubes (Unsal et al., 2001)				✓	
CONRO (Castano et al., 2000)					✓
Polybot (Yim et al., 2003)					✓
CoBoLD (Liedke and Worn, 2011)					✓

Table 2.3: Types of communication system.

	Types of communication system		
Robots	**One-to-one**	**One-to-many**	**Organism mode**
Kilobot (Rubenstein et al., 2014)		IR transmitter and receiver	
e-pucks (Mondada et al., 2009)		Bluetooth	
Snakebots (Kuyucu et al., 2013)	WiFi and Zigbee	WiFi and Zigbee	
s-bots (O'Grady et al., 2012)	RGB LEDs and omnidirectional camera	RGB LEDs and omnidirectional camera	
Symbrion and Replicator (Winkler and Worn, 2009)	IR transmitter and receiver	ZigBee	Internal bus system
marXbots (Bonani et al., 2010b)	RGB LEDs, CMOS camera, omnidirectional camera	RFID, Bluetooth, WiFi	
Sambot (Wei et al., 2011)		ZigBee	CAN bus

2.4 Research Motivation and Objectives

Motivated by the requirement of robot adaptivity to operate in an unstructured and unknown environment, a number of research groups have proposed approaches using swarm robotics and self-assembly. However, the proposed common strategies have a limitation in adapting with the change of environment, such as the approach to bridge two separated zones by a changeable obstacle size. Taking inspiration from nature, there is an example that is remarkably achieved by an ant colony to reach inaccessible areas if they work collaboratively, but cannot perform the achievement if they work individually.

This research proposes an approach to overcoming the problem of when a robotic swarm meets an obstacle that needs to be traversed. From consideration of the literature, there is a problem in applying the common strategy of assembling robots initially at the initial zone to form a structure, then stepping over or crossing an obstacle, then disassembling the structure into swarm robots. If the size of the obstacle changes, or if the number of swarm robots changes, there is a probability that the structure cannot traverse to another area. Furthermore, if the number of robots in a swarm is very large, it will take a long time to assemble all robots.

Therefore, the aim of this research is to develop a strategy for self-assembly, crossing an obstacle and self-disassembling to break free or escape from an isolated area by forming a temporary structure. It is expected that the robot swarm is able to traverse over any size of obstacle to reach the opposite area. Furthermore, if the number of robots is not large enough to carry out the task, it is expected that the swarm robots will not fall into the obstacle, but that they will go back to the initial zone.

To achieve the aim of the research, a number of investigations have to be conducted, such as: i) the investigation of the shape of the structure that is formed by self-assembled swarm robots that can work for any size of obstacle and swarm; ii) the investigation of parameters influencing the connection and the structure, such as the structural stability, weight, deflection and span, related to the obstacle width; iii) the further investigation of swarm robot connection mechanism in physical conditions and then implementing the obtained factors into the simulation; iv) the investigation of swarm robot adaptivity in an unstructured and unknown environment using self-assembly with only a simple rule according to the characteristics of the swarm robotic system.

According to characteristics of swarm robots, the swarm in this approach should have key features as follows:

1. Autonomous mobility: robots in the swarm autonomously move in the environment with a differential-drive steering mechanism.
2. Limited sensing and communication abilities: in the proposed approaches, robots in the structure only call the closest free robots to join the structure.
3. Simplicity: no individuals can cross the opposite zone alone, but, when they co-operate in building a structure, swarm robots can achieve the other area.
4. Decentralised control and coordination mechanism: all robots have the same controller in each robot when they wander exploring the environment, have the same opportunity to find the void or the target and have the same opportunity to join the structure; in the latter approaches, they have no knowledge of either the swarm size or the void size.
5. Homogeneity: considering the cost and one of the swarm robot' characteristics, all of the robots in the proposed methods are homogeneous.
6. Scalability: after being developed, the swarm aims at scalability so that any number of robots can be added into the swarm.

And considering advantages of a robotic system that implements the swarm concept, the swarm in this approach should achieve following advantages:

1. Robustness. Loss of a few individuals do not make the system stop working because they have the same controller in each robot; compared to other approaches that

apply a leader scheme (a robot is chosen as the seed) that meets a non-robust method when the leader cannot operate in the middle of the task, all robots in the proposed approaches have the same opportunity to be the seed or to find the target, so that it is not a problem if a small number of robots meet a condition of failure to operate.

2. Flexibility. Any number of robots can be added into the swarm and it performs well according to the given main task, finding the target. The swarm is also able to cope with the different size of voids, the subtask of which is to traverse the void.
3. Scalability. In the proposed approaches, any size of robotics swarm can perform well; if the number of robots is not enough, robots can retreat to the initial area; if the number is far enough, they can traverse any size of void through a temporary self-assembled dynamic structure.

2.5 Research Questions

As social insects can perform adaptivity through a such remarkable mechanism, taking inspiration from an ant colony, there is a hypothesis that swarm robots are adaptable within reaching a previously inaccessible area by performing a self-organising assembly. By cooperatively constructing a structure, it is hypothesised that swarm robots can bridge the void to the previously inaccessible area. This hypothesis generates a number of research questions, as following:

1. How are simple self-assembly rules defined to achieve the adaptivity of swarm robots, where any number of robots can be added into the swarm and robots have to achieve the target that is located on the separate area, and those areas are separated by a void with an unknown size.
2. How to choose the shape of the structure constructed by assembled swarm robots that is strong and stable for any size of voids so that swarm robots are able to finish the main task, such as find the target or the food source.
3. How to design the swarm robots and the strong connection mechanism of swarm robots to support the strong and stable structure?
4. How are metrics defined to measure the performance of the self-organising assembly behaviour of the robotic swarm?

2.6 Discussion

Considering the characteristics of swarm robotics and its advantages, robots in this research have a number of key features, such as autonomous mobility, limited sensing and communication abilities, simplicity, decentralised control and coordination mechanism, homogeneity, and scalability. According to the taxonomy adopted, the requirement of self-organising assembly and the related practical systems, such as modular robotics and self-assembly in swarm robotics, there are a number of problems to be solved in achieving the purpose of the research, such as the connection mechanism, the simulation environment, the challenge of applying swarm robotic characteristics, and the formed final structure.

A rigid connection mechanism is required to build a structure that is able to bridge two separated zones, supporting a structure that has a long span to reach another area, and the structure will not deflect and bring the assembled swarm robots to fall over the void. Besides the connection mechanism, the robot shape also influences the formed structure aside from the simplicity of movement aspect in the environment. Based on these aspects, robots for self-assembly have been designed, discussion of which will be presented in Chapter 6. The physical aspects obtained from the models will be used for the simulation of self-organising assembly behaviour.

Considering key features of swarm robotics and the requirement of self-organising assembly behaviour, a specific simulation environment need to be built, as the existing simulation environments do not support those aspects. The important requirements for a simulation environment are the following: it has to support swarm robotic systems specifically, not only general multi-robot systems; users can define the robot shape, so the shape is not a default one; users can define the dimension of the environment and its features, such as obstacles; and it supports the simplicity of displaying robots in 2D, as this type will not require a high-definition graphic card.

The proposed self-organising assembly method is an approach to achieve an adaptivity of swarm robots in an unstructured and unknown environment. Therefore, swarm robots should have an ability of traversing any size of obstacle regardless of their number, as scalability is an important feature to be achieved. Considering the simplicity and the accomplishment time, the proposed method will apply the method to build a dynamic structure as inspired by an ant colony; however, as structures for each small crossing and large crossing cases are different, the deflection related to the span should be considered carefully.

Chapter 3

Simulation Environment

This chapter presents the requirements of the simple simulation environment for self-organising assembly in swarm robotics as well as the review of existing simulation environments. Motivated by the review, it has been decided that there is a need to develop a new simulation environment, the features of which are also discussed in this chapter.

3.1 Simulation Environment Requirements

After reviewing self-assembly in swarm robotics in particular the void crossing problem, it was noted that the simulation environment for self-organising assembly using swarm robotics requires the following aspects:

1. An easy installation process of a simulation environment; so that no inherent problems will affect the simulations as well as the simplicity to run a simulation.
2. As the system is to simulate a robot swarm, it is important that the simulation environment can simulate a multi-robot system and it should be easy to define any number of robots working in the environment.
3. Although a swarm robotic system is a type of multi-robot system, it applies decentralised control rather than the centralised control other multi-robot systems use. Therefore, this control should be able to implement in the simulation environment.
4. It is important to define robot geometries and features easily, as there is a requirement of specific geometries and features to perform the self-organising assembly behaviour. Furthermore, it is better if there is no requirement of other rendering softwares to define both aspects.
5. It is also important to be able to define the environment easily, including its size, its features including any obstacles.

6. As the main behaviour in this research is self-assembly, it is important that the assembly process can be simulated although in a simple way, such as aggregation at a very short distance.

7. A 2D and non-HD simulation environment is required, so that there is no requirement of using a specific HD graphic card, and it can reduce the real time of simulation.

8. It is better to easily record the video of simulation as a proof of the approach.

Based on the aforementioned requirements, a number of simulation environments are reviewed in the next section to decide which simulator is to be used or implemented.

3.2 Comparison of Readily Available Simulation Environments

Comparing a number of simulation environments, anyKode (ANYKODE, 2014) has advantages with the capability to simulate a number of environments and to accommodate a wide range of geometries. However, it requires a HD graphics card and the simulation appearance is in 3D and too complex. Using ARGoS (Pinciroli, 2014), users can modify the environment, add a number of obstacles, add up to 4,000 robots in real time (Bhaumik, 2012) and simulate up to 10,000 simple wheeled robots 40% faster than in real time (Pinciroli et al., 2011). However, the simulation started lagging when it simulated 278 foot-bots. Users also have difficulty in defining or making another robot model, as the provided model is only a foot-bot, although features on it can be extended. Users have to use other software platforms to define or make a model, such as Blender, Collada or X3D. Claytronics (Claytronics, 2014) can simulate over than twenty million spherical shape modules without lagging (Ashley-Rollman et al., 2011), but it seems the simulator does not support swarm robotics. It also requires a specific rendering simulator (DPRSim) for observing real-time performance. delta3d (delta3d, 2014), a simulator whose main purpose is for military applications (Carpin et al., 2007), can also be used for robotic applications. However, it results in a large file for a video frame in a recorded simulation. EZPhysics (EZPhysics, 2014) is not considered useful for a simulation environment as it requires a high resolution graphics card to simulate a 3D environment.

A test using Gazebo (Open-Source-Robotics-Foundation, 2014) has been carried out employing 35 simple robot models whose features are differential-drive wheels, castors, grippers and a Hokuyo laser scanner. However, to carry out a simulation it is ineffective, as users have to copy and paste robots and set the motion of each robot. For further use, it requires a specific software platform, such as Blender or SketchUp. In a test using Jasmine-Breve (Jasmine, 2005; Klein, 2008), a simulation environment for swarm robotics, has been carried out, but it ran slowly and sometimes the simulation froze. lpzrobots (LpzRobots, 2013) has been developed for research into self-organised behaviour. Unfortunately, some parts are under license and it requires a high resolution graphics card. PI estimator is used to control the motion of each robot in the Swarm Robot Project Simulator (Swarm-Robot-Project, 2014), a Matlab-based simulation environment. The simulator only supports e-puck robot models, but users can define other robot models. The collision avoidance algorithm did not work well and some parameters had to be changed for successful simulation using more than twenty robots. But it is apparent that a similar approach can be implemented for the simulation environment used.

MORSE (OpenRobots, 2014) is not considered useful, as it requires a high resolution graphics card and specific software to support the simulation, as well as Microsoft Robotics Developer Studio (Microsoft, 2012), which even requires specific hardware. The installation and the running simulation in Player/Stage (Player, 2010) were difficult, since there were many bugs. To run the simulation, users have to open three terminals and windows and the default robot and sensor positions cannot be changed. SeSam (SeSAm, 2012) can simulate several tens of thousands of simple agents. However, the installation crashed with the new Java version and it took a very long time to run. USARSim (USARSim, 2013; Carpin et al., 2007) requires a specific framework and some parts are under license. v-rep (Robotics, 2014) is quite promising, as it can work on any operating system and graphics card; it does not need specific requirements and can be written in a number of programming languages. However, it seems that the simulation is only for one or two robots and not for a swarm. Webots (Cyberbotics, 2014) requires a high resolution graphics card because it runs in a 3D environment. The simulation runs slowly, particularly when users build their own robot models. When it is closed with an unfinished program or model, it crashes during subsequent launches.

Among the reviewed simulation environments, ARGoS has been applied to simulate the void crossing problem for marXbots (Mathews et al., 2010). As noted in Section 2.2.3, a number of simulation environments have been developed to specifically simulate the void crossing problem such as Swarmbot3D based on Vortex for s-bots (Mondada et al., 2004), a specialised simulation environment using Maude for s-bots (Bruni et al., 2015), and a specialised simulation environment for Symbrion and Replicator project (Vonásek et al., 2015). Other simulation environments have been used to simulate obstacle climbing or all terrain navigation including Webots for snakebots (Kuyucu et al., 2013) and Microsoft Robotics Developer Studio combined with real-time physics engine PhysX for Sambots (Li et al., 2015).

Another simulation environments, Claytronics, seems able to simulate void crossing problem as long as the seed is predefined and there is no requirement to simulate the real physical connection mechanism as robots used are atom-likes to form any form of structures (Ashley-Rollman et al., 2011). Swarm Robot Project Simulator can simulate the random movement of robots in the swarm and if a modification is taken, it is a promising simulator to illustrate the void crossing problem. A number of simulation environments such as Gazebo, Jasmine-Breve, Player/Stage and SeSam were claimed to be able simulating a large number of agents but when the tests were conducted, they did not produce the required environment.

The summary of existing simulation environment is presented in Table 3. Following reviewing and testing a number of simulation environments, evaluating how simulations were conducted for a swarm robotic system, and seeing the performances and further requirements for void crossing simulation, it was decided to develop a bespoke system for the following reasons:

1. There is a need to deploy a scalable number of swarm robots, from two to thousands, by only defining the robot numbers with the same features and characteristics. There is no need for users to put robots on the arena one-by-one and define each one's features and characteristics.
2. Users should easily be able to choose the shape of swarm robots. Basic features such as wheels and sensors are provided.
3. As it has to simulate scalable swarm robots, the simulation environment can only be implemented in 2D to prevent slow computation or crash if it is implemented in 3D.
4. It is better to implement the simulation environment by considering the wide range of users where not all of them can be provided by a high resolution graphic card.
5. A specific hardware should not be added in order to run a simulation environment.
6. The simulation should be recordable as a video to see the performance of the swarm, from the exploration at the initial zone, the construction and the exploration at the target zone.
7. The simulation time metrics used should be recordable to see the performance of the swarm in exploring and bridging the arena as the arena dimension is fixed.
8. All robots have a fixed initial ID number, which is useful to see each performance in accomplishing the task, whether they become the seed or find the target, and also their position in the structure.
9. The position of the seed should be also recordable to prove that all robots have no *a priori* knowledge of the gap position.

Considering that the vehicle model using Ackermann steering geometry (MilanHurban, 2013) and the Swarm Robot Project Simulator (Swarm-Robot-Project, 2014) that run on Matlab, the approach can be simulated by combining both simulators. The first simulator defines how to create a specific 2D robot model and sets its features, including: robot dimensions; setting the wheel dimensions, turning angle and their velocity; updating data for each sampling time; and setting the distance sensor to objects and to the void. However, because the purpose of the first simulator is to simulate only a single vehicle or robot, a similar approach for the second simulator is required. It provides an approach for creating the work of multi-robot systems, including: the setting of the number of robots; simple obstacle avoidance (although robots still collide with each other); setting of random initial position; and setting of updating data for each sampling time for multi-robot systems. Both simulators run in Matlab. This means users are using scripts to write codes and a number of functions provided in Matlab can be called upon to support the main script without making or predefining them. Separating the main script into a number of subsections is also allowed in Matlab to avoid lagging, both in the simulation and writing codes. According to the characteristics of swarm robotics, the simulation will apply a limited sensing ability. The distance sensor for each robot only depends on the distance calculation between a robot and objects. Hence, misalignment between two or more robots as collision avoidance between dynamic obstacles can be ignored.

3.3 The Swarm Robots Simulation Environment using Matlab

The simulation environment can simulate homogeneous swarm robots, mainly with a wandering feature. If users need to add other tasks such as self-assembly, object assembling, fault detection, collective transport, path optimisation, task allocation and task partitioning, they can add other loops in the main loop. Users can also change the setting of global variables, sampling time, the swarm size, target and obstacles positions, arena and main obstacle size, and robot size, as long as the 2D model in rectangular shape. It will later add options of other polygons.

Taking the scripts used for of simple line structure as the whole flowchart can be seen in Figure 3.1, for the first setting, it is necessary for users to determine which variables are global and are used in other scripts or functions. As the main scripts can be divided into a number of scripts, global variables are important to be used as well as in functions to be called. The time is set at 0s and the time sampling is set at 0.01s, but can be changed as users need, Algorithm 1. If users require a smooth robot movement, time sampling must be less than 0.01s. Choosing 0.01s as the time sampling was based on the running of initial experiment. If the time sampling was larger than 0.01s, it didn't fully show the robot's movement. If the time sampling was less than 0.01s, the robot's movement was smooth and represented the real world robot's movement, but the simulation run longer.

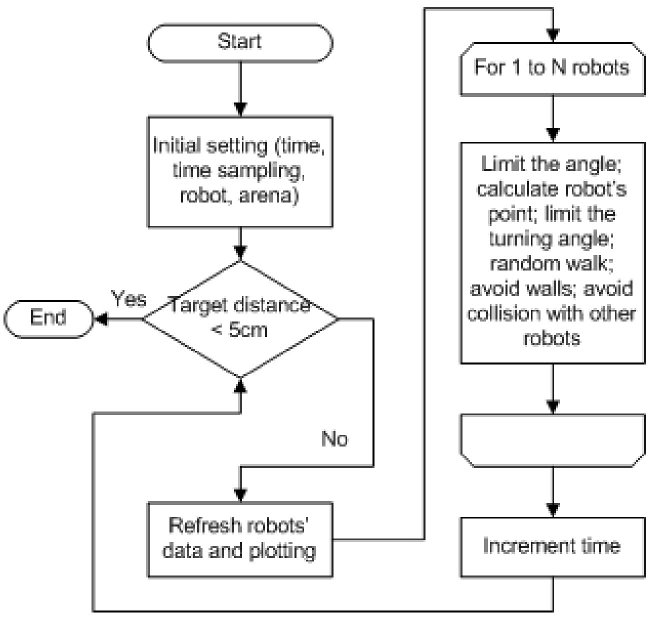

Start
Initial setting (time, time sampling, robot, arena)
Target distance < 5cm
Yes
End
No
Refresh robots' data and plotting
For 1 to N robots
Limit the angle; calculate robot's point; limit the turning angle; random walk; avoid walls; avoid collision with other robots
Increment time

Figure 3.1: Flowchart for an algorithm for a swarm of robots exploring an environment.

Algorithm 1 The setting of initial time and time sampling

```
global N keepLooping modeStart SearchStart

%% time setting
T = 0; % (s)
dt = 0.01; % (s)
```

The following setting is the swarm size. Users can put any number of robots, but subsequently they have to choose between fixed starting positions or random starting positions. If users choose fixed starting positions, they subsequently have to input the centre position of each robot. After the swarm size setting, there is setting of robot parameters such as the half length of the robot, half width of the robot, the distance between wheels, wheel radius, wheel width, maximum velocity of robots (13.89 ms^{-1} is suggested as the safe maximum velocity to minimise collisions with obstacles or walls), turning angle maximum, and initial velocity, initial angle, initial angular velocity, and initial turning angle, Algorithm 2.

If users require the target to measure the simulation accomplishment, they can set the centre position of the target. But, if the simulation accomplishment is determined by time limitation, it will have to be set later in the main loop. Simple obstacles can be put on the arena, as robots have to avoid them during wandering. Users only have to set their centre positions. The next setting is arena and the main obstacle (void). Users can set the arena size, and there is the option to use a grid on the arena. If users need the void to be plotted, then they can choose the provided size of the void. For mapping the arena, including its features, robots and wheels shapes, and the movement of robots, users can rely on a number of functions. All the arena settings are shown in Algorithm 3.

In wandering mode, there are two important functions that have to be used, refresh robot data and refresh plot data. Users can increase or decrease the direction of random movement of robots. If they want robots to move in a straight line until the meet obstacles or walls, the setting number is 0, else it is suggested up to 7. If users need to split the main scripts, they can add and call another script under the exploration loop. The main cycle script can be seen in Algorithm 4.

Algorithm 2 Robot parameter initialisation

```
%% robot settings (free mode)
N = 50; % swarm size/number of robots
hlr = 5.0; % half length of the robot (cm)
hwr = 5.0; % half width of the robot (cm)
wwd = 7; % distance between wheels (cm)
wwr = 2.1; % wheel radius (cm)
www = 0.1; % wheel width (cm)
vmax = 13.89; % maximum velocity (13.89 m/s, about 50 km/h, the safe velocity to not
        collide with obstacles or walls)
dthetamax = 90 * pi / 180; % turning angle maximum

for i = 1:N
    robot(i).size.hl = hlr;
    robot(i).size.hw = hwr;
    robot(i).size.wd = wwd;
    robot(i).size.wr = wwr;
    robot(i).size.ww = www;
    robot(i).v = 0; % initial velocity
    robot(i).theta = 0 * pi/180; % initial angle
    robot(i).omega = 0; % initial angular velocity
    robot(i).dtheta = 0 * pi/180; % initial turning angle
    robot(i).dx = 0;
    robot(i).dy = 0;
    robot(i).structure = -1; % the initial state
end

% choose the polygon type (square or octagon) in makeBodyComb_2.m function
% choose between fixed robot positions and distributable robot positions
% fixed robot positions
%robot(1).x = 20; robot(1).y = 150;
%robot(2).x = 20; robot(2).y = 180;
%robot(3).x = 20; robot(3).y = 120;
% distributable robot positions
coordx = randi([10 190],N,1); coordy = randi([10 290],N,1);
```

The running simulation will be plotted on a figure, such as Figure 3.2, where initial robot positions are fixed and obstacles are set on the arena, and Figure 3.3, where initial robot positions are random and there is no obstacle on the arena. Robots start moving at the initial zone at the left hand side of the arena, and they have to find the target at the target zone at the right hand side of the arena. The square black area is the void that has to be crossed by the swarm.

Algorithm 3 Arena parameter inisialisation

```
%% target position setting; you can choose to not using target
target.x = 475; % x position
target.y = 150; % y position

%% obstacle position setting; you can choose to not using obstacles
obstacle.x.a = [150]; obstacle.y.a = [40];
obstacle.x.b = [120]; obstacle.y.b = [250];
obstacle.x.c = [180]; obstacle.y.c = [100];
obstacle.x.d = [400]; obstacle.y.d = [60];
obstacle.x.e = [450]; obstacle.y.e = [210];

%% basic plot
map.figure = figure('color','white'); % setting the arena in white
set(map.figure, 'Position', [100 100 1000 600]); % the plotted figure scaled twice of the arena
    dimension
hold on;
axis on; % in order to make it easier viewing the movement of robots on the arena

% fixed map dimensions
map.w = 500; % the length of the arena (cm)
map.h = 300; % the width of the arena (cm)
map.x = [0 map.w map.w 0 0];
map.y = [0 0 map.h map.h 0];
map.border = area(map.x, map.y, 'facecolor','w');
map.size = 1; % how large the plot around the robot is

% plot the void (vertical void); you can choose to not using the void
map.vertx = [245,245,255,255,245]; % points to set the void width: 1 robot length
map.verty = [0,300,300,0,0];
map.verts = [245 0; 245 300; 255 300; 255 0]; % setting the lines to form the 1-robot void
map.faces = [1 2 3 4];
map.cliff = patch('Faces', map.faces, 'Vertices', map.verts, 'FaceColor', 'k');

% plot the target in blue with 5cm radius
map.target = plot(target.x, target.y, 'o', 'Color', 'b', 'MarkerSize', 5, 'MarkerFaceColor', 'b');

% plot obstacles in red with 5cm radius; optional
%map.obstacle.a = plot(obstacle.x.a, obstacle.y.a, 'd', 'Color', 'r', 'MarkerSize', 5, ...
    'MarkerFaceColor', 'r');
%map.obstacle.b = plot(obstacle.x.b, obstacle.y.b, 'd', 'Color', 'r', 'MarkerSize', 5, ...
    'MarkerFaceColor', 'r');
%map.obstacle.c = plot(obstacle.x.c, obstacle.y.c, 'd', 'Color', 'r', 'MarkerSize', 5, ...
    'MarkerFaceColor', 'r');
%map.obstacle.d = plot(obstacle.x.d, obstacle.y.d, 'd', 'Color', 'r', 'MarkerSize', 5, ...
    'MarkerFaceColor', 'r');
%map.obstacle.e = plot(obstacle.x.e, obstacle.y.e, 'd', 'Color', 'r', 'MarkerSize', 5, ...
    'MarkerFaceColor', 'r');

axis equal; axis tight; grid minor;
set(gca,'Layer','Top');
```

Algorithm 4 The main computation loop

```
count = 0; keepLooping = true; modeStart = N;
%writerObj = VideoWriter('movie.avi'); %open(writerObj); % for video capture
while keepLooping == true
    while SearchStart == true
        T = T + dt; tStart = T;
        [robot, border, cliff, dynObstDistance, target] = refreshBotDataComb(robot, ...
            border, cliff, map, target, dt, N); % refresh data of each robot
        refreshPlotDataComb(robot, map, target, N); % refresh the plot of each robot
        for i = 1:N
            robot(i).theta = wrapToPi(robot(i).theta); % to limit the angle up to 360°
            [robot] = calcPointsComb(robot, i); % calculate the current robot's point
            if robot(i).v > vmax
                robot(i).v = vmax; % limit the maximum robot's velocity
            end % of velocity limitation
            if robot(i).dtheta > dthetamax
                robot(i).dtheta = dthetamax;
            elseif robot(i).dtheta < -dthetamax
                robot(i).dtheta = -dthetamax;
            end % turning angle limitation
            robot(i).v = robot(i).v + robot(i).ddistance/100;
            for r = rand(N,1) % can be changed from 1 to 7
                if r(i) < probl
                    robot(i).dtheta = robot(i).dtheta - 3;
                elseif r(i) < 1-probr
                    robot(i).dtheta = robot(i).dtheta;
                else
                    robot(i).dtheta = robot(i).dtheta + 3;
                end
            end % random turning angle (random walk)
            if (border.distance(i).a < 15 || border.distance(i).b < 15 ...
                    || border.distance(i).c < 15 || border.distance(i).d < 15)
                robot(i).v = -vmax/13.89; robot(i).dtheta = -dthetamax;
                robot(i).v = 2*vmax/13.89;
            end % avoiding collision with borders
            for j = i:N
                if i == j
                    continue
                end
                if dynObstDistance(i,j) < 15
                    robot(i).v = 0; robot(j).v = 0;
                    robot(i).theta = robot(i).theta - dthetamax/18;
                    robot(j).theta = robot(j).theta - dthetamax/18;
                    robot(i).v = vmax; robot(j).v = vmax;
                end
            end % avoiding collision with other robots
            if target.distance(i) < 5
                SearchStart = false; keepLooping = false; fprintf('Mission accomplished.')
                break; %close(writerObj);
            end % if the distance of any robot to the target is less than 5cm, the exploration
                stops
        end
        pause(dt); count = count + 1;
        %F = getframe(); writeVideo(writerObj, F);% for video capture
        %if tStart > 4.99 % option to set simulation stop based on time
            %keepLooping = false; fprintf('Mission failed.')
            %break; %close(writerObj);
        %end % if choosing the exploration runs based on time
    end % search mode at the starting zone
    % add script here for other tasks
end
```

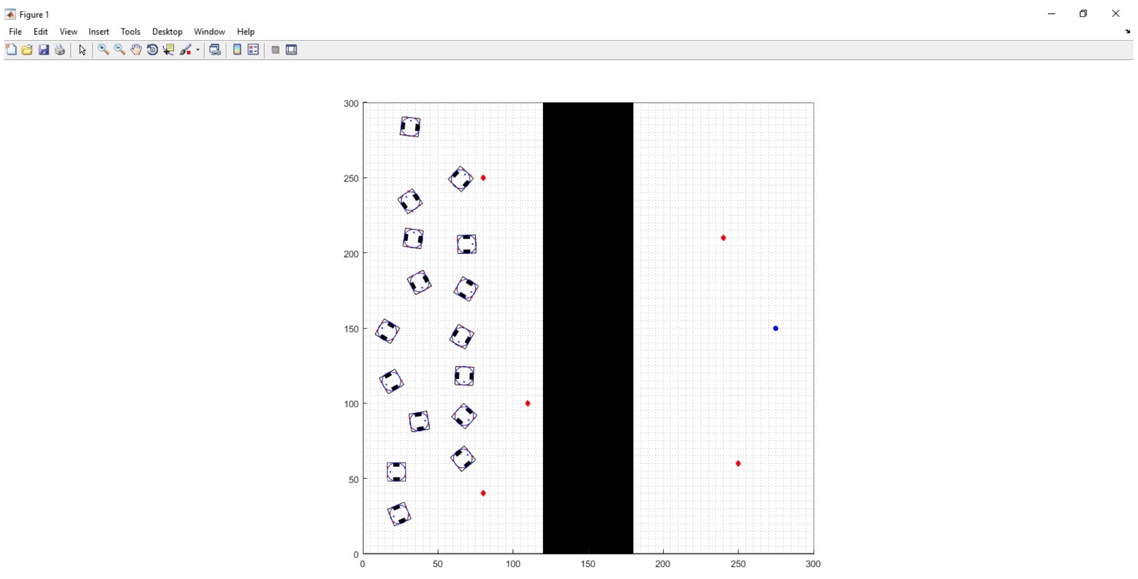


Figure 3.2: Running simulation for fixed initial robot positions and obstacles are set on the arena. The void is shown as the large black area.

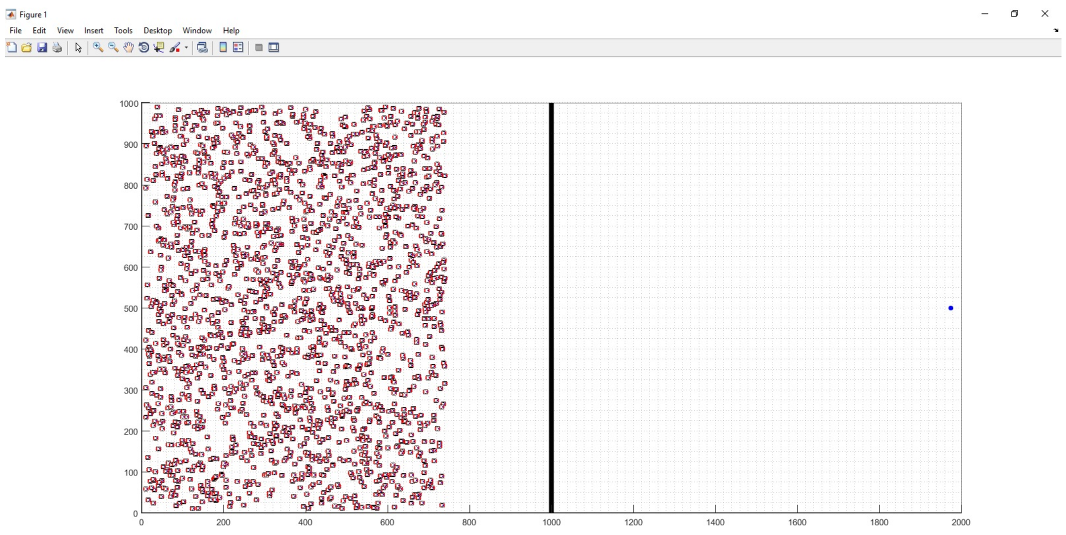


Figure 3.3: Running simulation for initial random robot positions and there is no obstacle set on the arena. The swarm contains five hundred robots. There is a circulating area between the void and the swarm to give space when the void size is increased.

3.4 Conclusion

After reviewing a number of simulation environments, it was decided that the simulation environment is the combination of two simulators running on Matlab. Users can define features of swarm robots and arena used in the simulation. All robots are homogeneous and each robot has the same opportunity to wander, to be the seed in self-assembly mode, and to find the target.

In the following chapters, the simulation environment is applied to simulate a number of self-assembly methods for void crossing. In Chapter 4, there are two prior algorithms for simple line structures. The first algorithm applies a swarm that has *a priori* knowledge of its size, but any robot can initiate the structure and find the target, and the structure can traverse any size of void particular with its length. If the length is not enough, the structure retreats to the initial area and the robots disassemble. After disassembling at the opposite area, all robots are still memorising their position in the structure. Because it is a burden for scalability that robots know their number and position, this method has been changed. The swarm does not know its size, but robots have *a priori* knowledge of the void size. According to the main aim of the research, adaptivity, both methods have been subsequently revised. In this revised method, swarm robots construct a single line structure to traverse any size of void and they do not know the swarm size. This approach is proposed to answer the challenge of traversing any small size of void and the scalability aspect of swarm robotics. Considering large crossing when there is a higher probability of deflection occurring in a single line structure, an approach proposing a dynamic 3D complex structure is discussed by employing thousands of swarm robots, as shown in Chapter 5. Later, the shape of the structure changes according to the size of the void to be traversed, but robots still have no *a priori* knowledge of their number and the void size.

Table 3.1: Robot simulator summary, developed from mentioned papers and sites. The speed variable was from mentioned papers and indicative.

Name	**Running Cost**	**Operating System**	**Graphic Card**	**Programming Languages**	**Special Requirements**	**Speed**
anyKode (ANYKODE, 2014)	Commercial	Windows Linux	NVidia ATI	C/C++ C++ CLI C# J# VB# MatLab Java	-	-
ARGoS (Pinciroli, 2014)	Free open source	Linux Mac OS X	All types	XML	Blender Collada X3D	Fast
Claytronics (Claytronics, 2014)	Under development	Linux	NVidia ATI	Meld LDP	DPR-Sim	-
delta3d (delta3d, 2014)	Free open source	Windows Linux	All types for basic application	C++	Stage Delta3D Editor MS Visual Studio Python PyroRobotics Player/Stage	-
EZPhysics (EZPhysics, 2014)	Free open source	Windows	NVidia ATI	C++ API MatLab	-	-

Continued on next page

Table 3.1 – *Continued from previous page*

Name	**Running Cost**	**Operating System**	**Graphic Card**	**Programming Languages**	**Special Requirements**	**Speed**
Gazebo (Open-Source-Robotics-Foundation, 2014)	Free open source	Linux Mac OS X	All types	SDF (XML)	Blender SketchUp	-
Jasmine-Breve (Jasmine, 2005; Klein, 2008)	Free	Windows Linux Mac OS X	All types	Python steve	-	Slow
LpzRobots (LpzRobots, 2013)	Some parts are under license	Linux Mac OS	NVidia ATI	C++	ODE OpenSceneGraph	-
Swarm Robot Project Simulator (Swarm-Robot-Project, 2014)	Free	Windows Linux Mac OS X	All types	MatLab	Matlab	-
MORSE (OpenRobots, 2014)	Free open source	Linux Mac OS X Windows	NVidia ATI	Command-line Python	Blender	-
MS Robotics Studio (Microsoft, 2012)	Free	Windows	NVidia	C# MS Visual Studio C# VPL	Kinect for Windows SDK V1 32- and 64-bit MS Speech Platform Runtime	-
Player/Stage (Player, 2010)	Free open source	Unix based OS	All types	C++	-	-

Continued on next page

Table 3.1 – *Continued from previous page*

Name	**Running Cost**	**Operating System**	**Graphic Card**	**Programming Languages**	**Special Requirements**	**Speed**
				Java Python XML Tcl		
SeSAm (SeSAm, 2012)	Free	Windows	All types	Java	Java	-
		Linux Mac OS				
USARSim (USARSim, 2013) (Carpin et al., 2007)	Some parts are under license	Windows Linux	MOAST	Unrealscript	Player	Fast
v-rep (Robotics, 2014)	Commercial	Windows Linux Mac OS X ROS	All types	C/C++ Python Java MatLab Octave Urbi	-	-
Webots (Cyberbotics, 2014)	Commercial	Linux Windows Mac OS X	NVidia ATI	C/C++ Python Java MatLab	-	-

Chapter 4

Crossing a Small Void

4.1 Introduction

This chapter explains the developed algorithms for small void crossing, the robot model used and its dimension, the arena used in the simulation, the scenario, the assumptions taken, the algorithms. Following the discussion of the result and the implementation are summarised.

Considering a search and rescue operation, swarm robots might meet obstacles (ruins or voids), where the swarm has to form a structure to overcome the obstacles. Hump or hill crossing and wall climbing were not considered as the study case as the problem can be solved by forming a line or a snake-like structure adjusting with the size of obstacles, Section 2.2.3. The void crossing problem has the challenge of stabilising the structure while adjusting itself with any size of void, keeping its rigidity and avoiding the bending or fracture, and the proposed algorithms in this research are tried to solve all emerged challenges.

Initially, two cases (termed A and B) for simple line structures are proposed in this chapter. Following work on the initial algorithms for simple line structure, two further cases, C and D, were developed to resolve problems which emerged in the two initial cases. The weakness of case A is its non-scalability aspect because robots know their number in the swarm. The scalability is embedded in case B, but robots have *a priori* knowledge of the void width to traverse. In the advanced case, the advantages of both initial cases have been combined. Any number of robots in a swarm are able to cross any size of void as long as the number satisfies with the void size. In the case when the number of robots is unsatisfying, they will retreat at the initial zone.

In the simulation for the advanced case, the initial position of robots can be switched from fixed position (case C) into random position (case D). The purpose of fixed initial position is to see if the robot position determines a certain robot to become the seed.

Table 4.1: The evolution of simple line structure algorithms, from A to D. C and D are the combination of A and B, where the difference is the initial positions of robots.

Case	**A**	**B**	**C**	**D**
Robots know the swarm size	Y	N	N	N
Robots know the void size	N	Y	N	N
Initial position	Fixed			Random

In case D, the data of simulation time, the success task and the relation between the swarm size and simulation time are collected, as this swarm represents the scalability.

4.2 Robot Model and Dimension

The robot model used in the simulations for swarms, both in the initial and advanced cases, is shown in Figure 4.1. The simulation model features include differential-drive steering and a ground sensor at the centre-front of the robot for detecting the presence of the voids as well as an obstacles detector at the centre of the robot. To interconnect individual robots, two male connectors on the front face and two female connectors on the rear face were provided. To help robots in positioning themselves perpendicularly to the cliff, two additional ground sensors were placed under the centre of the male connectors. The deployed swarm robots that were simulated had parameters, as can be seen in Table 4.2.

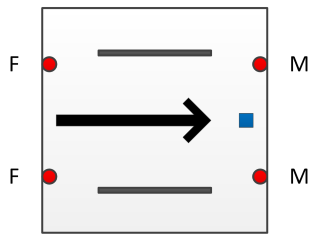


Figure 4.1: The robot model used in simulations. The direction to travel is shown by the arrow. Male (M) connectors are placed on the front face and female (F) connectors on the rear face. The wheels are shown as the two lines, inside the robot structure.

Several parameters changed from initial cases A and B to the advanced cases C and D after evaluations. In the development in accordance with the purpose size of swarm robot, the designed robot's dimension could be decreased from 12cm-cuboid to 10cm-cuboid without sacrificing the design and number of connectors. And since the wheel width does not affect the speed of the robot, it was reduced from dimension 1.9cm to 0.1cm in the later simulations.

Cuboid shape and parameters used for robot model both in simulation and later realised a real platform (Chapter 6) have been chosen after evaluating several robot models as

Table 4.2: Parameter of swarm robots in simulation of algorithms for simple line structures. The changes to cases C and D were made after a number of practical robots were analysed in depth.

Parameters	**Model for cases A and B**	**Model for cases C and D**
Length of the robot	12cm	10cm
Width of the robot	12cm	10cm
Distance between wheels	9.34cm	7cm
Wheel width	1.9cm	0.1cm
Wheel radius	2.1cm	
Maximum velocity	14m/s	

discussed in Chapter 2. As the main proposed approach is self-assembly, the robot shape and connection mechanism play the important role to support the rigid structure of assembled robots. A tight assembly between robots and distributed connection mechanism support the rigid and strong connection and decrease the deflection so that the span of the structure is long. Instead of cylinder, cuboid is chosen as it brings to the tight assembly. A pair of connectors at both front and rear robot makes the weight distributed to all the surface instead of centralised at one point. The robot dimension has been carefully considered as it is not large for a swarm robot, but not small to accommodate connection mechanisms. The wheel dimension is adjusted with the commercial robot wheel. In accordance with the simple requirement of swarm robot, wheels are arranged into differential-drive mechanism. The distance between wheels was adjusted to improved performance.

4.3 Assumptions

Regarding the structure and the features of individual robot model as described in Section 4.2, the following assumptions have been applied to each case:

1. Robots can detect the presence of a void.
2. Communication between robots is very limited. Generated broadcast only informs other robots when a robot: detects the void for the first time; initiates construction; recruits other robots to join the structure; detects the target landing area; detects the target and informs all robots to stop the exploration; and robots in the structure: let all robots in the structure know their number to decide for recruiting or moving forward and backward; let the front robot disassemble from the structure to explore the target zone; let the rear robot disassemble from the structure to explore back the initial zone.
3. There is no predefined leader to initiate the construction of the structure as all swarm robots have the same opportunity to become the seed robot.

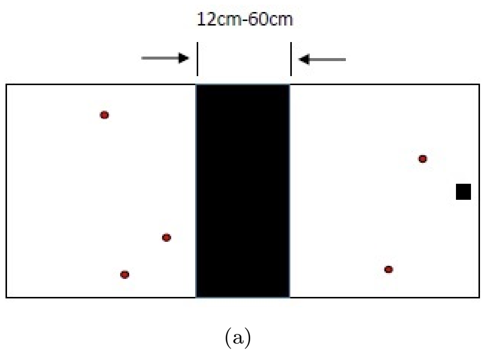
4. The start position of the swarm is fixed for cases A, B and C. The purpose of this setting is to prove that any robot can be the seed robot.
5. The initial position of the swarm is random for case D to verify the scalability aspect and the relation between the swarm size and T_t, the time required for the simulation to be accomplished.
6. Considering the practical implementation, the coupling configuration only permits a structure one robot wide to be constructed, as robots only have connectors at the front and rear.
7. The robot's couplings are considered to be infinitely stiff and the robot is considered to have negligible mass.
8. The forces experienced during the assembly-disassembly process are ignored.
9. The practical physical connection process is not simulated in detail, hence misalignment between two robots is ignored.
10. A single robot is assumed to be able to pull or push at least one other robot, and has a significant braking capability, so it cannot be pushed forward by another robot during the coupling process.
11. The edge of each void is even as well as the terrain. Therefore, it is assumed that robots have no difficulty in positioning themselves according to the void.
12. The elevation of the launch and landing areas is the same as the vertical sides of the void.
13. When the structure is moving across the void, any obstacles encountered during the assembly-disassembly process are ignored, as the complete structure cannot be steered considering the structural stability.

Each case of simple line algorithm has two main tasks, distributed individual level tasks and collective behaviour tasks, which will be presented in Section 4.6. The distributed individual level tasks are the tasks in which individual robots explore the arena to find the void at the initial zone or the target at the target zone. The collective behaviour tasks are the tasks where individual robots have to collaboratively build a structure to help themselves in crossing the void.

4.4 Arena

In addressing the arena size, the length of the arena needed to be considered as well as the prediction of simulation time according to the swarm size. A large arena leads to a longer simulation time because swarm robots need more time to wander. However, a wider void requires a longer structure and, in consequence, it requires a longer arena. Therefore, there were walls as borders to limit where robots could wander, but the wandering zone dimension could be changed based on the swarm size.

In case A, because the swarm size was only up to ten robots, the arena dimensions were only 2.50m $\times$ 2.50m. When the smaller dimension was chosen, the arena could not accommodate the longer structure, as robots in the structure would collide with the static free robots. When the larger dimension was applied, the simulation was run in a longer time because robots needed more time to detect the void when they explored the initial part of the arena. A larger arena was required when case B was applied and the dimensions were changed to 5.0m $\times$ 3.0m based on the same considerations. The later dimension has also been applied for cases C and D, considering the length of the structure and the exploration time in detecting the void. However, obstacles that were placed on the arena for initial algorithms were eliminated to accommodate the larger swarm size and reduce the exploration time to find the void.



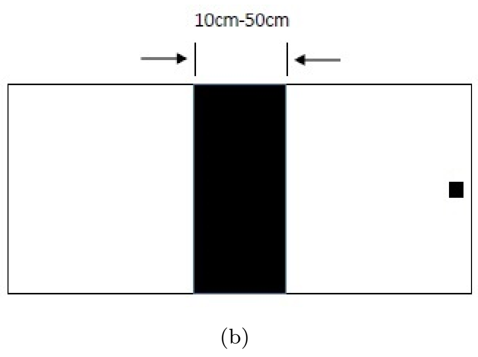


Figure 4.2: Arenas that were used in the simulations explained in the discussion. (a) The arena that was used for cases A and B. (b) The arena that was used for cases C and D. Circles in the arena are static obstacles, and the square is the target to be found by swarm robots.

The arena was equally divided by a void, the width of which was different for each case study, namely: (a) 12cm, (b) 24cm, (c) 36cm, (d) 48cm and (e) 60cm for cases A and B. Later, for cases C and D, the void width was reduced to satisfy the size of robot model, namely: (a) 10cm, (b) 20cm, (c) 30cm, (d) 40cm and (e) 50cm. Three static obstacles were placed on the left side and two other obstacles were located on the right side, together with the target that had to be found by the swarm. The purpose of placing obstacles is to check the obstacle avoidance feature of robots in small swarms. Later, for cases C and D, obstacles were not placed on the arena as the swarm size was large. The illustration of each arena for initial and revised approaches and their features can be seen in Figure 4.2. In this arena, the robots start in the initial zone and the final objective is in the target zone.

4.5 Scenario

In the simulation, a swarm consists of a small number of homogeneous swarm robots which are required to locate a target by traversing a void that separates two halves of an arena, as illustrated in Figure 4.2. Individual robots have no *a priori* knowledge of the void's location in case A, or the void width in case B, and any member of the swarm can autonomously act as the seed robot to initiate the structure's construction.

Once any swarm robot detects the void while wandering on the initial side of the void, this robot becomes the seed robot. Then, it broadcasts a '*void found*'signal to all other members. In cases A and B, free robots stop moving. But in cases C and D, free robots continue the exploration while avoiding the void. Then, the closest robot moves forward to the rear of the seed robot and connects to form a simple line structure. It is followed by another closest robot to join the structure. When the number of robots in the structure passes a preset parameter, the structure moves towards the target zone across the void. Simultaneously, the other robots join the structure to maintain structural stability. When the structure has traversed the void and the number of robots in the target zone is sufficient to maintain stability, the front robot decouples from the structure. The process of additional robots joining the structure continues, while the structure continually moves forward. The process continues until there is no robot at the initial zone. After all the robots have reached the target zone, they wander and search for the target.

If the stability criteria of the structure are not satisfied, or the number of swarm members is not enough to cross the void, the structure will retreat to prevent loss or damage to the robots. Robots in the structure will decouple, then wander on the initial side. After a period of time, they will stop moving and the completion of the task has failed.

There are three main approaches that have been evolved, proposed and simulated for applying simple line structure for crossing a small void: case A, case B and cases C and D. The first approach is case A, when all robots have no *a priori* knowledge of the void location and its width. However, all robots have a knowledge of the swarm size and, thus, scalability is not able to be applied. The cantilever stability is not considered in this algorithm, where, in practice, this balance requires redistributed robots.

Because case A will meet a practical problem when a robot or robots fail, scalability has been considered in order to overcome it by proposing case B. The swarm can consist of any number of robots, from three up to eighteen, and each robot has no *a priori* knowledge of the swarm size. However, this affects the *a priori* knowledge of the void width for the swarm because the algorithm is different from the previous approach. But all robots still have no *a priori* knowledge of the void location.

As for the advanced case, cases C and D, advantages of cases A and B are combined. All robots have no *a priori* knowledge of the void location and its width, and also the swarm size, so that scalability can be applied. To reduce the simulation time, no static obstacle is placed on the arena, so that all robots concentrate on finding the void and crossing it to reach the target zone and find the target. When the structure is being constructed, free and disassembled robots can continue the exploration at initial and target zones whereas in the initial algorithms they have to stop moving.

4.6 Simple Line Structure Algorithms

Each algorithm for simple line structure consists of two main tasks, the distributed individual tasks (search mode, exploration), where all swarm robots have to explore the environment by using Levy walk while searching for the target, and the collective behaviour tasks (build mode, construction), where robots have to construct a structure to help them in traversing the void. The Levy walk is a movement with a broad jump length distribution, although it possesses a finite mean squared displacement (Chechkin et al., 2008). The relationship between exploration and construction can be seen in Figure 4.3.

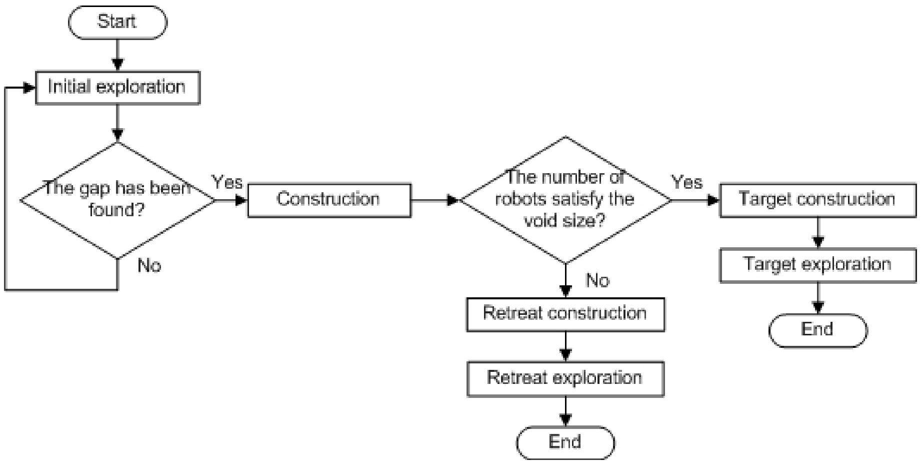


Figure 4.3: The main flowchart of simple line structure algorithms that describe the relationship between all exploration modes and construction mode. In the initial exploration, swarm robots have to find the void location first to build a structure. In the construction when the number of robots is enough to maintain the stability of the structure, all robots will cross the void by continuing to the target construction and explore the target zone. In contrast, when the swarm size is not enough, all robots will retreat by performing retreat construction and continue exploring the initial zone for a while.

Each main task is differentiated by a positive/negative sign of state. For distributed individual level tasks, robots use negative states as the sign that they are not joined to the structure. As for collective behaviour tasks, robots use positive states to describe that they are at a particular position of the structure. Details of each state can be seen in Table 4.3.

Table 4.3: State used in simulation of algorithms for simple line structures.

State	**Explanation**
-3	Retreat state; distributed individual level tasks; robots wander at the initial zone after they have failed to cross the void
-2	Target state; distributed individual level tasks; robots wander at the target zone to locate the target
-1	**Initial state; distributed individual level tasks; robots wander at the initial zone to locate the void**
1	Collective behaviour tasks; the first robot is positioned in the structure
2	Collective behaviour tasks; the second robot is positioned in the structure
3	Collective behaviour tasks; the third robot is positioned in the structure
⋮	⋮
n	Collective behaviour tasks; the n^{th} robot position in the structure

Initially the state in all of robots is (-1), which allows them to wander at the initial zone to locate the void. Once a robot detects the void, it becomes the seed robot and its state changes to (1). The second robot that joins it in forming a line structure has a state (2) and so forth. After the structure has successfully traversed the void, the front robot decouples and its assemble state changes into (-2). In case A, the first robot still holds the assemble state (1), the second robot still holds the assemble state (2), the third robot still holds the assemble state (3) and so forth, although their position in the structure changes: the first robot is now not the first robot in the structure, the second robot becomes the first robot, the third robot becomes the second robot and so forth. But, in cases B, C and D, their assemble states change: the second robot changes into (1), the third becomes (2) and so on.

In a situation where the structure does not satisfy the stability criteria or it does not have enough swarm members to cross the void, the structure will retreat to prevent loss or damage to the robots. The rear robot in the structure will decouple and its assemble state will change to (-3). In case A, the *n-1* robot still holds the assemble state (*n-1*) although its position in the structure becomes the n^{th}. But, in cases B, C and D, its assemble state changes into (n), the *n-2* becomes (*n-1*) and so on.

The detail of distributed individual level tasks and its exploration task is discussed in Section 4.6.1 and the detail of collective behaviour tasks for simple line structure algorithm is discussed in Section 4.6.2.

4.6.1 Distributed Individual Level Tasks

The distributed individual level tasks of each algorithm consist of three main explorations: i) initial exploration, ii) target exploration and iii) retreat exploration.

4.6.1.1 Initial Exploration

The state of all robots at the initial time is set as (-1), and their initial positions can be set into fixed positions for initial cases A and B, or fixed or random positions for advanced cases C and D. Robots explore the initial zone to locate the void. All of them have no *a priori* knowledge of void location and its size, except for case B where they know the size of the void. Once a robot detects a void, it becomes the seed robot and initiates the construction. The algorithm changes into a construction algorithm. The flowchart of the initial exploration can be seen in Figure 4.4.

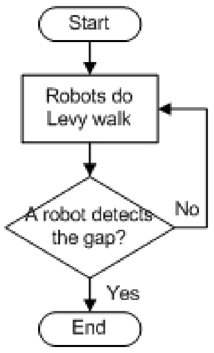


Figure 4.4: The initial exploration algorithm. Robots explore the initial zone. If a robot detects a void, it becomes the seed robot to initiate the construction.

4.6.1.2 Target Exploration

In the target exploration, once the robots successfully cross the void and disassemble from the structure, the state turns to (-2), indicating they are ready to wander on the target side. They wander to locate the target while avoiding obstacles, the void and arena walls. The algorithm for this phase can be seen in Figure 4.5. In case A, this mode is enabled once all robots have reached the target zone. Once a robot disassembles from the structure, it will position itself at a particular location and wait until all robots disassemble. After the last two robots have dispersed, the rear robot in the structure sends a signal to all robots to wander. Because this strategy is not sufficient for scalability, the (-2) state is enabled for an individual robot when it has reached the target zone in cases B, C and D, and it can wander right after disassembling from the structure. Once a robot detects the target, the simulation will stop and the main mission, finding the target, is considered a success.

4.6.1.3 Retreat Exploration

When the number of robots in the swarm is not enough to reach the target zone or to maintain the structural stability although part of the structure has reached the target ground, the structure will retreat and the rear robot decouples from the structure. Once

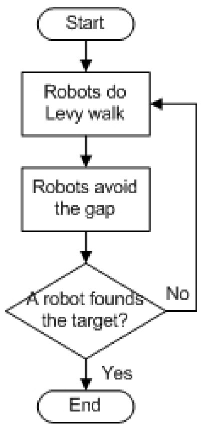


Figure 4.5: The target exploration algorithm. After robots have decoupled and are dispersed, they wander at the target zone and move randomly. They have to avoid obstacles and arena walls as well as avoiding the void. Once a robot detects the target, the simulation stops and the task is considered a success.

all robots have disassembled from the structure, they will do Levy walk while avoiding the void. After a while, all robots stop moving and the simulation stops. This algorithm can be seen in Figure 4.6. The state for retreated robots is (-3).

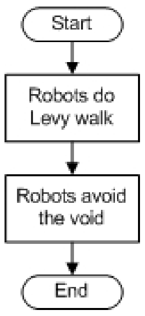


Figure 4.6: The exploration algorithm when robots fail to cross the void. After robots decouple and disperse, they wander back into the initial zone while avoiding obstacles, arena walls and the void.

Between initial and target explorations or between initial and retreat explorations, there is a collective behaviour task to do, building a simple line structure constructed by assembled swarm robots. The constructed structure is required to enable swarm robots crossing the void. The details of the algorithm are discussed in the next subsection.

4.6.2 Initial Cases

All cases only apply a simple obstacle avoidance that works well for static obstacles. However, collisions still occur when robots are facing dynamic obstacles, such as other robots. For case A, robots memorise their number position in the structure after disassembling to make them position themselves in a particular way at the opposite zone (or at the initial zone when they fail to traverse the void) to avoid the collision, based on odd and even numbers. Then, disassembled robots stay still until all robots disperse from the structure to explore the opposite zone finding the target. For cases B, C and D, right after disassembling, the memory of robots number position in the structure is erased. And, because of a simple obstacle avoidance, there is a possibility that robots are spinning and piled up in front of the structure at the opposite zone. Thus, it is possible that the wandering time at the opposite zone becomes longer.

4.6.2.1 Case A

In Figure 4.7, case A is proposed. It can be applied to a sum of five to ten robots, but only five, seven, eight and ten robots were simulated. Once a robot detects the void, it becomes the seed robot (R_s). Then, it broadcasts to other robots that the void has been found and it positions itself orthogonal to the edge of the void. After the position has been set, the closest robot in the initial zone moves and connects to the seed robot, forming line structure.

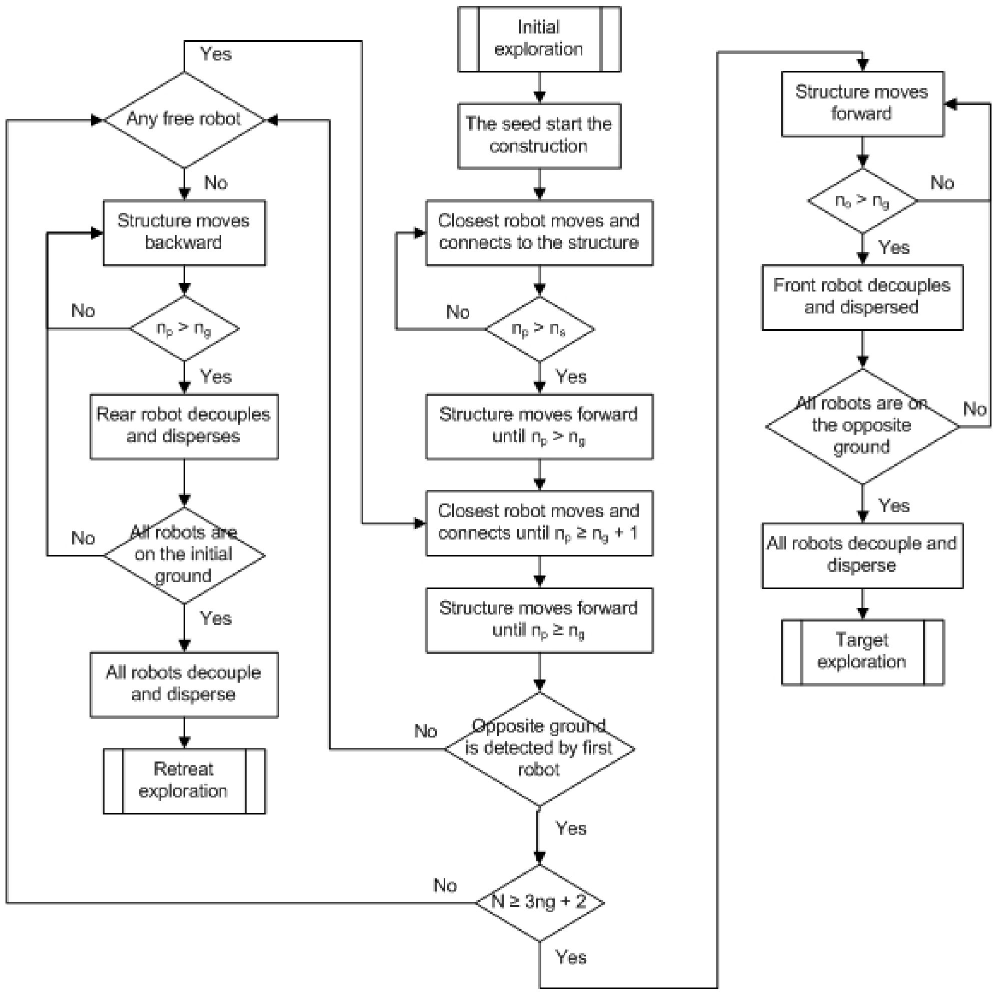
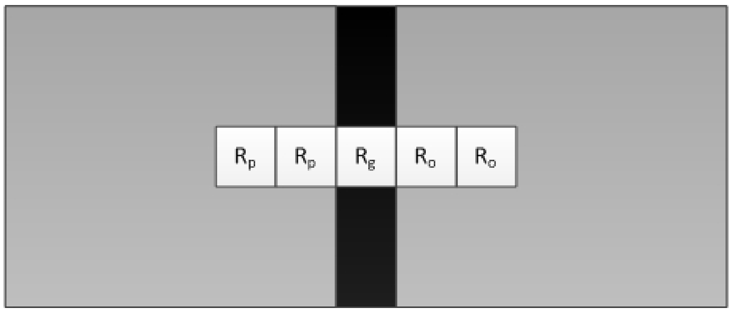
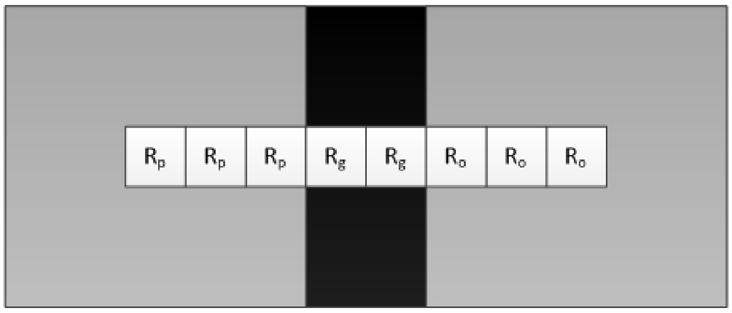

Initial exploration
The seed start the construction
Closest robot moves and connects to the structure
$n_p > n_s$
No
Yes
Structure moves forward until $n_p > n_g$
Closest robot moves and connects until $n_p \geq n_g + 1$
Structure moves forward until $n_p \geq n_g$
Opposite ground is detected by first robot
No
Yes
$N \geq 3ng + 2$
No
Yes
Any free robot
Yes
No
Structure moves backward
$n_p > n_g$
No
Yes
Rear robot decouples and disperses
All robots are on the initial ground
No
Yes
All robots decouple and disperse
Retreat exploration
Structure moves forward
$n_s > n_g$
No
Yes
Front robot decouples and dispersed
All robots are on the opposite ground
No
Yes
All robots decouple and disperse
Target exploration

Figure 4.7: The build algorithm for case A. In this case, a fixed number of robots have to cross a number of different voids, from 12cm to 60cm.

Then, the structure checks if $n_p > n_s$, where n_p is sum of the robots in the structure in the initial zone, R_p, as well as n_s is sum of R_s in the structure. If this requirement has not been fulfilled, the next closest free robot moves and connects to the structure. After three robots ($n_p > n_s$) have been assembled, the structure moves forward until the number of robots on the ground (n_p) is larger than the number of robots over the void (n_g). Then, the process continues with other free robots moving to and assembling with the structure until $n_p \geq n_g + 1$, and then the structure moves forward until $n_p \geq n_g$. Then, the front robot in the structure checks if the target ground is detected. If it is detected, the structure checks the total number of robots that are assembled in the structure (N) and compares it to the number of robots over the void. If $N \geq 3n_g + 2$, then the structure moves forward until the number of robots on the target ground (n_o) is larger than n_g. Then, the front robot decouples and disperses, moves to a location the distance and direction of which have been set, and will stay still. Moving forward, disassembly continues until all robots in the structure reach and position themselves at the target ground. If this condition has been accomplished, all robots decouple and disperse and the algorithm switches into target exploration.



(a)



(b)

Figure 4.8: The structural stability of case A. (a) To hold a robot over the void, the minimum number of robots at both grounds is two robots. (b) And if there are two robots over the void, at least three robots are required to hold them (bottom). These illustration satisfies the minimum requirement structural stability, where $n_p = n_g + 1$ and $n_o = n_g + 1$, thus $N_{min} = 3n_g + 2$.

If the target ground has not been detected, or if $N < 3n_g + 2$, the structure calls the closest free robot to assemble. If there is any free robot, it will move and connect to the structure until $n_p \geq n_g + 1$ and the process continues with the structure moving forward and the front robot checking the target ground. But, if there is no free robot before the target ground has been detected and the total number of robots is less than the required number compared to the void width, then the structure moves backward, all robots decouple and disperse and the process switches to retreat exploration.

To ensure the structural stability for case A, a conservative relationship between the number of robots over the void (n_g) and the minimum number of robots in the structure (N_{min}) has been determined, where $N = 3n_g + 2$. This relationship is defined to ensure that the structure would, at no time, topple into the void. This requirement gives the results $n_p \geq n_g + 1$ and $n_o \geq n_g + 1$, where n_p and n_o are the number of robots on the initial side and target side, respectively. If the minimum number is taken, then $n_p = n_g + 1$ and $n_o = n_g + 1$, thus $N = n_p + n_g + n_o = 3n_g + 2$, as can be shown in Figure 4.8.

4.6.2.2 Case B

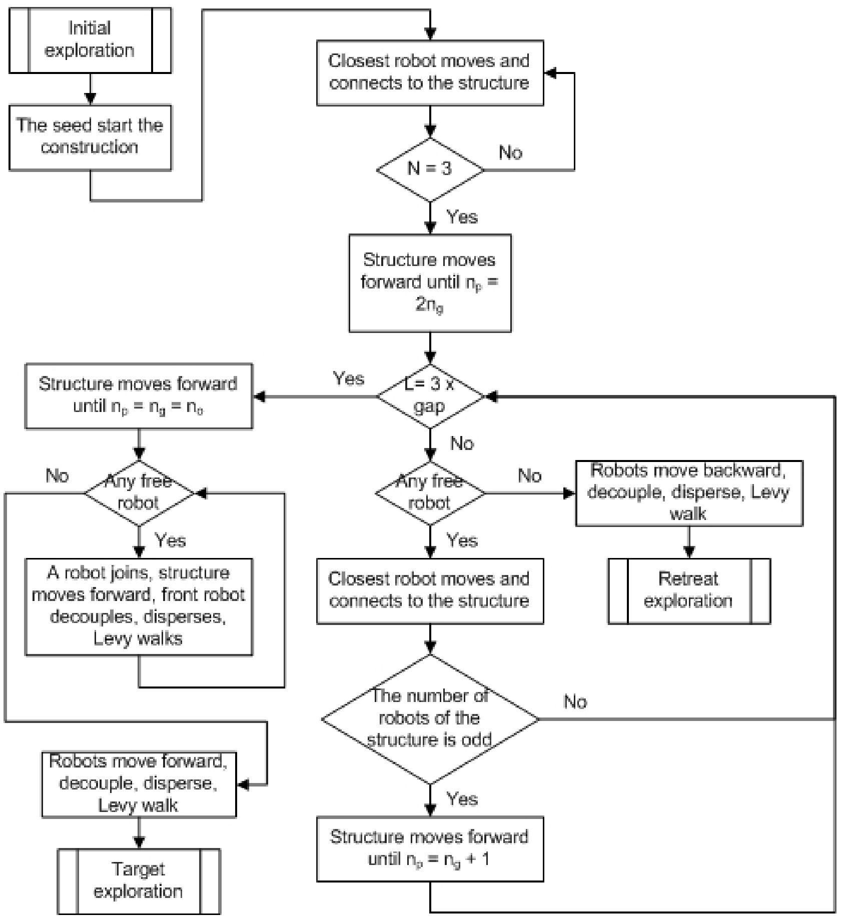

Initial exploration
The seed start the construction
Closest robot moves and connects to the structure
N = 3
No
Yes
Structure moves forward until $n_p = 2n_g$
L= 3 x gap
Yes
No
Structure moves forward until $n_p = n_g = n_o$
Any free robot
No
Yes
A robot joins, structure moves forward, front robot decouples, disperses, Levy walks
Robots move forward, decouple, disperse, Levy walk
Target exploration
Any free robot
No
Yes
Robots move backward, decouple, disperse, Levy walk
Retreat exploration
Closest robot moves and connects to the structure
The number of robots of the structure is odd
No
Yes
Structure moves forward until $n_p = n_g + 1$

Figure 4.9: The build algorithm for case B. In this case, a different size of robotic swarm has to cross a void of fixed width.

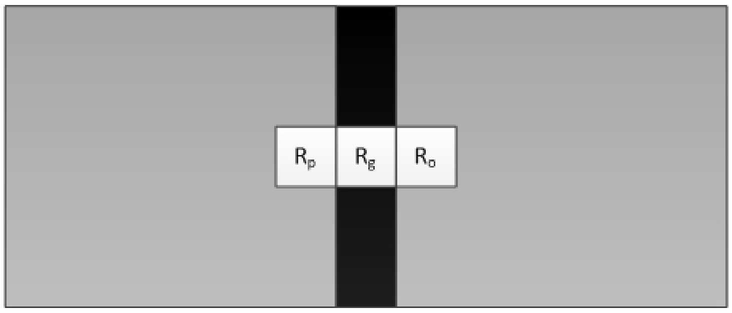
Because case A cannot accommodate scalability, where the predefined number of robots in a structure will meet a problem if several robots have failed in the operation or a number of robots are added to the swarm, case B has been proposed. Taking the same case studies for different void sizes, any number of robots can be deployed in the arena. However, all robots have *a priori* knowledge of the void size. In addition, although the number of assembled roobts to cross the void has been reduced, the conservative relationship is still applied for the safety. As robots are not guaranteed to be able to grip to the ground at any friction level, it is safe to take the conservative relationship to prevent the structure falling into the void.

In this approach, as per the flowchart illustrated in Figure 4.9, once a robot in initial exploration detects the void, it becomes the seed robot. Then, it broadcasts to other robots informing them that the void has been found. After it has positioned itself orthogonal to the edge of the void, the closest free robot moves and connects to it. Then, the structure checks itself to see if the number of robots in the structure (N) equals three. If not, the closest free robot will move and connect to the structure.

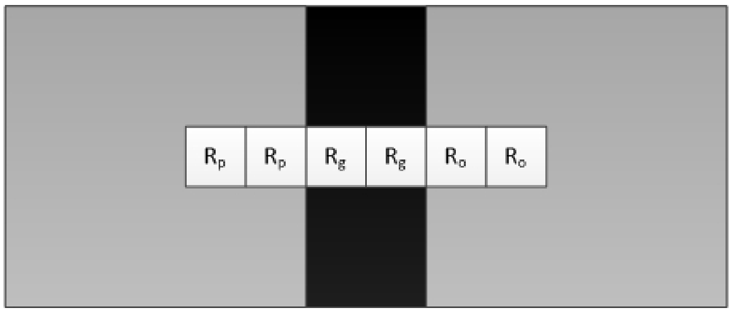
Once $N = 3$, the structure moves forward until $n_p = 2n_g$. If $L = 3 \times void$, then the structure moves forward until $n_p = n_g = n_o$, where L is the length of the structure. If there is no free robot, the whole structure moves forward to the target ground and all robots decouple and disperse. Then, the mode switches to target exploration. But, if there is a free robot, the free robot moves and connects to the structure. Then, the structure moves forward until the second robot reaches the previous first robot position on the target ground. Then, the first robot decouples and disperses. The process repeats to check if there are any free robots.

In the case where $L \neq 3 \times void$, there is checking for any free robot. If there is no free robot, then the structure moves backward, because its length is less than three times the void width. Then, all robots decouple and disperse and the mode switches to retreat exploration. However, if there is a free robot, the closest robot moves and connects to the structure. Then, the structure checks if the number of robots in the structure is odd. If yes, the structure moves forward until $n_p = n_g + 1$ and the process goes back to checking if $L = 3 \times void$. But if the number of robots is even, it will go directly to checking if $L = 3 \times void$.

Because the predefined number for a structure in case A requires more robots at each ground side (n_p and n_g), after a careful consideration and a simple observation using assembled bricks, the structural stability using these numbers can be reduced. The conservative relationship between N, n_g, n_p and n_o for algorithm B was changed. The requirement resulted in the relationships $n_p = n_g = n_o$ and $N = n_p + n_g + n_o = 3n_g$. The illustrated relationship is shown in Figure 4.10.



(a)



(b)

Figure 4.10: The structural stability of case B. (a) If there is a robot over the void, a robot at each ground side satisfies the structural stability requirement. (b) If there are two robots over the void, two robots at each ground side are enough to hold them (bottom). Reducing the minimum number of ground robots at each side is satisfying the requirement of structural stability, where $n_p = n_g$ and $n_o = n_g$, thus $N = 3n_g$.

4.6.3 Advanced Cases C and D

The construction algorithm for advanced simple line structure algorithm can be seen in Figure 4.11. Once a robot has detected the void, it becomes the seed robot. Then, it sets orthogonally to the void and broadcasts the information that the void has been found so that other robots will not initiate the structure construction. Other robots continue wandering at the initial zone while avoiding the void.

Then, the seed robot calls the closest free robot to get close and connect to the rear part of the seed robot. The structure formed by assembled robots checks itself to see if there are three robots in it. It continues to call the closest free robot to join the structure. When the number has been satisfied, the structure moves forward until $n_p = 2n_g$, or the front robot is over the void and it is held by two other robots. The front robot checks if there is the target ground.

When the target ground has been detected, the structure moves forward until $n_p = n_g = n_o$, or the number of robots at the initial zone equals to the number of robots over the void equals to the number of robots at the target zone. If there are other free robots, the structure continues calling the closest one to join, then it moves forward

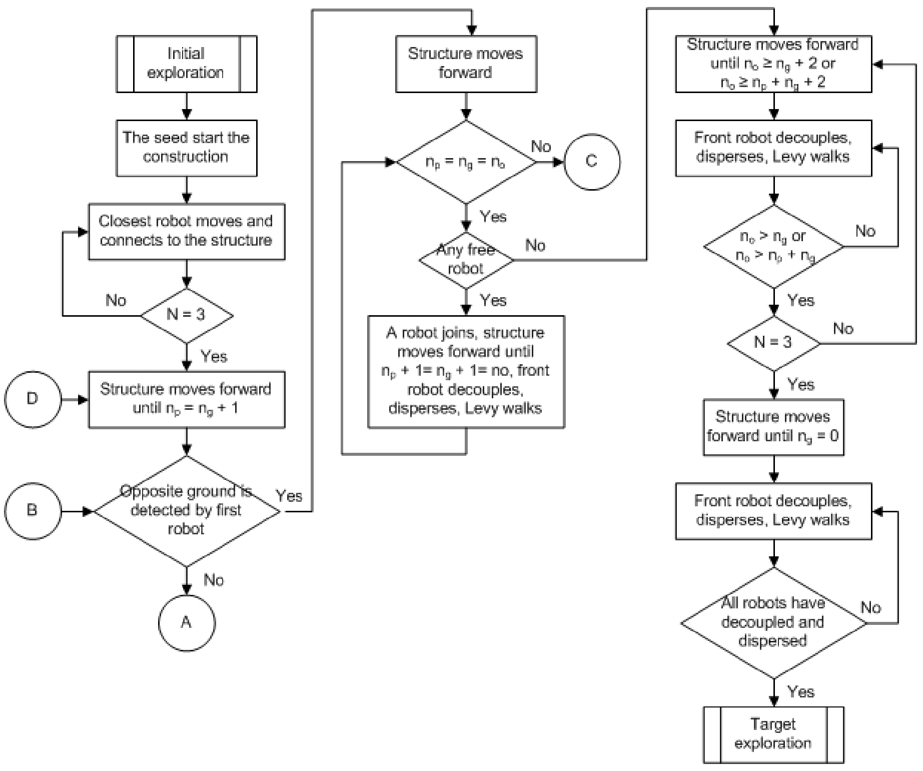

Initial exploration
The seed start the construction
Closest robot moves and connects to the structure
N = 3
No
Yes
D
Structure moves forward until $n_p = n_g + 1$
B
Opposite ground is detected by first robot
Yes
No
A
Structure moves forward
$n_p = n_g = n_o$
No
C
Yes
Any free robot
No
Yes
A robot joins, structure moves forward until $n_p + 1 = n_g + 1 = n_o$, front robot decouples, disperses, Levy walks
Structure moves forward until $n_o \geq n_g + 2$ or $n_o \geq n_p + n_g + 2$
Front robot decouples, disperses, Levy walks
$n_o > n_g$ or $n_o > n_p + n_g$
No
Yes
N = 3
No
Yes
Structure moves forward until $n_g = 0$
Front robot decouples, disperses, Levy walks
All robots have decoupled and dispersed
No
Yes
Target exploration

Figure 4.11: The build algorithm for cases C and D part 1. In this case, a different size of robotic swarm has to cross different sizes of void. Points A-D are given in Figure 4.12.

until $n_p + 1 = n_g + 1 = n_o$. Then, the front robot disassembles, disperses and wanders at the target zone. The process of calling other robots continues until there is no free robot.

If there are no more free robots to join to the structure, it moves forward until $n_o \geq n_g + 2$ or $n_o \geq n_p + n_g + 2$; then the front robot decouples, disperses and wanders at the target zone. The structure then checks if $n_o > n_g$ or $n_o > n_p + n_g$. If it is not satisfied, the next front robot disassembles, moves away from the structure and explores the target area.

When there are already just three robots in the structure, the structure moves forward until all robots reach the target ground, or $n_g = 0$. Then, the front robot continues decoupling, dispersing and exploring the target area. When all robots have disassembled from the structure, or there is no structure, they start to do the target exploration.

In the case when the front robot does not detect the target zone, the structure will continue calling free robots to connect to its rear. It checks if the number of robots in it is $n_p \geq n_g + 2$ or $n_p \geq n_o + n_g + 2$, then it moves forward until $n_p = n_g + 1$, and rechecks if the target ground has been detected.

When there is no free robot but the structure does not detect the target ground, or if robots at each area (initial, void and target) do not satisfy $n_p = n_g = n_o$, the structure

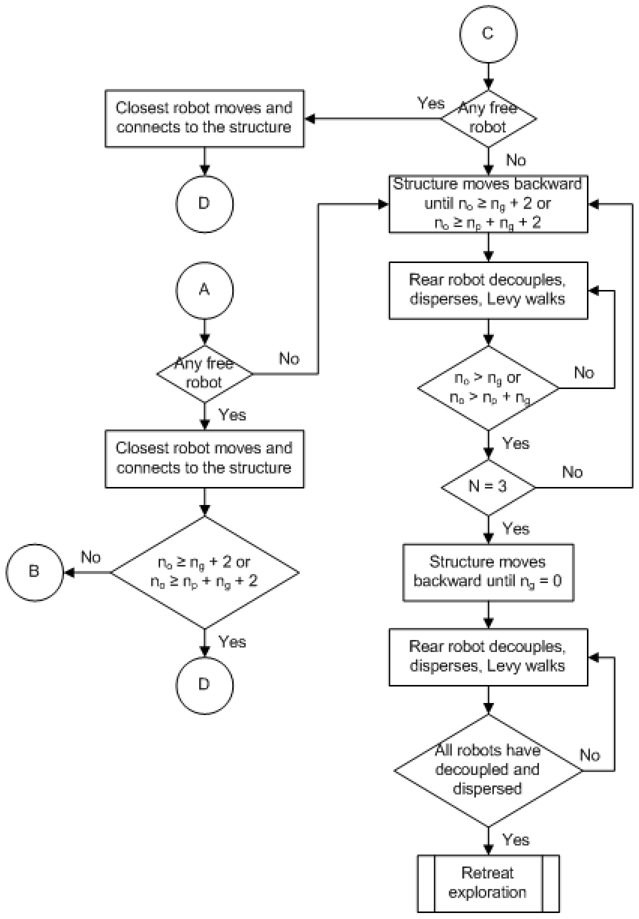

C
Any free robot
Yes
Closest robot moves and connects to the structure
D
No
Structure moves backward until $n_o \geq n_g + 2$ or $n_o \geq n_p + n_g + 2$
Rear robot decouples, disperses, Levy walks
$n_o > n_g$ or $n_o > n_p + n_g$
No
Yes
N = 3
No
Yes
Structure moves backward until $n_g = 0$
Rear robot decouples, disperses, Levy walks
All robots have decoupled and dispersed
No
Yes
Retreat exploration
A
Any free robot
No
Yes
Closest robot moves and connects to the structure
$n_o \geq n_g + 2$ or $n_o \geq n_p + n_g + 2$
No
B
Yes
D

Figure 4.12: The build algorithm for cases C and D part 2. In this case, a different size of robotic swarm has to cross different sizes of void. Points A-D are connected to Figure 4.11.

moves backward until $n_p \geq n_g + 2$ or $n_p \geq n_o + n_g + 2$. The rear robot then disassembles, disperses and explores the initial zone again. The retreating of the structure and the decoupling of the rear robot continue until $n_p > n_g$ or $n_p > n_o + n_g$. When the number of robots in the structure is only three, it moves backward until $n_g = 0$, or all robots have reached the initial area. Then, the rear robot continues to disassemble, disperse and explore. When all robots have decoupled and dispersed, the algorithm changes into retreat exploration.

The difference between cases C and D is the setting of robot initial positions. The rest of algorithm is the same. In case C, all robot initial positions are set fixed, varying from three to eighteen robots. They are facing the same direction, which is the void. In case D, all robot initial positions are set at random, varying from three to fifty robots. Each robot is also facing a random direction.

4.7 Results

The obtained results of two initial algorithms have the same characteristics, swarm robots are able to traverse the void if their number satisfies the structural stability according to the void width. When their number is not enough, they are able to retreat to the initial zone. However, there is a different behaviour between these types because of the different characteristics of the algorithms. Results shown in this section represents similar results that can be found in Appendix A. For cases A and B, five running simulations were conducted for each swarm size in each n-robot void. But, for advanced cases C and D, the conducted running simulations were thirty.

4.7.1 Case A

In this case, swarms are set to five, seven, eight and ten swarm robots. Each swarm size has to cross a void and it has no *a priori* knowledge of its width. It can be 12cm, 24cm, 36cm, 48cm or 60cm. Five simulations were carried out for each swarm size and each void width. The random initial position of swarm robots was not applied because it is highly possible to make the closest robot to the void become the seed robot. Moreover, by setting the fixed initial position, it can be seen that all robots have the same opportunity to be the leading robot, as has been the case in Figure 4.13. Although the frequency with which each robot ID is used is not the same, it demonstrates that any robot is able to become the seed robot to initiate the building of the structure.

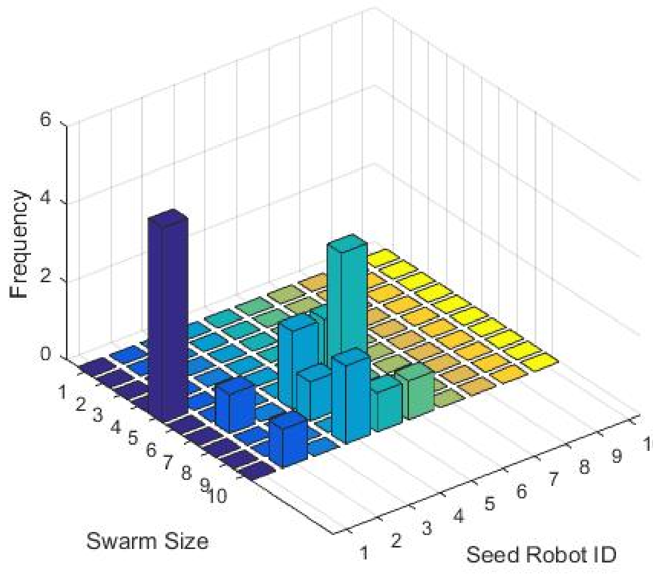


Figure 4.13: Frequency with which a specific robot became the seed robot for case A for three robot wide void.

As expected, simulation results show that the strategy applied to the swarm works. All swarm sizes could pass the void and reach the target void for the case study with a 12cm void width. It is appropriate with the requirement of $N = 3n_g + 2$. To traverse a 12cm void (a one robot wide void) a structure needs to be at least 60cm long to hold the stability. Following this requirement, the greater the number of robots the more the void is able to be crossed. The illustration of how a swarm consists of five robots crossing the void can be seen in the captured snapshots in Figure 4.14.

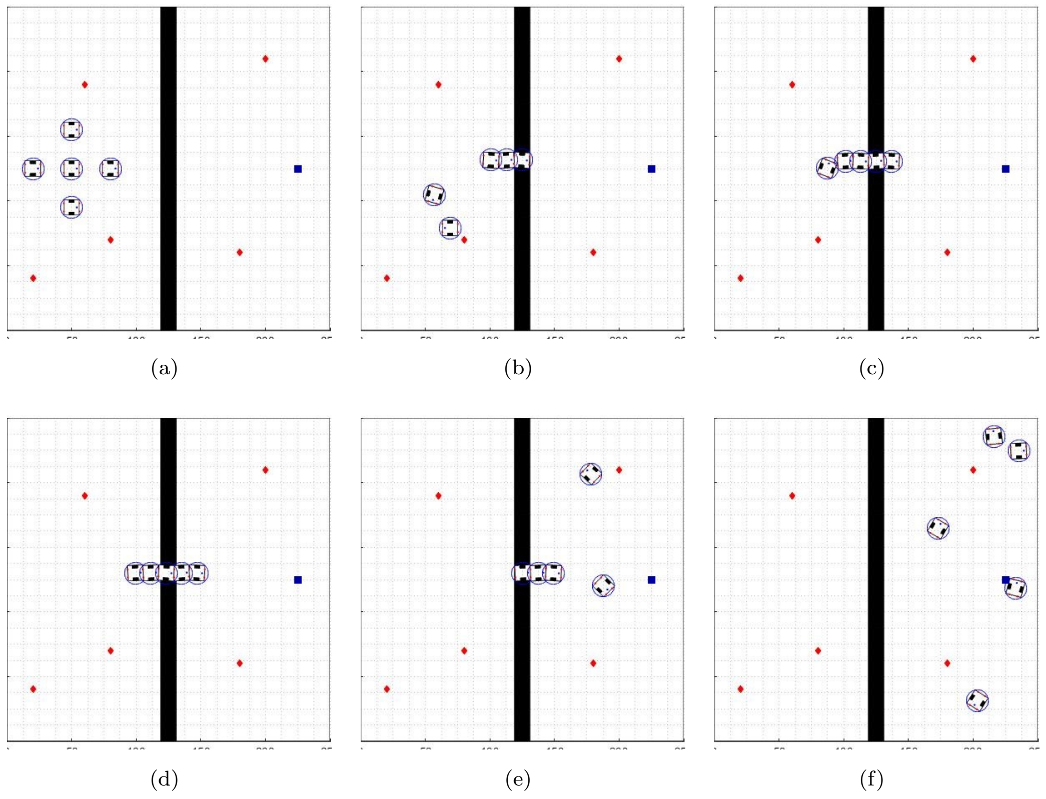


Figure 4.14: Five robots attempting to cross a one robot wide void. (a) The initial position when robots were ready to wander at the initial zone. (b) Three robots have assembled and the structure has moved forward when two others were joining and/or are waiting to join the structure based on their distance. (c) The last robot joined the structure when the structure had moved further on. (d) All robots have assembled and the structure has moved further. (e) Two robots have disassembled and avoided the void while the others in the structure have moved further. (f) All robots have decoupled and one robot has detected the target.

However, a swarm of five robots could not traverse a 24cm void, as illustrated in Figure 4.15 because, to cross it, it was necessary to have at least eight robots. Although it could not pass the void, the structure retreated to the initial zone to avoid falling into the void and the swarm robots dispersed to wander again at the initial zone for a period of time.

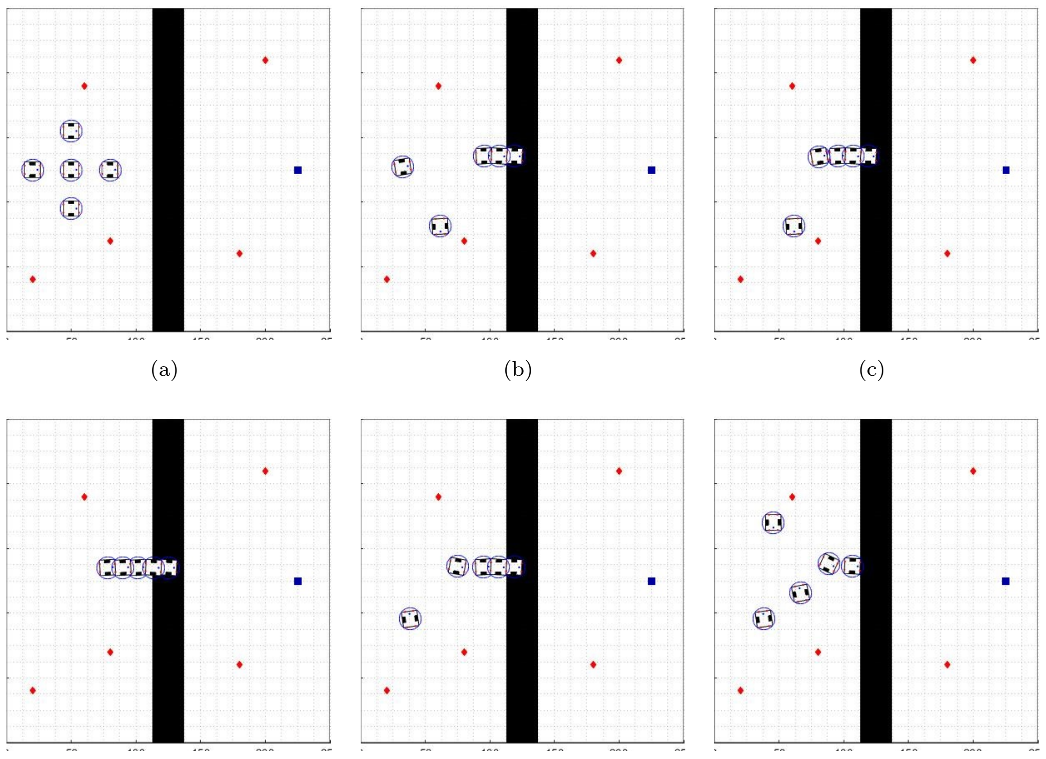

(a) (b) (c)
(d) (e) (f)

Figure 4.15: Five robots attempting to cross a two robot wide void. (a) The initial position of five robots. (b) Three robots have coupled onto a structure to cross the void. (c) The most distant robot has moved to join the structure. (d) All robots have assembled and the structure has moved forward to cross the void. (e) The structure has retreated and the robots have decoupled. (f) All the robots have decoupled and reached the initial zone to wander in it for a period of time.

In the case of a swarm consisting of more than eight robots, a series of snapshots of how ten robots would traverse the void can be seen in Figure 4.16. A minimum number of eight robots is required in the structure to hold stability while the other robots can freely wander at the initial or target zones.

The further simulation is illustrated in Figure 4.17 where ten robots failed to cross a 60cm void. The required minimum number of robots to traverse a 60cm void is seventeen. From the snapshots, it can be seen that the structure retreated when the front robot could not reach the target area. Then, all robots decoupled in the initial area to wander in it for a period of time.

From all the results, all swarm sizes could pass the 12cm void, but it was only a swarm size of eight or ten that could traverse the 24cm void. All swarm sizes were unable cross a void larger than 24cm (36cm, 48cm and 60cm) because the number of robots was too few. However, they successfully retreated to the initial zone and decoupled to prevent the loss of robots into the void.

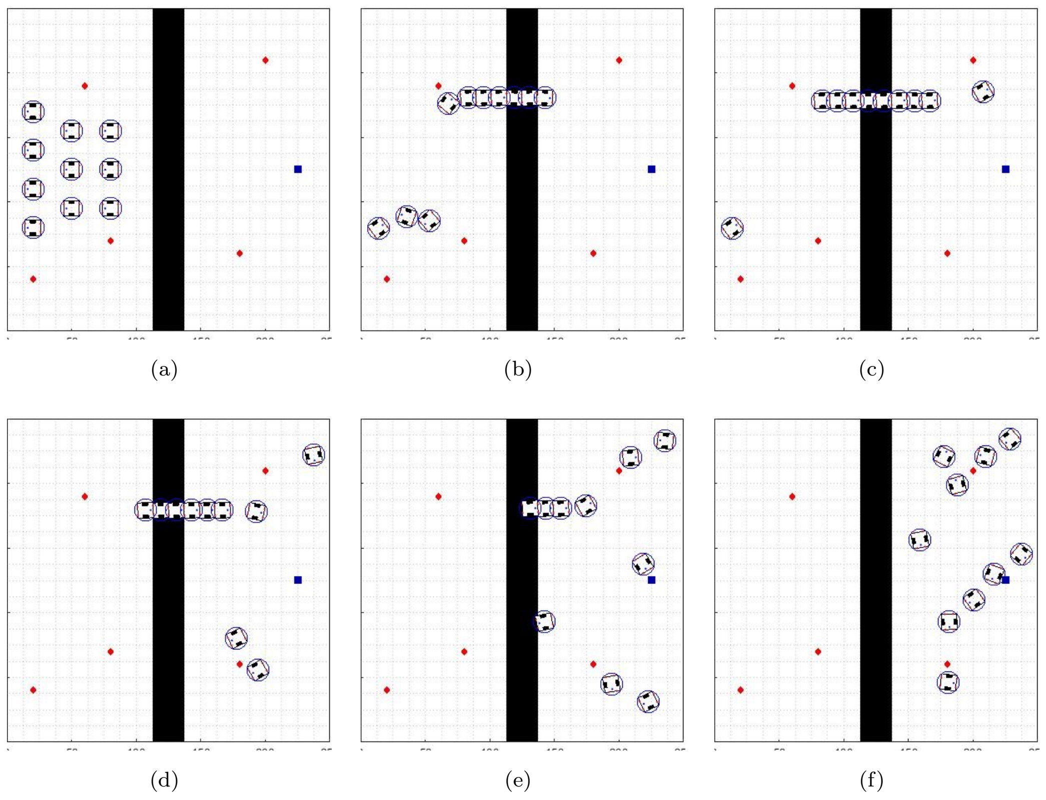


Figure 4.16: Ten robots attempting to cross a two robot wide void. (a) Ten robots were ready to wander. (b) Six robots have assembled while one robot has joined the structure and three others are waiting to join. (c) A robot has reached the target zone and wandered in it while the structure of eight robots waits for another robot to join it and moves forward. (d) All robots have joined the structure and a number of them have wandered in the target zone. (e) The structure has moved further and more robots have decoupled. (f) All robots have disassembled and wandered in the target area, then one robot has detected the target.

For this approach, a strategy of setting a direction when a robot decouples from the structure has been applied for initial and target areas. In the target zone, it has been set based on a sequence such that the odd numbered robots reaching the ground move to the upper side of the arena and the even numbered ones move to the lower side before they wander in the target zone. Meanwhile, at the initial zone, when the structure fails to cross the void, the strategy directs robots to the upper side, the lower side or straight back based on the sequence and then they remain still before all the robots disassemble. After all the robots have decoupled, then they wander for a period of time. This strategy is to prevent collision of robots into one another. However, this demonstrates the slow response in writing Matlab script when more robots were added to the swarm, because disassembled robots still held assemble numbers of **build** mode. This is also the reason why the maximum number of robots in a swarm is only ten.

The summary of T_a and T_d can be seen in Table 4.4 and Table 4.5, respectively, where

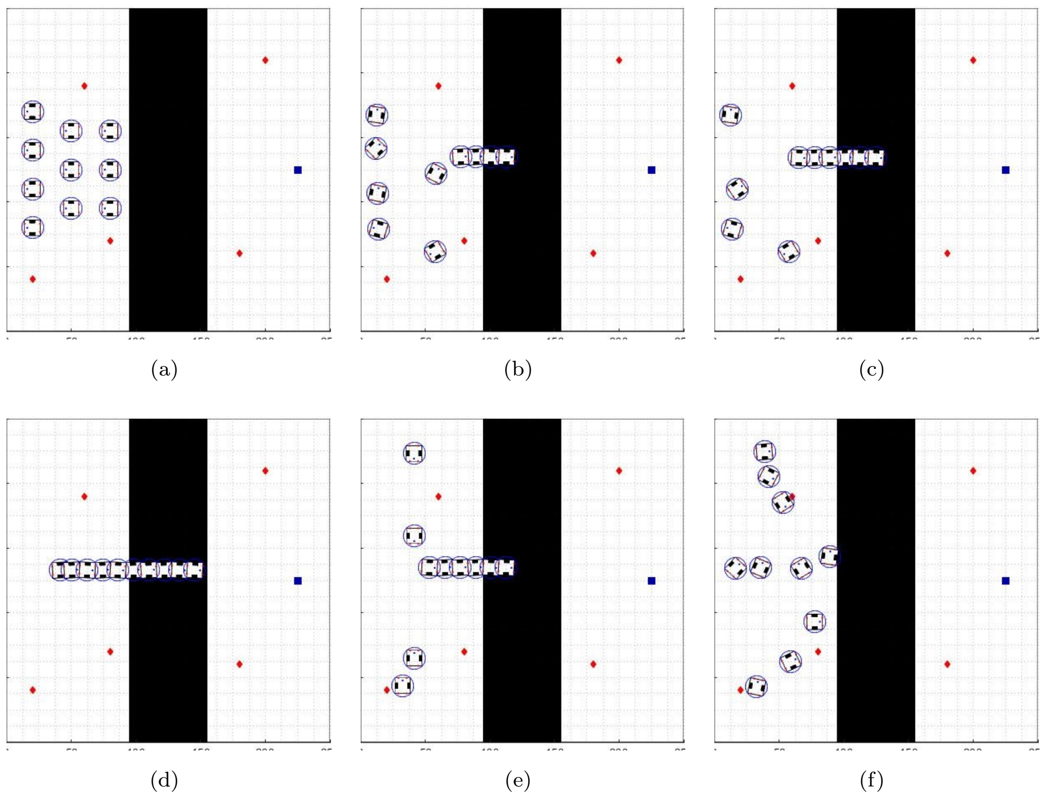


Figure 4.17: Ten robots attempting to cross a five robot wide void. (a) The initial position. (b) Four robots have coupled while others wait and move to join the structure. (c) The structure grew in length and moved further towards the target ground. (d) All robots have assembled in the structure, but it moved back to the initial zone because the front robot could not reach the target area and there was no other robot to support the structure. (e) A number of robots have decoupled at the initial zone. (f) All robots have decoupled and they wander in the initial area for a period of time.

T_a is the time required for assembly process in the construction, and T_d is the time required to disassemble after robots retreating. The total time for robots to successfully reach the target zone depended on T_s, T_a and T_o, where T_s and T_o cannot be predicted. T_s and T_o are the time required for the exploration at the initial zone and target zone, respectively. It was expected that the smaller the swarm size, the shorter the simulation time, because the structure needs less time when the self-assembly process happens. Indeed, T_a for the small swarm size was also short. However, we could not predict T_s and T_o, because the smaller the swarm size, the longer the time that the robots wander. This is because the density of robots in the arena is smaller so that a robot rarely meets other robots in a range of time and this reduces the probability of a robot detecting the void or the target. Moreover, in the target zone, if a robot location is close to the target, it is faster for the swarm to finish the task. However, in several cases sometimes the larger swarm required more time to wander. This is because robots in it had to make more effort to avoid each other while avoiding other obstacles.

T_t in the case when robots failed to traverse the void depended on T_s, T_a, T_d and T_b (the time required to explore the initial zone after retreating) where we could only predict that the larger the swarm it takes a longer time in T_a and T_d. For all T_t for case A, it can be seen in Table 4.6. In general, T_t is longer when the swarm size is larger, but shorter when the void size is larger because mostly they failed to cross the void, and the wandering time is shorter because the initial zone is smaller.

It was also expected that the wider the void, the shorter the T_s because the wandering area becomes narrow and it increases the probability of robots detecting the void. However, this limited area also caused another problem. It limited the robot motion and coverage and also restricted the process of building the structure. The longer the structure, the longer the required arena.

Because problems emerged with this approach, the arena dimension for case B was changed for a longer arena. The slow response problem in writing Matlab script was investigated and it resulted in applying the dynamic variables for robots in the structure. Because case A would meet a problem during practical implementation when there is a probability that the swarm size will change, case B needs to be developed. This is because the structure is highly dependent on the number of robots in the swarm. If one robot or more has failures, the structure cannot be built.

Table 4.4: The average assembly time (in seconds) to form a structure for all void widths (in cm) and swarm sizes for case A.

Void Width	Swarm Size			
	Five	Seven	Eight	Ten
12cm	54	82	105	131
24cm	35	57	94	131
36cm	28	58	63	94
48cm	30	60	66	91
60cm	29	51	64	94

Table 4.5: The average disassembly time (in seconds) when the structure fails to cross all void widths (in cm) and swarm sizes for case A.

Void Width	Swarm Size			
	Five	Seven	Eight	Ten
12cm	-	-	-	-
24cm	14	29	-	-
36cm	14	27	24	41
48cm	14	23	24	41
60cm	14	22	22	41

Table 4.6: The average simulation time (in seconds) to cross all void widths (in cm) and swarm sizes for case A.

Void Width	Swarm Size			
	Five	Seven	Eight	Ten
12cm	133	100	150	159
24cm	58	96	133	171
36cm	50	93	95	145
48cm	50	91	101	137
60cm	49	79	92	139

4.7.2 Case B

In this approach, a swarm that has no *a priori* knowledge of its size has to traverse a void of a known width. As with the previous approach, the void widths are 12cm, 24cm, 36c, 48cm and 60cm. However, the size of the swarm of robots can change from three, six, nine, twelve, fifteen to eighteen robots. Five simulations were also carried out for each void width and each swarm size, as well as the setting of the fixed initial position for the same reason. The summary of the frequency with which robot IDs became the seed robot can be seen in Figure 4.18.

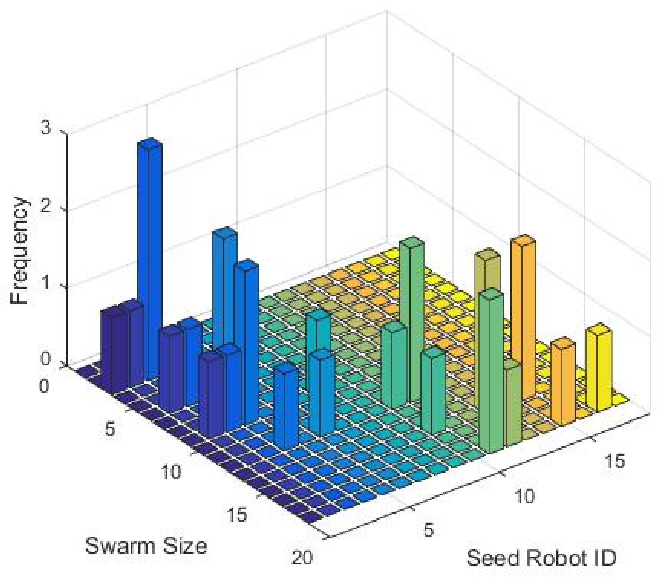


Figure 4.18: Frequency with which robots became the seed robot for case B, any robot can become the seed.

The required minimum number of robots in the swarm was reduced so that to cross a void it only needed $N = 3n_g$ robots. In other words, a 12cm void only needs three robots to be crossed, a 24cm void needs six robots to be traversed and so on.

In the general strategy it is the case that, when the number of robots is larger than $N = 3n_g$, the structure needs $N = 3n_g + 1$ so that when it moves forward and the front robot decouples, it will not topple into the void. After the front robot has disassembled, the second robot becomes the first robot and its assemble number in **build** mode changes from (2) into (1), the third robot (3) becomes (2) and so on. This demonstrates the use of dynamic variables.

The snapshots of three swarm robots crossing a 12cm void can be seen in Figure 4.19. They were successfully passing the void by autonomously assembling themselves to form a structure. The structure is long and stable enough to traverse the void.

The number of robots in the swarm can be increased up to eighteen. The results of how dynamic variables were applied can be seen in Figure 4.20. However, because this strategy and the obstacle avoidance only depend on the distance between robots to other objects, a number of robots got stuck in front of the structure. This problem was occurred because a robot could not move faster than expected to avoid a decoupled robot that was moving away from the structure.

The average of T_a and T_d can be seen in Table 4.7 and Table 4.8, respectively, and T_t in Table 4.9. As in case A, T_t in successful task in this case depends on T_s, T_a and T_o. And in unsuccessful task it depends on T_s, T_a, T_d and T_b. In general, T_t in this approach is longer than in case A because the arena is broader. It causes the robots to need a longer time to explore to locate the void or the target. The broader arena also affects the longer T_a because robots were spread less densely. This makes a robot need to move a greater distance to join the structure. By comparing the swarm size with respect to the void width at some points, the wider void reduces the wandering area, so that it can reduce the total simulation time. However, by comparing T_a and T_d for each swarm size, it is clearly shown that the larger the swarm size the longer the time needed to assemble and disassemble.

Although scalability has been successfully implemented, this approach still lacks certain characteristics. The fact that robots have *a priori* knowledge of the void width to be crossed has been set.

Table 4.7: The average assembly time (in seconds) to form a structure to cross a void (in cm) for different swarm sizes for case B.

Void	Swarm Size					
	3	6	9	12	15	18
12	38	92	155	204	268	315
24	29	85	139	200	250	311
36	32	69	137	175	249	297
48	23	77	105	188	243	277
60	21	78	110	150	233	280

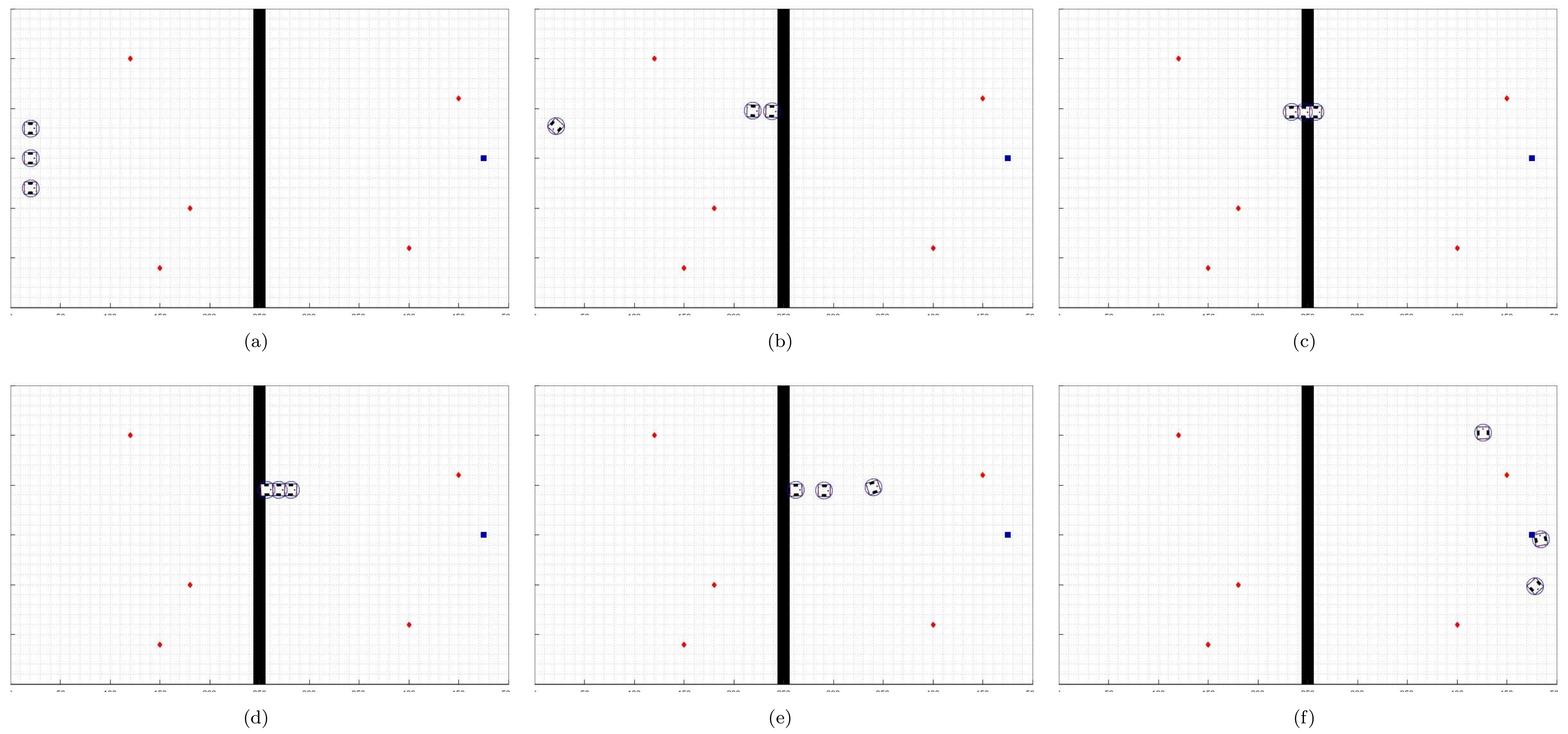


Figure 4.19: Three robots attempting to cross a one robot wide void. (a) Three robots were ready to wander. (b) The seed robot has been positioned orthogonal to the void while one robot has joined it to form a structure and another one is waiting to join them. (c) The structure is long and stable enough to cross the void. (d) The void has successfully been crossed. (e) Robots have decoupled from the structure to wander. (f) A robot has found the target.

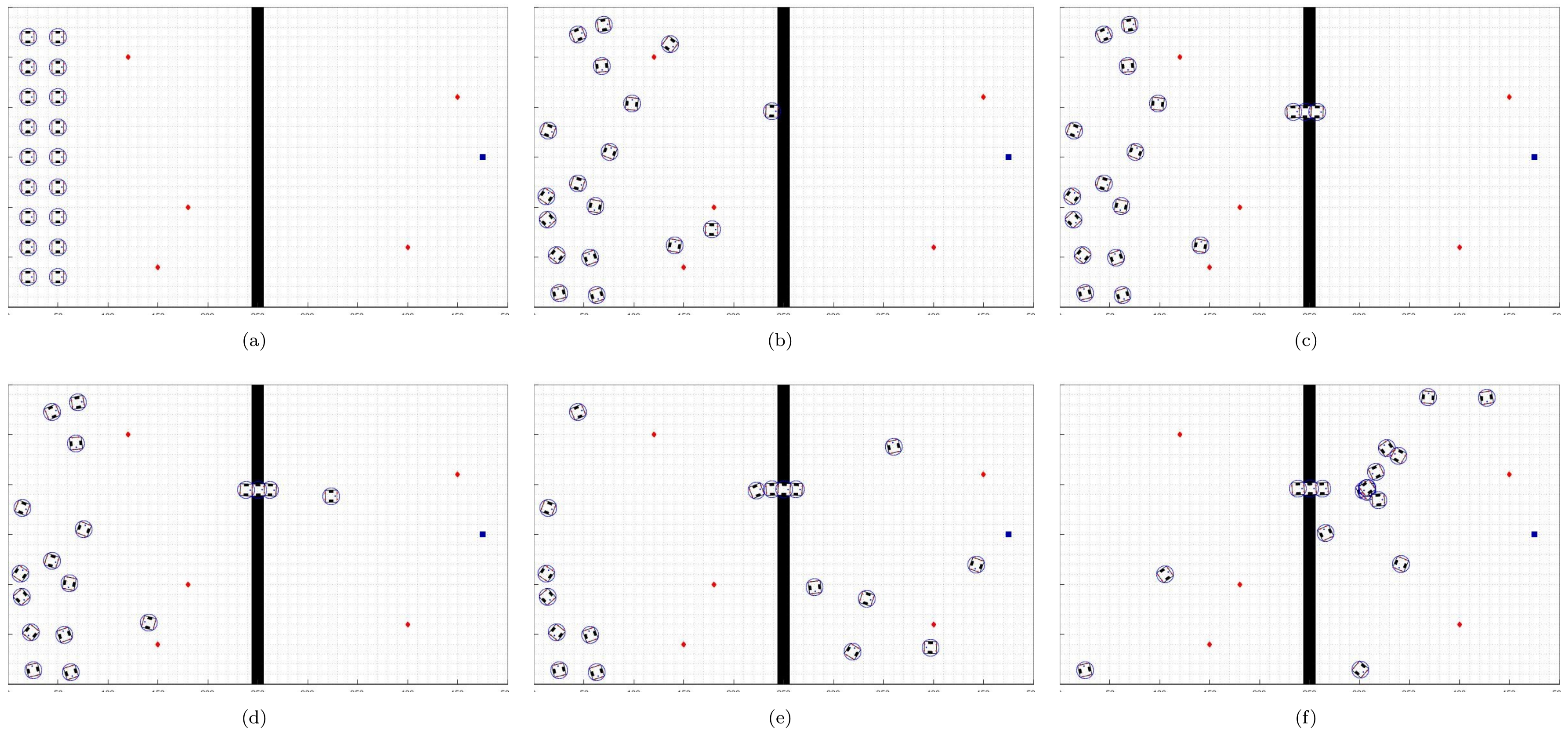


Figure 4.20: Eighteen robots attempting to cross a one robot wide void. (a) The initial position. (b) A robot has detected the void and it has become the seed. (c) The structure has been built by three robots, but it is waiting for another robot to join. (d) The front robot has dispersed while the structure is waiting for another robot to join. (e) A number of robots have wandered in the target zone while the structure keeps moving forward and other robots join it by sequence. (f) A number of robots got stuck because they could not avoid each other but, in the end, all the robots have successfully crossed the void and one robot has detected the target.

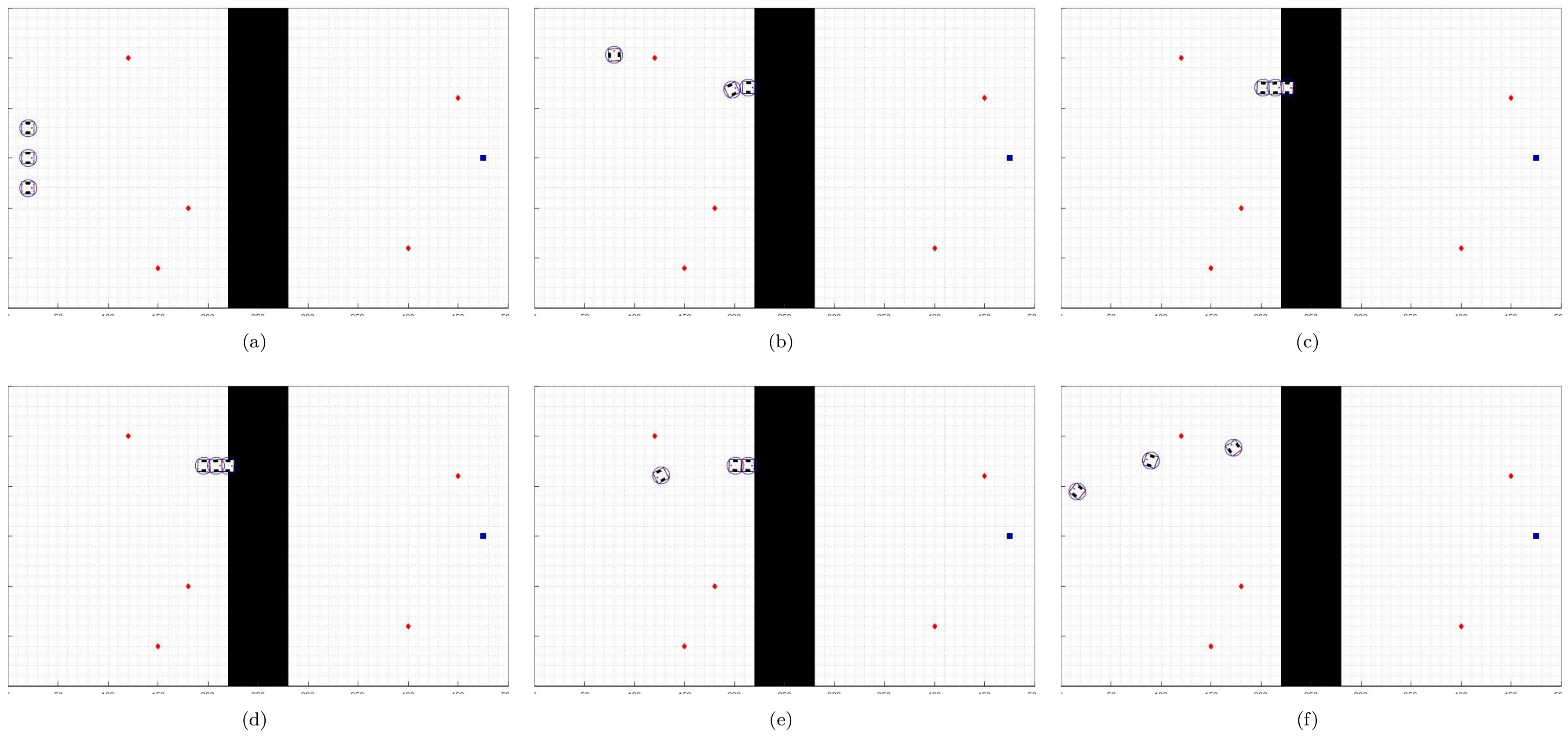


Figure 4.21: Three robots attempting to cross a five robot wide void. (a) The initial position. (b) A robot has joined the seed to form a structure while another one is waiting to be called. (c) The structure has moved forward until a robot is held by two other robots to keep stability. (d) Because the number of robots is not sufficient to reach the other side, the structure has retreated. (e) Robots have started to decouple. (f) All robots wander again in the initial arena for a period of time.

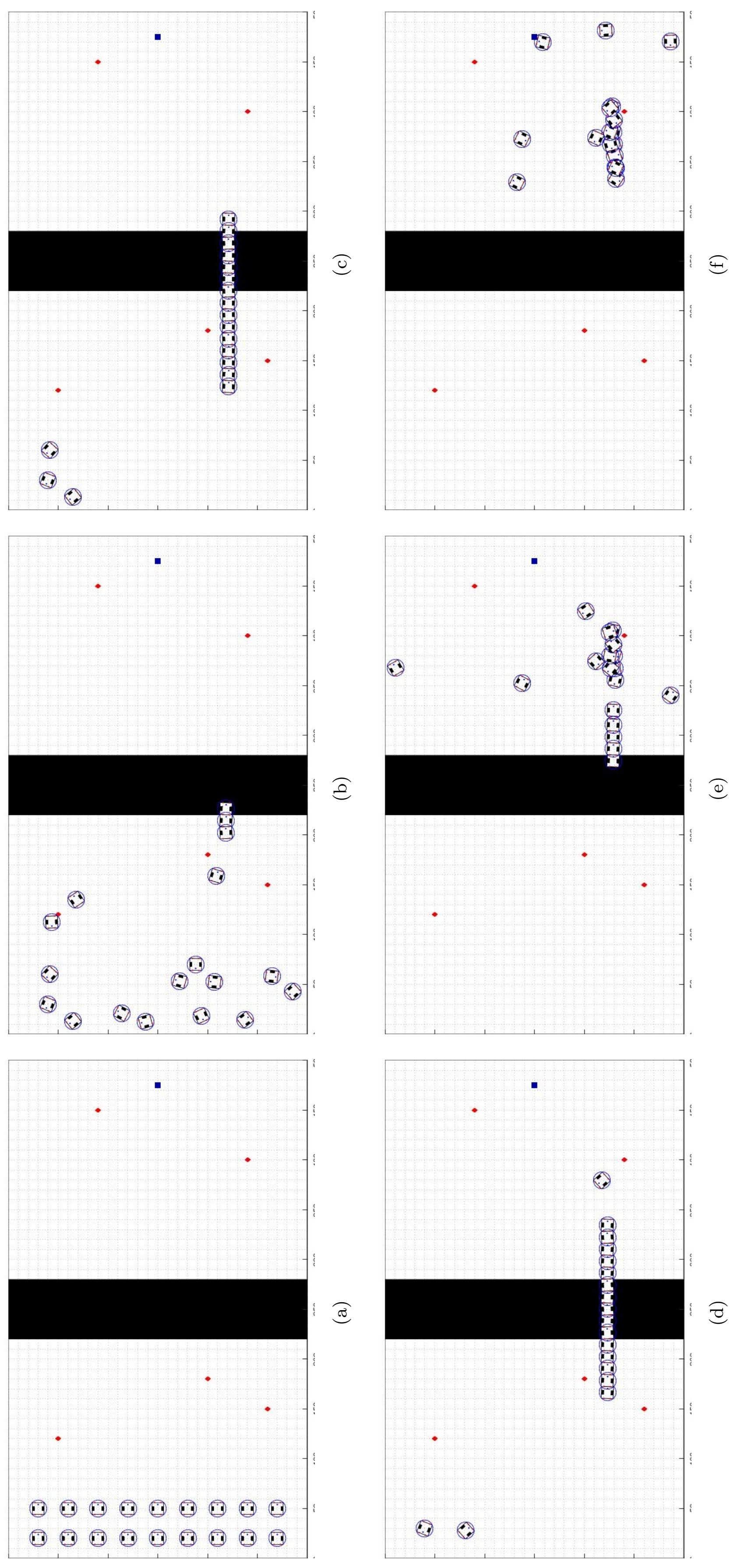


Figure 4.22: Eighteen robots attempting to cross a five robot wide void. (a) Eighteen robots are ready to wander. (b) A number of robots have assembled and are waiting for other robots to join the structure. (c) Fifteen robots have assembled and the structure can move forward to reach another area although it needs a long arena. (d) Fifteen robots remain assembled in the structure while one robot successfully wanders in the target zone and the others are ready to join the structure. (e) All robots have assembled and a number of them have decoupled and wandered in the target area, although some of them have got stuck. (f) One robot has successfully found the target.

Table 4.8: The average disassembly time (in seconds) when the structure fails to cross a void (in cm) for different swarm sizes for case B.

Void	Swarm Size					
	3	6	9	12	15	18
12	0	0	0	0	0	0
24	9	0	0	0	0	0
36	9	22	0	0	0	0
48	9	22	35	0	0	0
60	9	22	35	48	0	0

Table 4.9: The average simulation time (in seconds) to cross a void (in cm) for different swarm sizes for case B.

Void	Swarm Size					
	3	6	9	12	15	18
12	176	187	229	248	297	406
24	86	123	176	277	296	360
36	75	117	193	236	315	328
48	79	129	167	251	272	335
60	60	124	168	223	286	353

4.7.3 Case C

For case C, the swarm sizes are set from three up to eighteen swarm robots. Their initial positions have been set as fixed positions for every running simulation before they explore the initial zone to detect any size of void, which is 10cm, 20cm, 30cm, 40cm, or 50cm void. Thirty simulations were carried out for each swarm size and each void size.

Comparing all results obtained for frequency of seed robot ID, as can be seen in Figure 4.23, there is a trend that the robot becoming the seed depends on its initial position. When simulations were run for more than nine robots, the high probability of the robot becoming the seed is higher on the ID of 10 to 18, which initial positions were closer to the void. In summary, the closer the robots are to the void, the higher probability to become the seed robot. Yet, apart from this trend, all robots have the same opportunity to become the seed robot.

As proof that all robots do not have *a priori* knowledge of void position, Figure 4.24 shows that seed robot positions in y-coordinates are distributed according to the x-coordinates. Although the trend of seed robot positions is varied from fully distributed, slightly concentrated in the middle, the top, or the bottom of the arena, to random distribution, it shows that a robot is able to get closer to the void and has a chance to detect it at any location position.

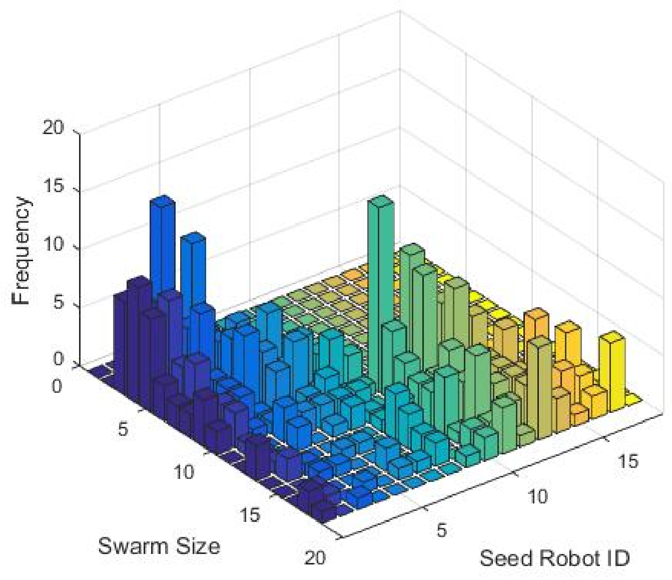


Figure 4.23: Frequency of a specific robot under fixed initial positions for three robot wide void to become the seed robot.

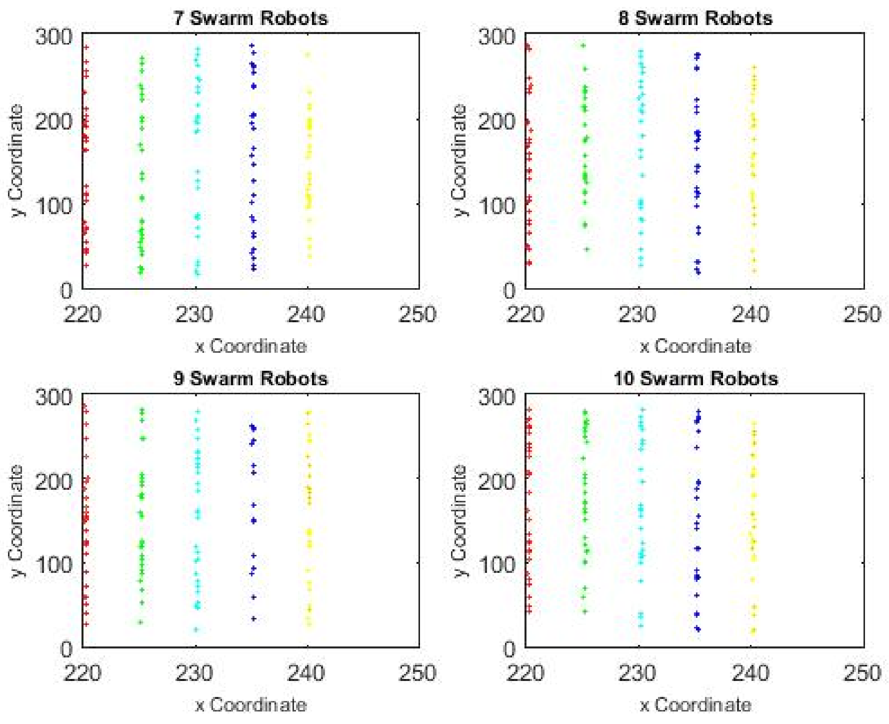


Figure 4.24: Positions of seed robots in fixed initial positions for seven to ten robots. x-coordinates 220, 225, 230, 235, and 240 show the void size of five, four, three, two, and one robot wide voids, respectively.

All robots also have the same opportunity to find the target once all of them have passed the void, as can be seen in Figure 4.26. Being the seed robot does not make it become

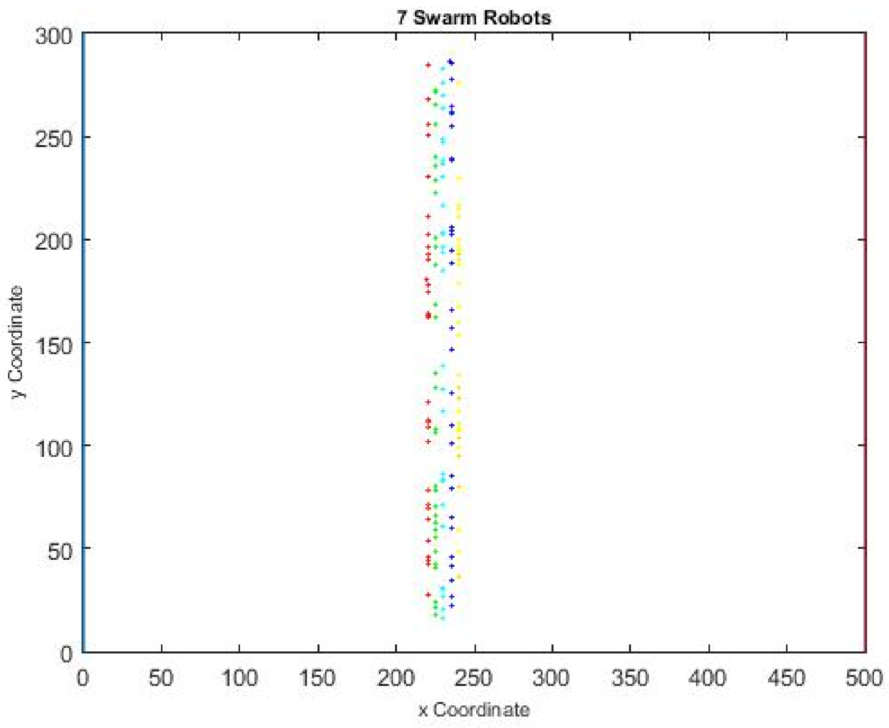


Figure 4.25: Positions of seed robots on the arena. They can be located at any y coordinate. The x coordinate depends on the void width; the wider the void the smaller the x coordinate.

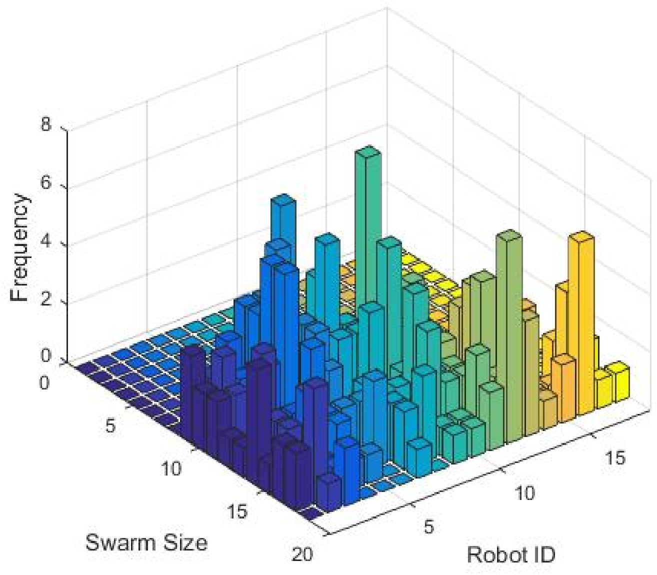


Figure 4.26: Frequency of robot ID detecting the target in fixed initial positions for three robot wide void.

the one to find the target, as shown in the figures, they are distributed randomly. Zero in the graphics is related to the swarm size. It means that, if the robot number is not enough, they failed to traverse the void and, therefore, they could not find the target, and there is no robot ID recorded.

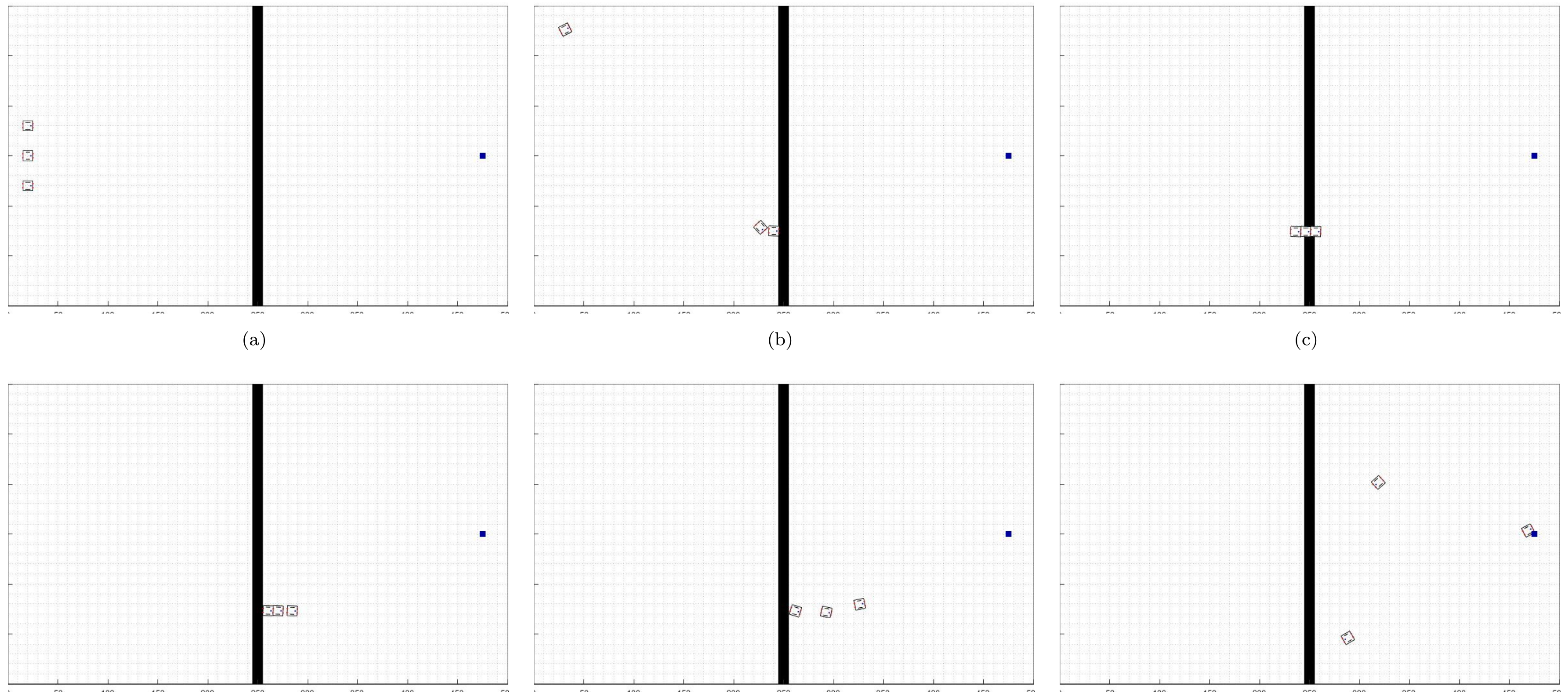
As the requirement to cross the void is $N = 3n_g$, where N is the required minimum robot number to traverse n-robot void to maintain the structural stability, if the void is one robot wide void size, then the minimum robot number to pass the void is three robots, as can be seen in Figure 4.27.

In the case of more than three robots in the swarm, the structure over the void is maintained to consist of three robots. When the front robot needs to disassemble, the structure will recruit another free robot at the initial zone to maintain the structural stability, as can be seen in Figure 4.28.

The same rule has also been seen for two, three, and four robot wide void cases, where a structure needs at least 3n robots in the swarms to traverse the void. When the number of robots is less than the requirement, the structure retreats and all robots disassemble to explore the initial zone again.

For the last case, when the five robot wide void has to be traversed by a swarm, a structure requires at least fifteen robots to maintain the stability over the void. A lower robot number causes it to fail to cross the structure and the target cannot be found, as can be seen in Figure 4.29. When the number is satisfied, it is able to accomplish the task, as can be seen in Figure 4.30.

From all recorded simulation times, Figure 4.31, it can be seen that the more robots in a swarm makes the simulation time longer. This time is influenced by the exploration at the initial zone, at the target zone, and back at the initial zone when the swarm fails to cross the void, and the construction or the retreat time. A discontinuity is occurred between swarm size of eight robots to nine robots because of the success or the failure in traversing the void. For a three robot wide void, a swarm consisting of eight robots or less failed to cross the void and, therefore, retreated back to the initial zone. A swarm consisting of nine robots or more is able to pass the void, so that it requires more time to move approaching the target area and searching for the target. This tendency also occurs in two robot wide void between five and six robots; in four robot wide void between eleven and twelve robots; and in five robot wide void between fourteen and fifteen robots. There is no discontinuity of time in one robot wide void because all swarm sizes are able to pass the void and find the target.



(a) (b) (c) (d) (e) (f)

Figure 4.27: Three robots attempting to traverse a one robot wide void in fixed initial positions. (a) The initial position of three robots to explore the initial zone and find the void. (b) A robot is joining the seed robot to construct a simple line structure while the other one is still exploring the initial zone. (c) The structure of three robots is moving towards the target zone and the front robot detects that there is a ground in front of it, so they continue to move forward. (d) All robots have reached the target zone and the front robot disassembles from the structure to wander at the target zone. (e) All robots have been decoupled and are ready to explore the target zone. (f) A robot has found the target.

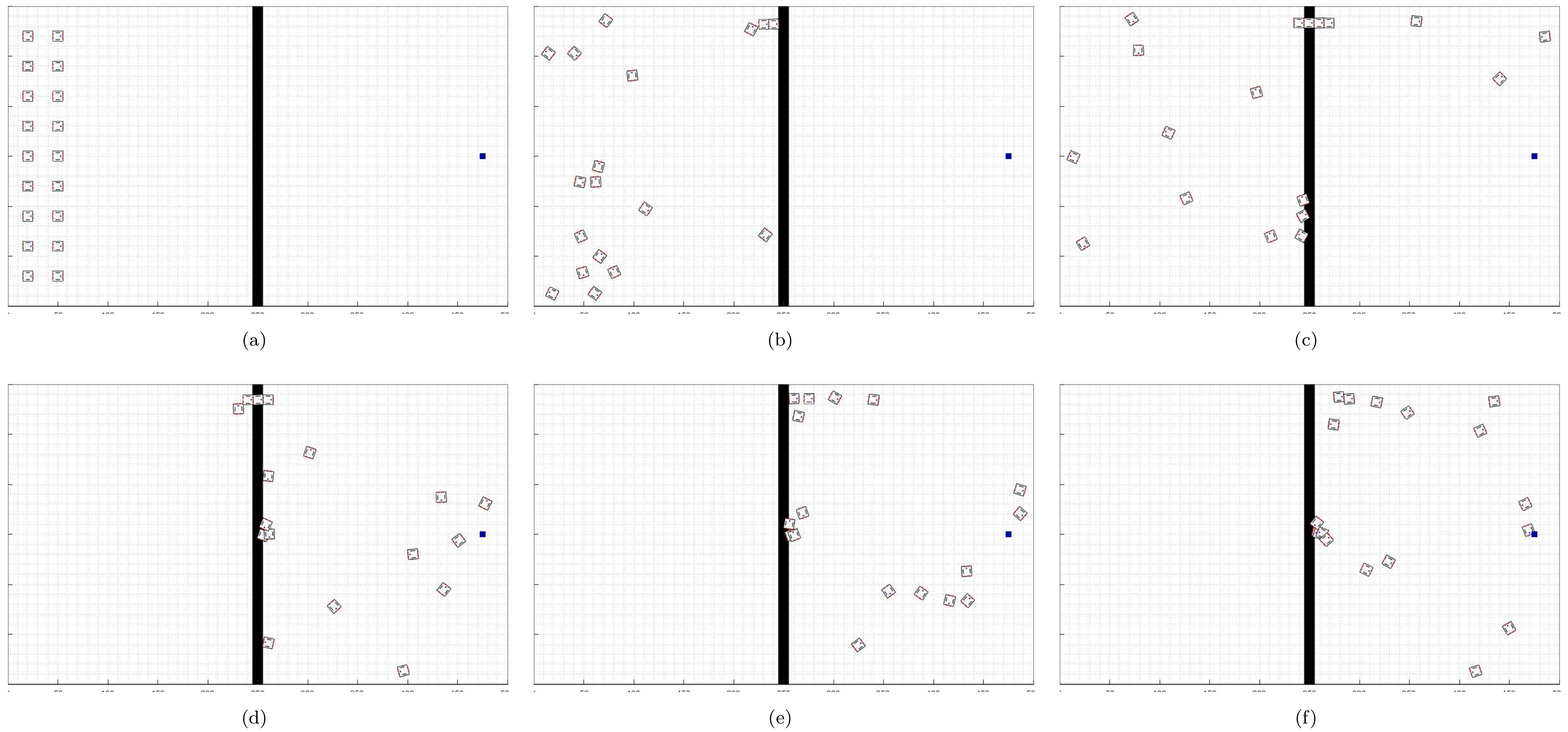


Figure 4.28: Eighteen robots attempting to traverse a one robot wide void in fixed initial positions. (a) Eighteen robots are ready to explore the initial zone. (b) Three robots are constructing the simple line structure while other free robots wander at the initial zone. (c) To maintain the structural stability while the front robot will disassemble from the structure, if there are still more free robots at the initial zone, the structure has to consist of four robots. (d) Robots reaching the target zone and decoupling from the structure are exploring the zone while the structure is still recruiting free robots at the initial zone. (e) All robots have reached the target zone and disassembled from the structure, ready to find the target. (f) A robot has found the target.

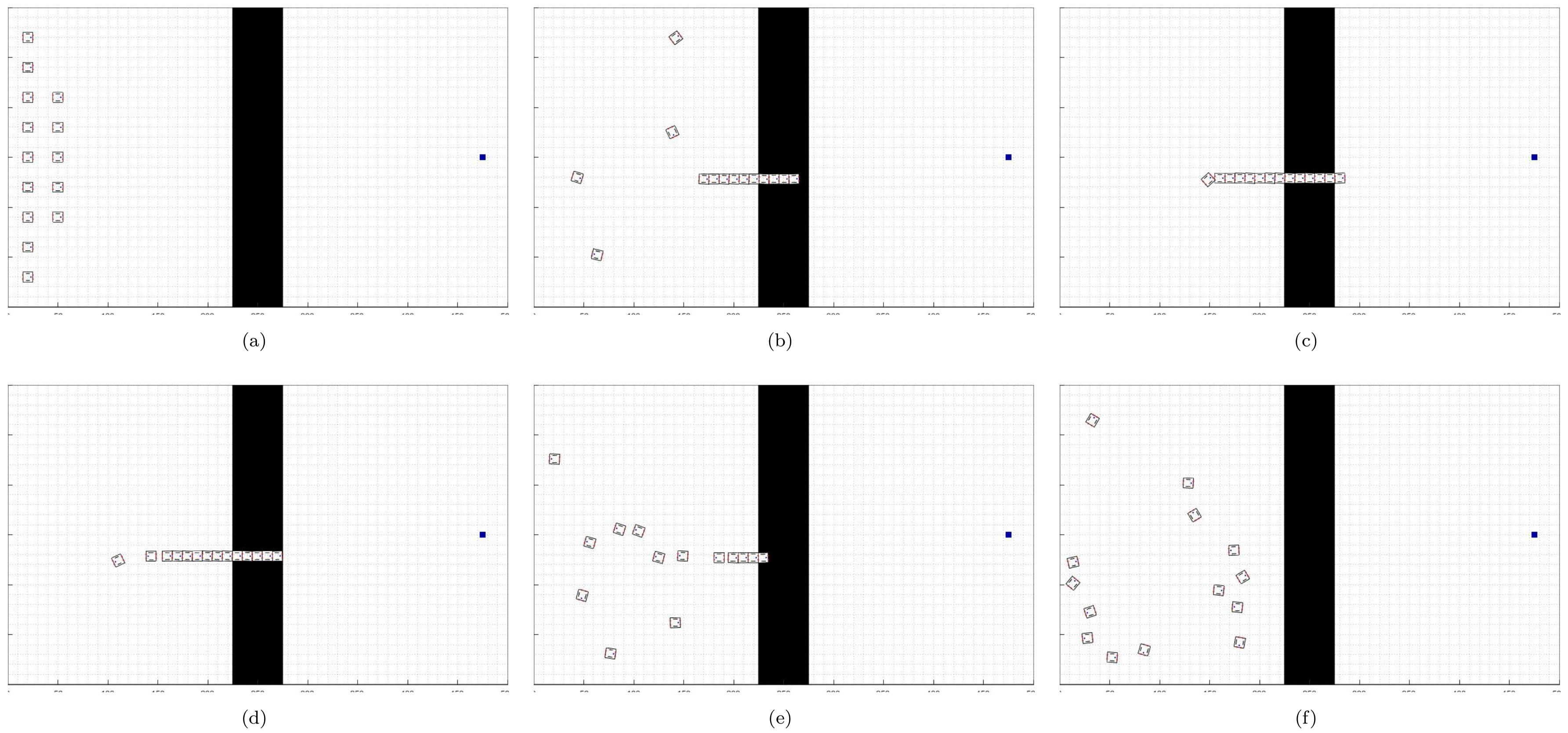


Figure 4.29: Fourteen robots failed to cross the five robot wide void in fixed initial position. (a) The initial position of the swarm. (b) The structure is moving towards the target zone while checking the target ground and recruiting free robots to join to the structure. (c) The last free robot is joining the structure and the front robot in the structure has reached the target ground, but the required number is not enough to maintain the stability. (d) Rear robots start to decouple and the structure retreats to the initial zone. (e) Robots in the structure continue to retreat and disassemble while decoupled robots wander back at the initial zone. (f) After a while, all robots stop moving.

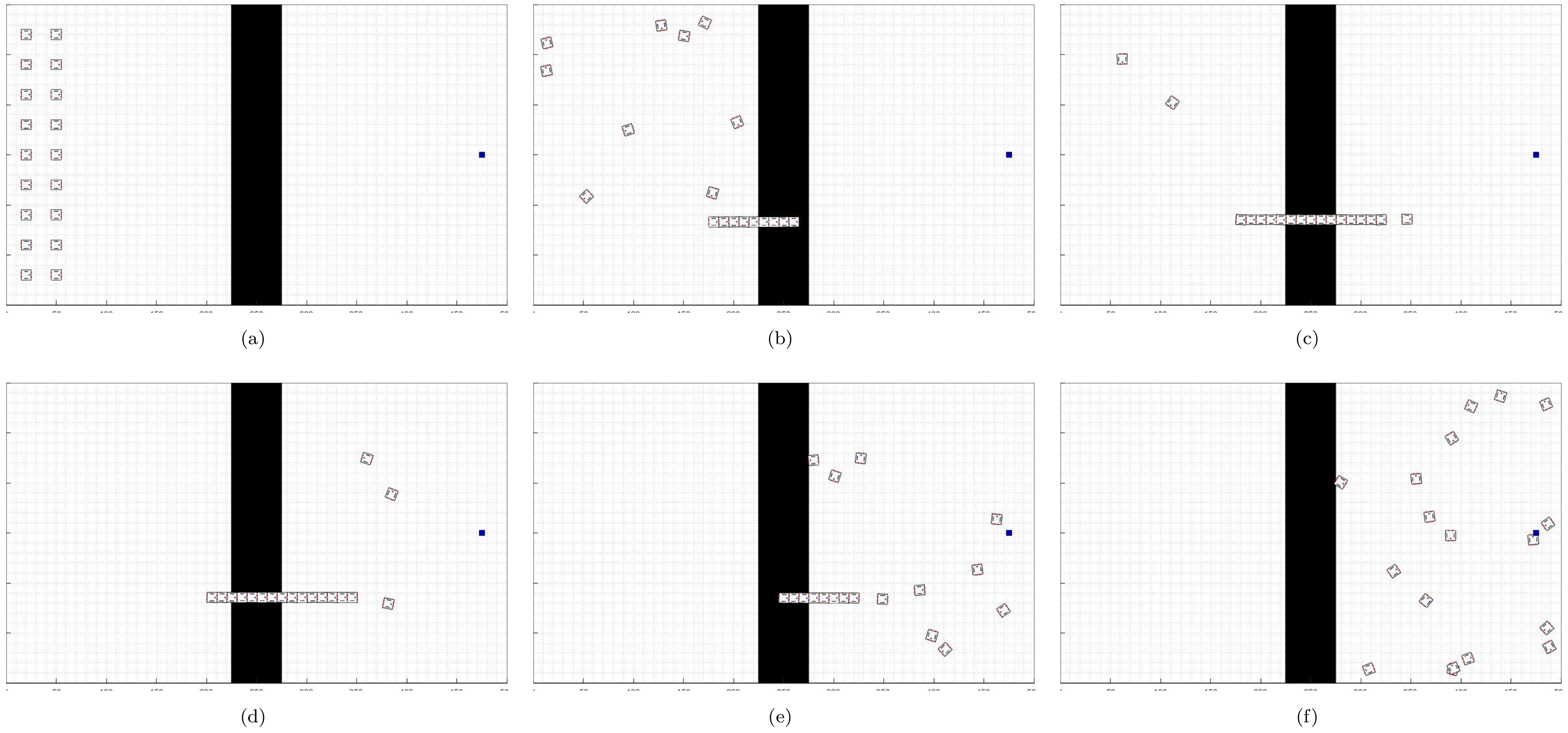


Figure 4.30: Eighteen robots are success to cross the five robot wide void in fixed initial position. (a) Swarm robots at their initial positions. (b) The structure is moving forward while recruiting other free robots and checking the presence of the target ground. (c) Fifteen robots are kept maintained in the structure for the structural stability when the front robot disassembles. (d) When there is no free robot at the initial zone, the whole structure is moving forward and the front robot decouples. (e) The process continues while decoupled robots explore the target area. (f) The target has been found by a robot.

A further time recording has been conducted for the exploration at the initial and the target zones, the assembly and moving forward and the retreat and disassembly. The exploration time at the initial zone is presented in Figure 4.32. There is a slightly decreasing time when the swarm size is larger at all void sizes. This means that more robots give a larger opportunity to find the void faster, so that the exploration time is reduced even though the robots are moving in a random direction.

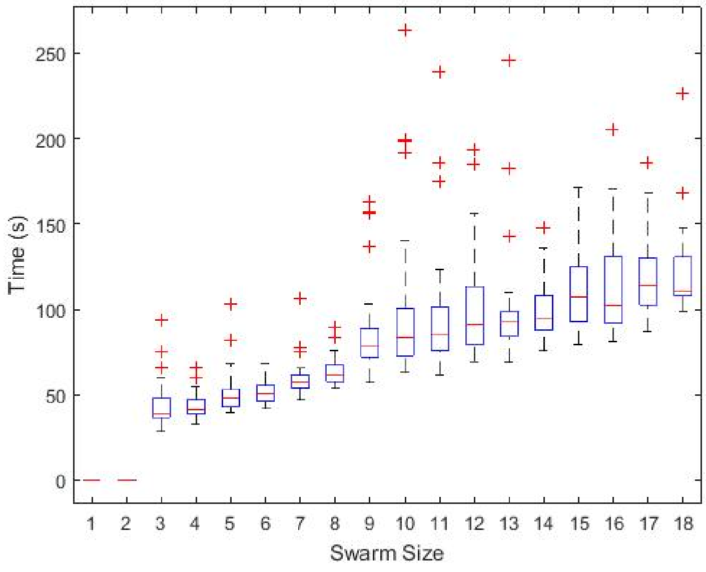


Figure 4.31: The simulation time for a swarm of a specific size to cross a three robot wide void. The results are an average thirty runs per swarm, and show the mean, 25% and 75%, in fixed initial position.

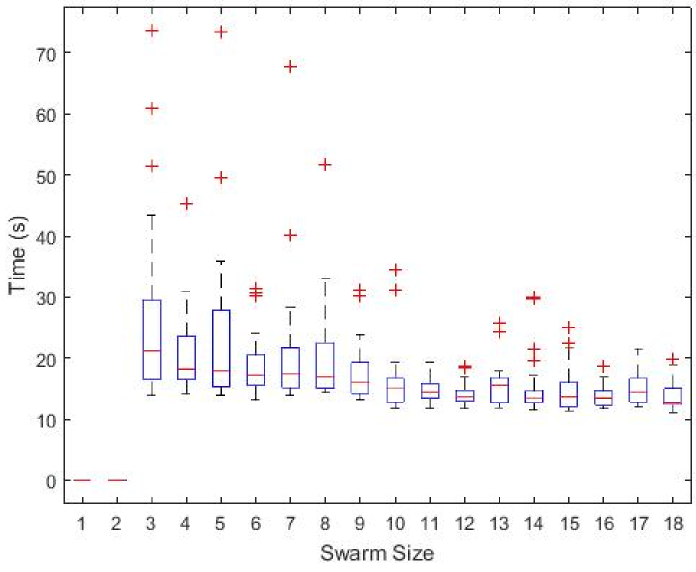


Figure 4.32: The exploration time at the initial zone for a swarm of a specific size to cross a three robot wide void. The results are an average thirty runs per swarm, and show the mean, 25% and 75%, in fixed initial position.

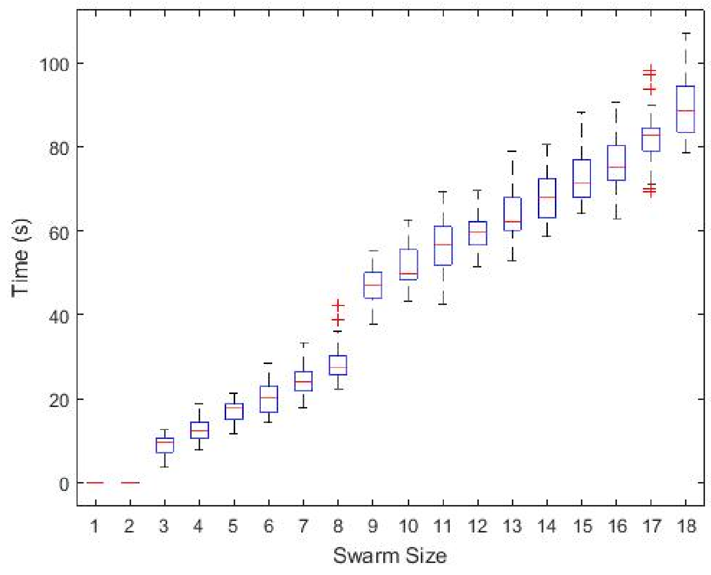


Figure 4.33: The assembly and moving forward time for a swarm of a specific size to cross a three robot wide void. The results are an average thirty runs per swarm, and show the mean, 25% and 75%, in fixed initial position.

For the assembly and moving forward time, as can be seen in Figure 4.33, it is clear that the more robots there are in the swarm makes a longer time for the swarm to construct the structure and move forward. The discontinuity at three robot wide void between eight and nine robots is caused by the success or fail factor of the swarm to cross the void. The failure in crossing involves the process of assembling and moving forward, yet it stops and changes to retreat when the number of robots is not satisfying the preset number to maintain the structural stability. The success crossing involves the whole process and the structure requires more time to pass the void allowing all robots to reach the target ground and disassemble to be ready to find the target. The factor of the area dimension does not affect the required time of the process because there is only slight differences between the recorded time in the five case studies. The area dimension was assumed to influence the time because free robots are still exploring the area and the distance between them and the structure could be affected by the zone size.

Retreat and disassembly time only occurs when the swarm fails to cross the void because of unsatisfying number according to the void size, as can be seen in Figure 4.34. In Figure A.38 in Appendix A, all swarm sizes succeed passing the void, so there is no recorded time for retreat and disassembly. For the case of two to five robot wide voids, there is a trend that the more the robots in the swarm and the larger the void size, the longer the time required to retreat and disassemble.

For the exploration time at the target zone when the swarm succeeds in passing the void as can be seen in Figure 4.35, it depends solely on the random movement of all robots, so there is no trend of increased or decreased graphic. However, the wide area of exploration causes the exploration to take a longer time.

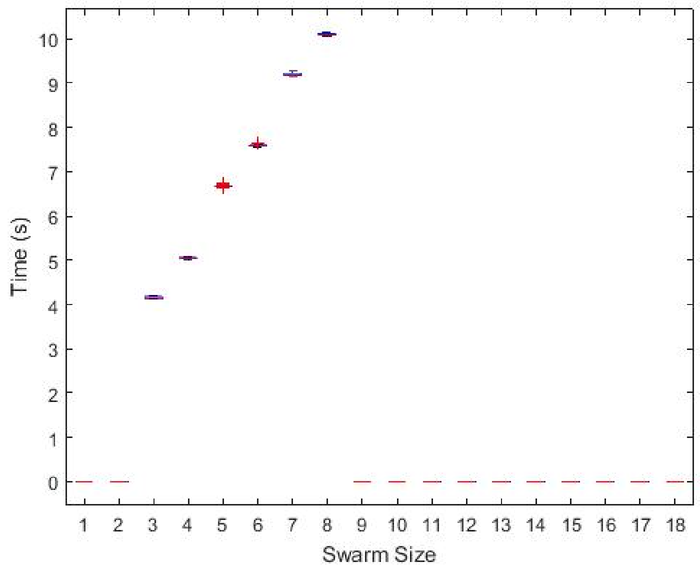


Figure 4.34: The retreat and disassembly time for a swarm of a specific size to cross a three robot wide void. The results are an average thirty runs per swarm, and show the mean, 25% and 75%, in fixed initial position.

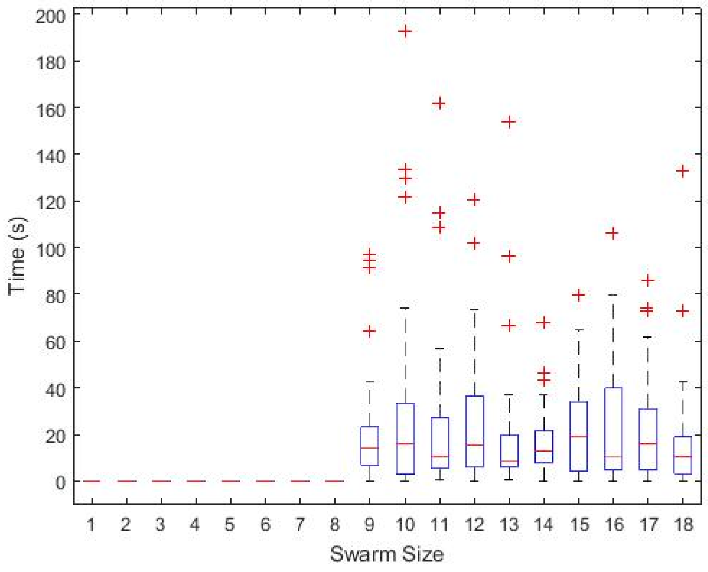


Figure 4.35: The exploration time at the target zone for a swarm of a specific size to cross a three robot wide void. The results are an average thirty runs per swarm, and show the mean, 25% and 75%, in fixed initial position.

4.7.4 Case D

For case D, the swarm sizes are set to three up to fifty swarm robots. Their initial positions have been set randomly for every running simulation before they explore the initial zone to detect any size of void. Thirty simulations were carried out for each swarm size and each void size.

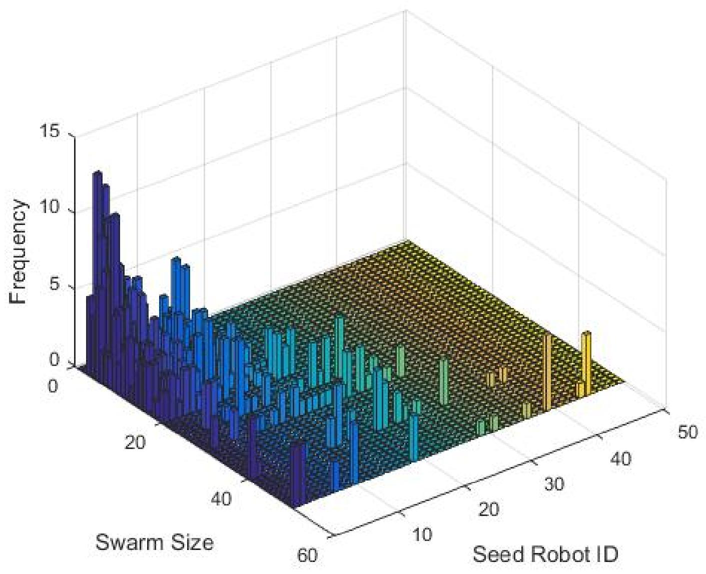


Figure 4.36: Frequency of seed ID in random initial positions for three robot wide void.

Comparing all results obtained for frequency of seed robot ID, as can be seen in Figure 4.36, the higher the swarm size results the more distributed of the robot IDs becoming the seed. Although, at some point, there is a high concentration of seed robots for certain IDs, all robot IDs have the same opportunity to become the seed, as the distribution of it still can be seen.

Similar to the fixed initial position case, the closer the robot to the void, the greater the opportunity of it to become the seed. However, at some point, when it turns away from the void, another robot can take the opportunity to become the seed, even though, initially, its distance was not the closest.

For the positions of seed robots in random initial positions, as can be seen in Figure 4.37, they have an opportunity to locate the void at any point, although there is a high concentration at some points. However, these distributions show that robots have no *a priori* knowledge of the void location.

Based on the results summarised in Figure 4.38, the distributed diagrams show that any robot can locate the target. Becoming the seed does not guarantee a robot can locate the target as, at the target ground, this robot is along with the others that are

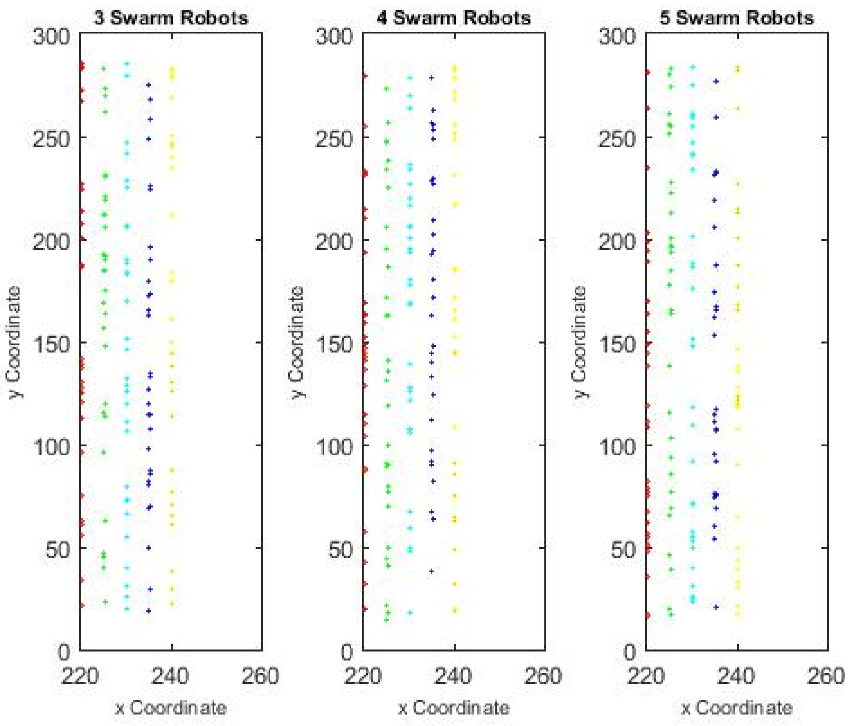


Figure 4.37: Positions of seed robots in random initial positions for three to five robots. x-coordinates 220, 225, 230, 235 and 240 show the void size of five, four, three, two and one robot wide voids, respectively.

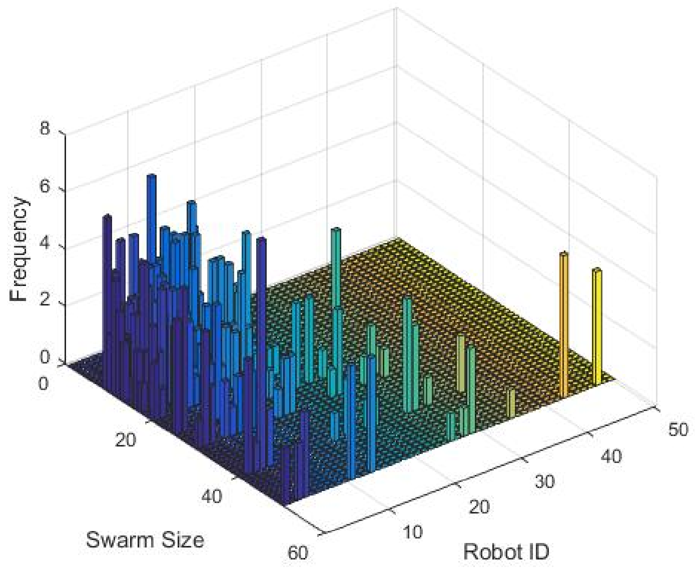


Figure 4.38: Frequency of robot ID detecting the target in random initial positions for three robot wide void.

wandering and exploring the ground. The opportunity to find the target becomes the same for all robots. Zero in the diagrams has three meanings: i) at some points, robots failed to cross the void, ii) no simulation has been conducted for certain swarm sizes, and iii) for certain IDs, they didn't find the target.

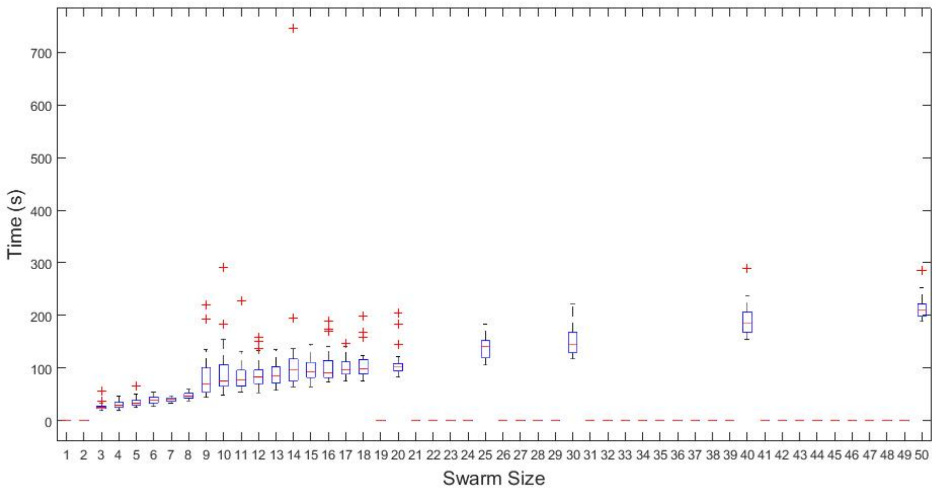


Figure 4.39: The simulation time for a swarm of a specific size to cross a three robot wide void. The results are an average thirty runs per swarm, and show the mean, 25% and 75%, in random initial positions.

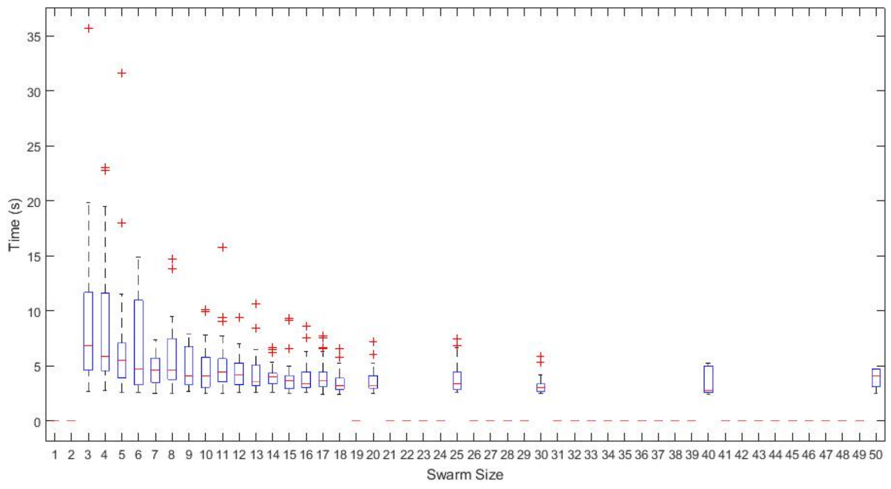


Figure 4.40: The exploration time at the initial zone for a swarm of a specific size to cross a three robot wide void. The results are an average thirty runs per swarm, and show the mean, 25% and 75%, in random initial positions.

Figure 4.39 shows the simulation time (not the real time) of the swarm conducting the task. The trend of it is linearly increased along with the increasing robot number in the swarm. At some point, there is a discontinuity in the diagrams such as eight to nine swarm size. This discontinuity is the boundary between the failed swarm to cross the void and the successful ones. When the swarm size was not enough to cross the void, they stopped the construction to retreat to the initial ground. That is the reason why the simulation time is shorter than it should be.

Figure 4.40 shows the exploration time at the initial zone. The exploration time is slightly decreased when the swarm size is larger. It shows that the more robots in the same dimension of arena gives the chance for them to find the void faster as the area coverage by the swarm is larger.

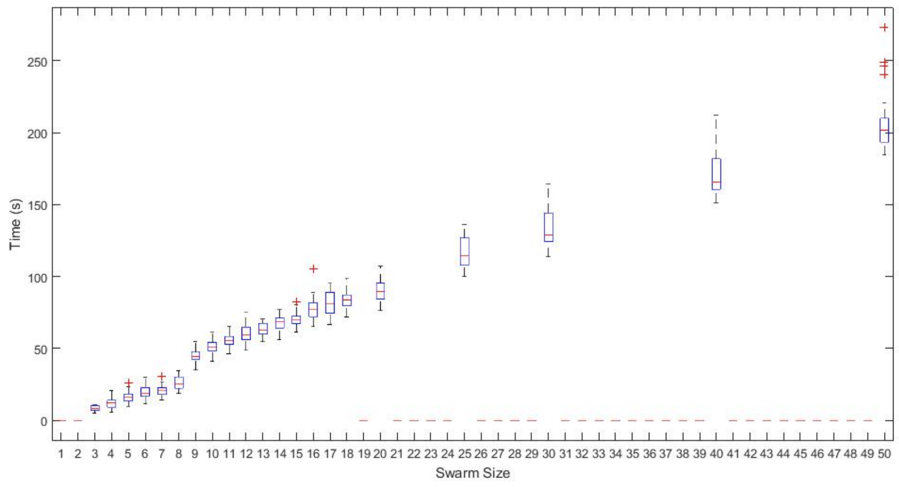


Figure 4.41: The assembly and moving forward time for a swarm of a specific size to cross a three robot wide void. The results are an average thirty runs per swarm, and show the mean, 25% and 75%, in random initial positions. The discontinuity at swarm size eight and nine indicates the change for a failed crossing attempt to a satisfied attempt.

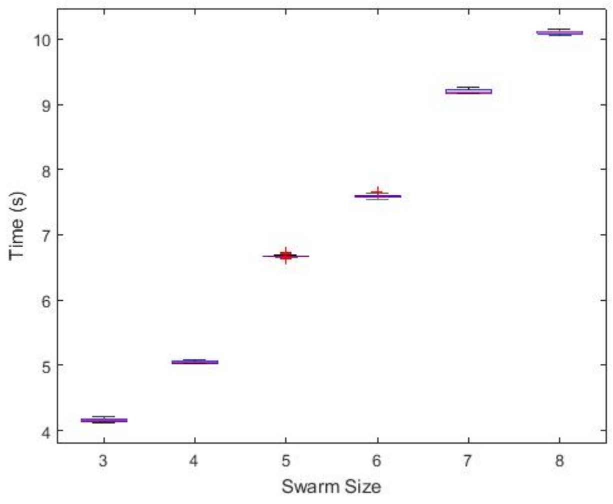


Figure 4.42: The retreat and disassembly time for a swarm of a specific size to cross a three robot wide void. The results are an average thirty runs per swarm, and show the mean, 25% and 75%, in random initial positions.

Figure 4.41 shows the construction time for random initial positions. The linearity of diagrams shows that an increased number of robots makes a longer time required to construct the structure for crossing the void. Discontinuities shown in the diagrams

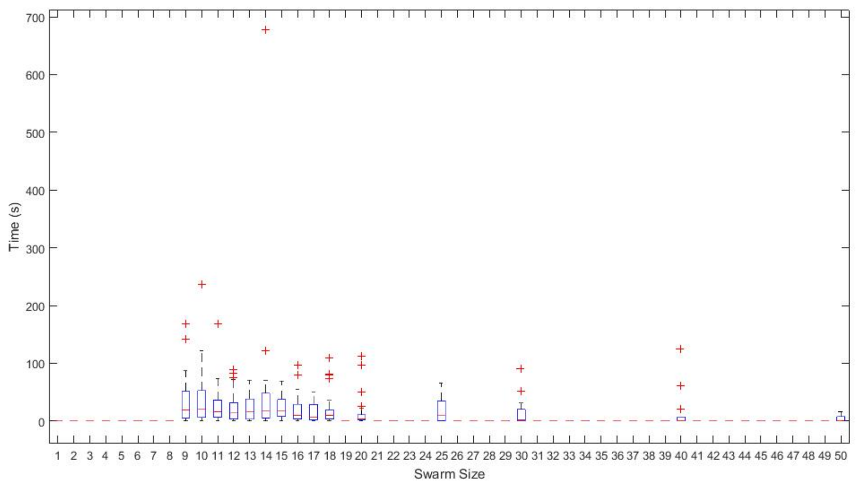


Figure 4.43: The exploration time at the target zone for a swarm of a specific size to cross a three robot wide void. The results are an average thirty runs per swarm, and show the mean, 25% and 75%, in random initial positions.

indicate the failed or successful void crossing related to the number of robots in the swarm and the void size. A swarm size not satisfying the required robot minimum number to cross the void will retreat to the initial ground, while the swarm with the satisfying robot number will continue the construction, reach and explore the target ground to locate the target. Figure 4.42 shows the retreat and disassembly time in random initial positions. The diagrams show the trend that a longer time is required to retreat and disassemble. Figure 4.43 shows the exploration time at the target zone in random initial positions. At some point, there is a trend that the larger robot number causes a shorter time to explore the area to locate the target.

4.8 Discussion

As the results have shown, the total simulation time depends on the swarm size. The larger the swarm size, the longer the simulation time took. The main factors that influence this time are the assembly and disassembly times required in the construction. However, this cannot strictly be generalised because, in some cases, the assembly time is longer, as free robots require a longer time to reach the structure, since their distance to the structure is large. The exploration time, both at the initial and target zones, can be generalised that the larger the swarm size, the less time to explore the zone, although in several cases it cannot be guaranteed, as robots have to avoid each other thereby increasing their time to wander.

Within the same dimension of the arena, it was expected that robots in a large swarm would detect the void or the target faster than the small swarm. However, because robots also have to avoid each other and other obstacles, the wandering time becomes longer. It was also expected that the longer and wider the wandering area the longer the time required by robots to explore the arena and to locate the void and the target. However, a longer arena is required when a longer structure is needed.

Each approach has both advantages and disadvantages. Case A solved the problem of robot collisions, because robot variables in collective behaviour tasks were kept stored when robots had to decouple from the structure. It also allows any swarm to cross any size of void because it has no *a priori* knowledge of the void size. However, it was affected by the limited number of robots in the swarm and the slow response in both simulation and writing the Matlab script. By applying scalability to case B, this problem was solved. However, robots in case B have *a priori* knowledge of the void size; therefore, a new setting has to be conducted to allow them to cross another size of void. Furthermore, there were still robot collisions when robots disassembled. Actually, this problem could probably be solved by applying some proximity sensors to each robot, although this would impact negatively on the simulation time. Another solution is by applying another simple rule for obstacle avoidance that will not affect the simulation time.

The advanced cases C and D combined the advantages of cases A and B. The swarm can cross any size of void without any *a priori* knowledge of the void size and location. Variables in constructing the structure were dynamic, as in case B. n-1 robot variables were shifted to n robot variables when the first robot of the structure decoupled at the target zone. The decoupled robot was no longer holding the construction variable. This strategy brings a faster response in writing the script and disassembled robots were directly able to explore the target zone. Additionally, to write the script was not in slow response, because the main script is divided into a number of simultaneous scripts. The strategy of having no *a priori* knowledge of the void size, as in case A, was also implemented for the advanced case.

Snapshots for simulations of case D were not taken, as they can be represented by simulations of case C. The difference between cases C and D is the simplicity of applying scalability where case C was not set to simulate it by setting fixed robots initial positions. The full mode of scalability can be applied and is proven by case D, where the user can input any number of robots in the swarm. To simplify the placement of robots, there is no *a priori* setting for the initial robot positions, so that any number of robots were randomly placed on the arena. Extra simulations were conducted for case D for twenty, twenty five, thirty, forty and fifty swarm sizes to strengthen the proofs of simulation time as well as assembly-disassembly and exploration times. These simulations also prove the scalability.

Evolved from initial cases to advanced cases, the simple line structure has addressed the void crossing problem. Compared to other approaches, applying bridge launching mechanism and dynamic variable for swarm robots make the swarm can achieve the longer span to reach the other ground separated by a void, up to five robot wide void, whereas other approaches only achieved up to two robot wide void. The swarm is also able to cross any size of void as long as its number satisfies the void width. In accordance to cross any size of void, this approach also considers scalability where any number of robots can be added into the swarm, and all robots can involve in constructing the structure with a specific rule: if the number of robots in the structure satisfies with the correct multiple of the void width, the structure moves forward to give a chance for recruiting a robot at the rear and releasing a robot at the front. In addition, there is only a few approaches that consider scalability for self-assembly task in swarm robotics. The robustness of this approach is guaranteed by an addition feature where other approaches have not considered: if the number of robots in the swarm does not satisfy with the void width, the swarm can retreat to the initial zone and all robots disperse. With this feature, a loss or damage of robots can be prevented. However, there is a limitation of the span as a slight deflection must be happened in the structure and it makes the structure difficult to reach another ground. Therefore, a more complex structure that involve all sides of robot body is proposed in Chapter 5 to strengthen the connection and rigidity, reduce the deflection, and achieve the longer span to cross a wider void.

Chapter 5

Crossing a Large Void

In the previous chapter, the structure's purpose was only to bridge a small void, and it has a span limitation because of the deflection that can trigger a fractured structure; therefore, a more complex structure is required to bridge a large void. It has to maintain structural stability while the construction is being conducted. It should also be able to prevent any large deflection causing the structure to fracture.

The basic algorithm of large void crossing is similar to the one of small void crossing, using the mechanism of the bridge launching, but it is expanded into a 3D structure. The final structure of the small void crossing is only a simple line using only a front-rear connection mechanism. The structure is extended to only one direction, the x direction, or it is only a one dimensional structure. For a large void crossing, the final structure is extended to x, y and z axes. The extension in z direction maintains the stronger connection between the swarm robots in the structure, and the one in y direction maintains it and helps in maintaining the structural stability when the construction and the movement of free robots to the target zone are conducted.

The discussion in this chapter consists of the arena used for the simulation, the robot model and dimension, the scenario, the assumptions taken, the algorithms themselves, the results obtained, and the discussion.

5.1 Scenario

The scenario applied to the swarm for a large void crossing is similar to that of a small void crossing. All the robots in the swarm are considered to be identical. They have to cross the void first before reaching the target zone to locate the target. All robots have no *a priori* knowledge of the void size nor its location so they will find it while they explore the initial zone. Once a robot locates the void, it is settled as the seed to initiate the structure construction. The seed then recruits the closest free robot to construct a structure. When the constructed structure has maintained its structural stability and it is possible for the front robot to disassemble to explore the target zone, the process of recruitment and maintaining the shape of the structure are continued until there is no free robot at the initial zone. Then, searching for the target will take place when all robots have reached the target zone.

The difference between this approach and the small void crossing case is that the swarm will not retreat to the initial zone as it is set to contain a large number of swarm robots and its number is enough to cross the void. When the structure recognises a certain width, the construction changes into the more complex structure in a two dimensional extension for a stronger connection. This structure applies the connection mechanism at the front-rear and right-left sides of each robot. A similar rule is applied when they meet another certain width, but they have to construct a three dimensional structure for a stronger connection and more stable structure. This structure applies to not only front-rear and right-left sides of connection mechanism, but also the top-down side to support the vertical connection.

The scalability is not applied in this scenario as the deployed swarm is already a large number of robots. What is to be seen in this proposed algorithm is the adaptability of the swarm to cross a wider void to reach the target at the opposite zone. Robots in the swarm also do not have *a priori* knowledge of the swarm size. The construction will stop when the last robot at the construction sequence recognises there is no free robot wandering at the initial zone; therefore, they cannot recruit any more robots, and then the construction concentrates on moving robots in the structure to the other side of the ground.

5.2 Robot Model

The robot model used for large void crossing is also similar to the model for small void crossing, specifically the model for algorithms C and D for small void crossing (Section 4.6.3). Its shape is a square with the dimensions of $10cm \times 10cm$. It has a blue dot as the indicator of the front side. It also has two red dots at each front and rear sides as the marker of front-rear connectors. The model can be seen in Section 4.2 where the arrow shows the direction in which the robot moves.

5.3 Arena

The arena dimension for large void crossing has been determined after considering a number of aspects, such as the void width to cross, the number of robots in the swarm, and the estimated length of the structure when it is being constructed. Considering the rule of the algorithm and the limitation of structure span, the void to cross is limited to 10cm-500cm wide. The arena dimension of $2000cm \times 1000cm$, as can be seen in Figure 5.1, is chosen by considering the swarm size and the length of the structure. Subtracting the half width of the arena by the half of the void width, it gives a $750cm \times 1000cm$ area. Based on this coverage and the length of the structure in construction, this arena is able to accommodate the swarm in the construction process.

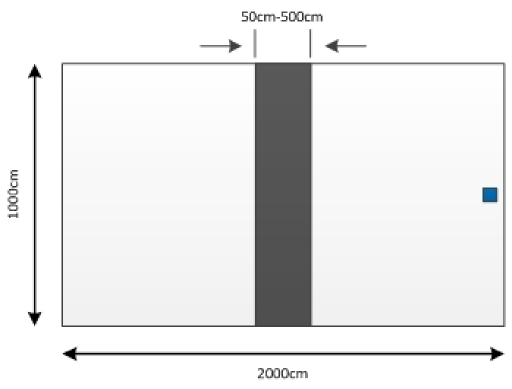


Figure 5.1: The arena that was used for the simulation of large void crossing. The dimension is larger than the arena for small void crossing in order to accommodate the larger void and increased number of robots in the swarm.

5.4 Assumptions

The following assumptions were taken related to the physical robot model, the structure constructed and the features of individual robots, and are based on those given in Section 4.2, and several assumptions mentioned in Section 4.3.

1. The initial position of the swarm is random identical to point 5 of Section 4.3, but the size of the swarm is fixed.
2. All the following assumptions of Section 4.3, 1, 3, and 7 to 13, are applied to complex structure algorithm.
3. Communication between robots is limited to three types.
    (a) Broadcast. This is used when a robot detects the void for the first time and initiates the construction, recruits other robots to join the structure, detects the opposite ground, detects the target and informs other robots to stop the operation, let other robots in the structure know its variable to decide for recruiting, moving forward and backward, and allow the front robot to disassemble from the structure to explore the target zone.
    (b) Intra robot communication. This is used when the recruiter guides the direction for the recruited robot to move closer to its rear to join the structure.

   (c) Infra Red. This is used for a coupling mechanism as both recruiter and recruited robots have to pair their connectors.

4. As robots have connectors at front-rear, left-right, and top-down sides to construct 3D structure by assembling themselves with other robots, the coupling configuration permits not only a structure one robot wide like the simple line structure. In reaching the opposite ground which maximum void width is thirty robot wide, the coupling configuration permits up to eight robot wide and four robot high.

5. When recruiting robots at the rear starting from eleven robot void wide in the simulation that is still being developed, the structure can recruit several robots to form a row instead of recruiting robot one-by-one. Thus, a number of robots move towards the rear of the structure at the same time and can also connect to the structure at the same time.

6. A simple obstacle avoidance is applied for the simulation so that, at some point, collision can still happen.

5.5 The Complex Structure Algorithms

Similar to the simple line structure, the complex structure algorithm has two main tasks: the distributed individual tasks (exploration mode), where all robots are exploring the environment by using Levy walk to search for the void and then search for the target; and the collective behaviour task (construction mode) where robots have to construct a more complex structure to help them in reaching the opposite zone by bridging the wider void. Note that there is no retreat mode in this complex structure algorithm, so that there are only two exploration modes: initial exploration and target exploration.

If, in the simple line structure algorithm, the state of robots is only expressed by positive and negative numbers to differentiate the state of initial, target and retreat explorations, and the construction task, the state used in this complex structure algorithm is more detailed. Each task has its own variable and, in collective behaviour tasks, there are different variables to differentiate the robot's position in the structure, as can be seen in Table 5.1.

All robots initially have the *start* variable set to state 1 to indicate they are starting the exploration at the initial zone to find the void. When a robot detects the void, the *start* variable becomes 0 and its *coreL* changes from 0 to 1. The next robot joining the structure will change the state of *coreL* from 0 to 2, and so on. When the first robot detects that the void is more than five robot wide, the structure starts recruiting a robot to join at the right side of the structure with variable *coreR*. If robots at the first row haven't detected the opposite landing after the structure has measured the void is ten robot wide, the structure starts recruiting robots to connect at layer X and side Y, both

Variable	**State**	**Explanation**
start	0	This state indicates that a robot does not conduct the initial exploration anymore.
	1	All robots initially hold this variable and state as the sign they have to start the exploration to find the void at the initial zone.
coreL	0	A robot is not at the core-left position in the structure.
coreL	n	A robot is at the n^{th} core-left position in the structure.
coreR	0	A robot is not at the core-right position in the structure.
coreR	n	A robot is at the n^{th} core-right position in the structure.
layerXside(L/R)Y	0	A robot is not at the layer x and side (L/R) y in the structure.
layerXside(L/R)Y	n	A robot is at the layer x and side (L/R) y in the structure.
OlayerXside(L/R)Y	0	A robot is not at the layer x and side (L/R) y in the structure and on the target zone.
OlayerXside(L/R)Y	n	A robot is at the layer x and side (L/R) y in the structure and on the target zone.
opposite	0	The state indicates that a robot is still at the initial zone or in the structure.
	1	This state indicates that a robot is already at the target zone and ready to find the target.

Table 5.1: Variables and states used in simulation of algorithm for complex structure.

at left and right sides. When the structure has to maintain its stability at both grounds, robots that are reaching the opposite ground at the core position still hold *coreL* or *coreR*. But robots at layer X and side Y will hold *OlayerXside(L/R)Y* to mark they are on the opposite ground. Once a robot disassembles from the structure and explores the target zone, the variable *opposite* changes from 0 to 1.

5.5.1 Distributed Individual Tasks

The distributed individual tasks for complex structure only consists of two tasks: i) initial exploration and ii) target exploration.

5.5.1.1 Initial Exploration

All robots have state 1 for variable *start* state, and 0 for other variables when the simulation begins. All of their initial positions are random. They have to find the void by walking in many directions using Levy walk. When a robot detects the void, it becomes the seed, the *start* changes to 1, and it informs other robots to avoid the void and be ready to be recruited to construct a structure. The flowchart is the same as the initial exploration of simple line structure in Section 4.6.1.1.

5.5.1.2 Target Exploration

When a robot disassembles from the structure, the state of the variable *opposite* changes from 0 to 1. When all robots have disassembled, the exploration to find the target begins. They move in any direction while avoiding the void. Once the target has been found by a robot, the task is complete. The flowchart of target exploration is the same as that of simple line structure in Section 4.6.1.2.

5.5.2 Collective Behaviour Task: Complex Structure

In constructing the complex structure to cross a larger void, the swarm still uses the same launching mechanism as well as the simple line structure construction. However, the construction is advancing, as it is allowed to expand the connection in the horizontal to both left and right sides, and in the vertical into several layers.

Figure 5.2 shows the step of the formed final structure, where there is a fixed rule to construct every structure, from the single line structure for one to five robot wide void, until the final complex structure for twenty six to thirty robot wide void. Figure 5.3 shows the cross section of the final complex structure consists of two sides, left and right, and four layers. The flowcharts given in Figure 5.4, Figure 5.5, Figure 5.6, Figure 5.7 and Figure 5.8 with all the crossing of a void up to thirty robot wide.

The change of structure at a five robot wide void which reason is the occurred deflection will be proved at Chapter 6. At the next step, the simple line structure will be strengthened by another line at its right and both lines form the core of the complex structure. There is also a requirement that the number of assembled robots have to be larger than the number of robots over the void to keep the balance of the structure when the structure has not yet found the opposite ground. Instead of only adding the robot number to the left and right sides of the core, robots can also be added vertically. Besides strengthening the connection to prevent the bending, this strategy is also reducing the space occupancy on the ground. Therefore, at some point layer by layer is added to the structure covering the inner layer, forming an arch-like structure if the side perspective is taken, to ease the process of connection in vertical.

a) Figure 5.2 (a), the simple line structure only allows a one robot wide structure and it only works for the void that is one to five robot wide void. It should be noted

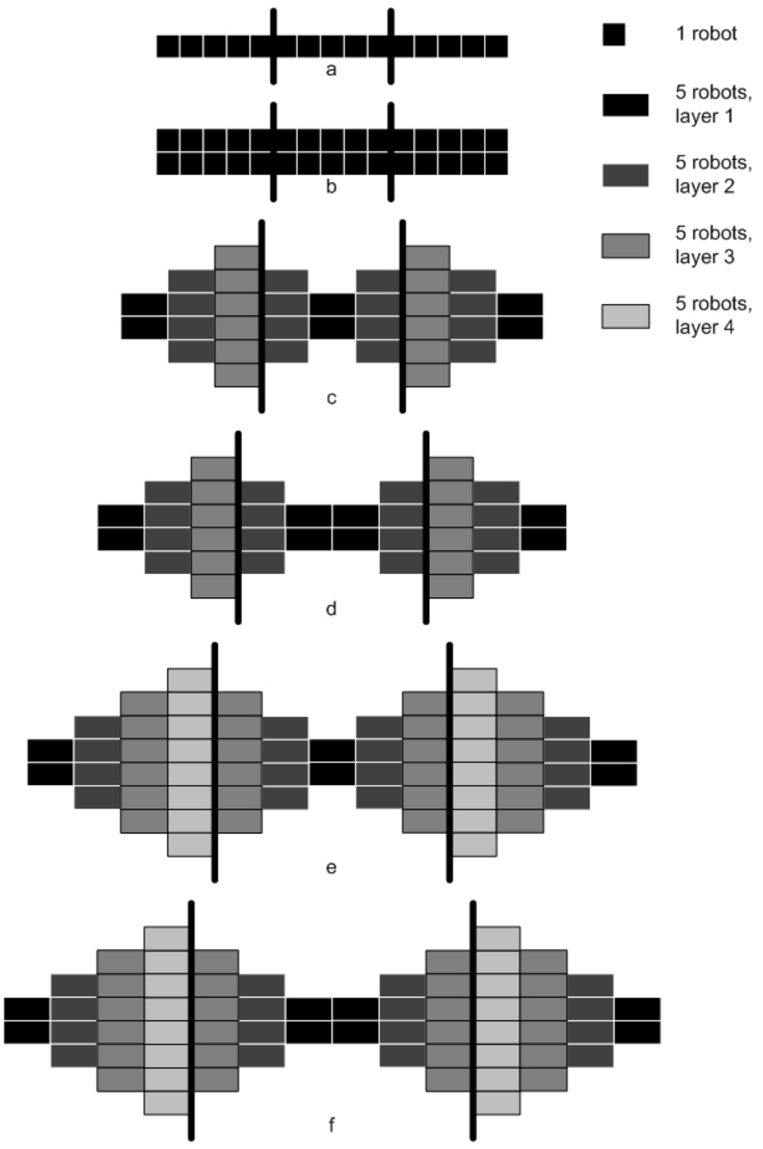
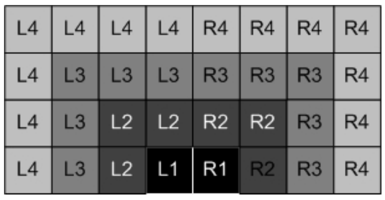


Figure 5.2: The top view of the complex structure. (a) The simple line structure for one to five robot wide void. (b) Another line join the structure at the right side for six to ten robot void wide. An arch-like structure is constructed for (c) eleven to fifteen robot wide void, (d) sixteen to twenty robot wide void, (e) twenty one to twenty five robot wide void and (f) twenty six to thirty robot wide void.



L4	L4	L4	L4	R4	R4	R4	R4
L4	L3	L3	L3	R3	R3	R3	R4
L4	L3	L2	L2	R2	R2	R3	R4
L4	L3	L2	L1	R1	R2	R3	R4

Figure 5.3: The cross section of the final complex structure, where L1 and R1 are at the core of the structure surrounded by addition robots in layers.

that the mechanism of construction is begun when a robot detects the void and it informs other robots to avoid the void. After it has settled, facing perpendicularly to the edge of the void, it starts to recruit other robots to build the structure. When the recruited robots number is larger than the seed, the structure moves forward until a robot is over the void. The process of recruiting and moving forward continues until the structure has settled into the relation of $n_p = n_g = n_o$. Then robots can be delivered to the target zone by recruiting a robot at the initial zone, moving forward, and releasing a robot at the target zone. After there is no robot to recruit, the structure will move forward and release robots at the target zone.

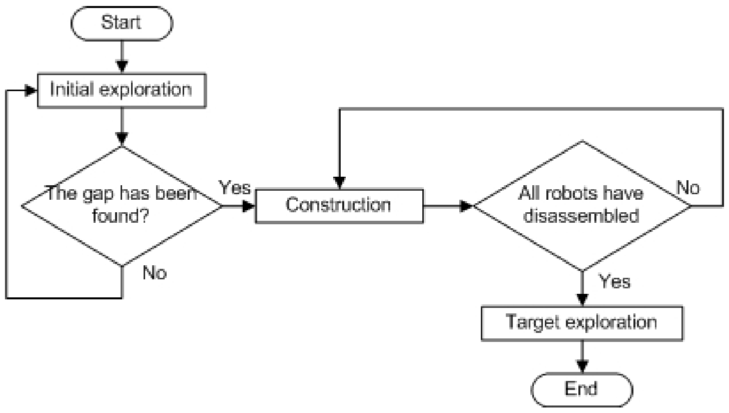


Figure 5.4: The relationship between exploration algorithms and construction algorithm in the whole simulation.

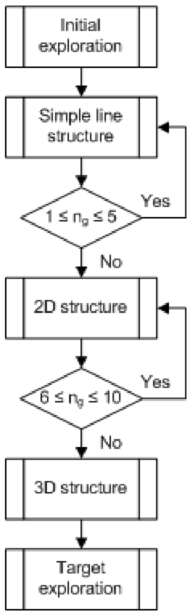


Figure 5.5: Steps of construction of complex structure.

b) The structure of Figure 5.2 (b) is built when the void is six to ten robot wide void. When the front robot has not detected the landing after five robots are over the void, the structure recruits robots to join the structure at the right side until the length is the same as the first line. After that, the structure continues recruiting robots to join both the left and right lines for each two robots and then moves forward one robot length over the void. The recruiting continues until $n_p = n_g = n_o$. Then similar to the simple line structure, the structure recruits a row of robots at the initial zone, moves forward a row, and then releases a row at the target zone. After there is no robot to join the structure, it will move forward and release robots at the opposite zone.

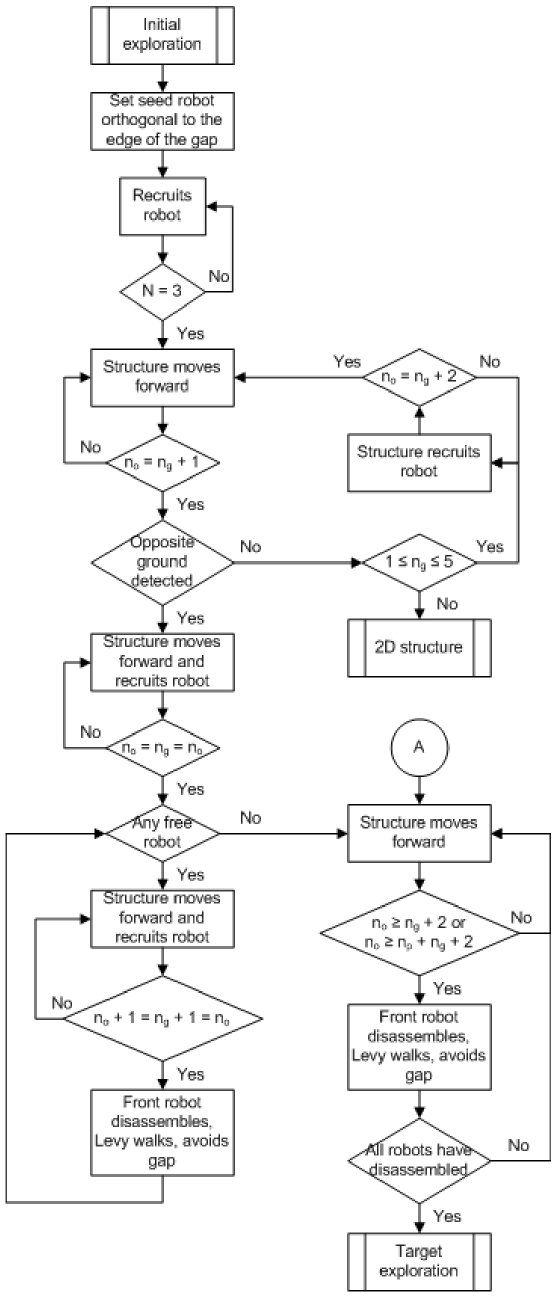

Initial exploration
Set seed robot orthogonal to the edge of the gap
Recruits robot
N = 3
No
Yes
Structure moves forward
Yes
No
$n_o = n_g + 2$
Structure recruits robot
No
$n_o = n_g + 1$
Yes
Opposite ground detected
No
$1 \le n_g \le 5$
Yes
No
2D structure
Yes
Structure moves forward and recruits robot
No
$n_o = n_g = n_o$
A
Yes
Any free robot
No
Structure moves forward
Yes
Structure moves forward and recruits robot
$n_o \ge n_g + 2$ or $n_o \ge n_p + n_g + 2$
No
Yes
No
$n_o + 1 = n_g + 1 = n_o$
Front robot disassembles, Levy walks, avoids gap
Yes
Front robot disassembles, Levy walks, avoids gap
All robots have disassembled
No
Yes
Target exploration

Figure 5.6: Simple line structure for complex structure. Point A connects to the flow diagram shown in Figure 5.7.

c) A wider rectangular in the structure in Figure 5.2 (c), for each layer 1, 2 and

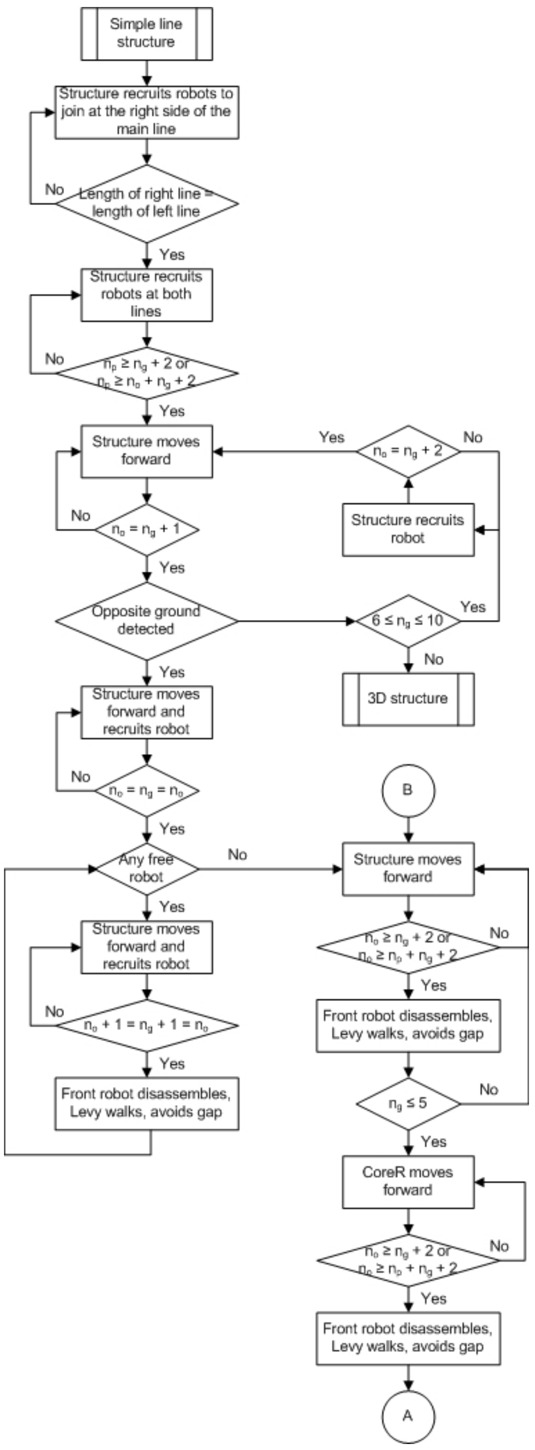

Simple line structure
Structure recruits robots to join at the right side of the main line
Length of right line = length of left line
No
Yes
Structure recruits robots at both lines
$n_p \geq n_g + 2$ or $n_p \geq n_o + n_g + 2$
No
Yes
Structure moves forward
$n_o = n_g + 1$
No
Yes
$n_o = n_g + 2$
Yes
No
Structure recruits robot
Opposite ground detected
$6 \leq n_g \leq 10$
Yes
No
3D structure
Yes
Structure moves forward and recruits robot
$n_o = n_g = n_o$
No
Yes
Any free robot
No
Yes
Structure moves forward and recruits robot
$n_o + 1 = n_g + 1 = n_o$
No
Yes
Front robot disassembles, Levy walks, avoids gap
B
Structure moves forward
$n_o \geq n_g + 2$ or $n_o \geq n_p + n_g + 2$
No
Yes
Front robot disassembles, Levy walks, avoids gap
$n_g \leq 5$
No
Yes
CoreR moves forward
$n_o \geq n_g + 2$ or $n_o \geq n_p + n_g + 2$
No
Yes
Front robot disassembles, Levy walks, avoids gap
A

Figure 5.7: 2D structure for complex structure. Point A is shown in Figure 5.6 and point B is shown in Figure 5.8.

3, is named a. When the front row has not detected the opposite ground after ten rows are over the void, the vertical construction begins. The core will extend itself until fifteen rows are on the initial ground, so, in total, the structure is twenty five robot length. Then, at each side of the cores, robots start to join the structure until the length reaches fifteen rows. These side lines will move forward

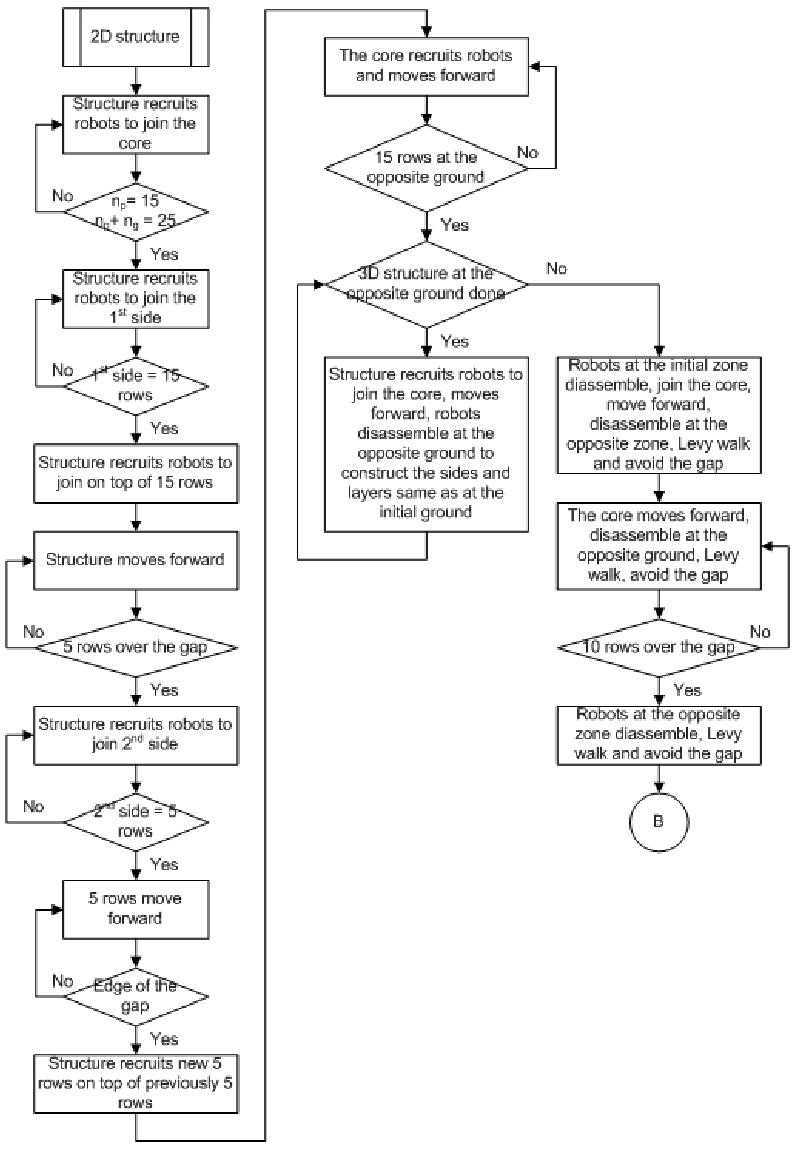

2D structure
Structure recruits robots to join the core
No
n_p = 15
n_p+ n_g = 25
Yes
Structure recruits robots to join the 1st side
No
1st side = 15 rows
Yes
Structure recruits robots to join on top of 15 rows
Structure moves forward
No
5 rows over the gap
Yes
Structure recruits robots to join 2nd side
No
2nd side = 5 rows
Yes
5 rows move forward
No
Edge of the gap
Yes
Structure recruits new 5 rows on top of previously 5 rows
The core recruits robots and moves forward
15 rows at the opposite ground
No
Yes
3D structure at the opposite ground done
No
Yes
Structure recruits robots to join the core, moves forward, robots disassemble at the opposite ground to construct the sides and layers same as at the initial ground
Robots at the initial zone diassemble, join the core, move forward, disassemble at the opposite zone, Levy walk and avoid the gap
The core moves forward, disassemble at the opposite ground, Levy walk, avoid the gap
10 rows over the gap
No
Yes
Robots at the opposite zone diassemble, Levy walk and avoid the gap
B

Figure 5.8: 3D structure for complex structure. Point B is shown in Figure 5.7.

until five rows are over the void, so it is similar to the relation of $n_p > n_s$, but, here, the relation becomes $n_{ap} > n_{as}$. Then a group of robots will join at the top of those fifteen rows, including the top of their cores. Then, five more rows of robots will join the structure at the outer sides. These rows will move forward until the first row reaches the edge of the void. Following these rows, a group of robots will cover their top and settle the connection to the structure. And, to further secure the structural stability, these two layers of five rows will be covered by the third layer of robots at the top. Then, the cores will recruit robots to join, move forward reaching the opposite landing, disassemble and construct the same structure as at the initial zone so that the whole structure is forming an arch-like bridge. After this structure settles, the cores continue recruiting at the initial zone, moving forward and releasing at the target zone, to deliver robots between two separated zones. After there is no robot to recruit, the structure in the initial zone will break away and join the core to be delivered to the target zone. When there is no robot to join the cores, the structure acts moving forward and releasing robots at the target zone, the opposite to what was done at the initial zone when it started to build the vertical structure. For the vertical structure, if it is divided equally into half structure and taking only one row, the relation between the core to its outer side and layer will become A, 4A and 9A, where A equals to a robot. The structural stability for the whole arch-like bridge is maintained as the relation of robot numbers at the (initial zone || over the void || and target zone) is $(2(a)+2(4a)+2(9a) = 28a \,||\, 2(4a)+2(a)+2(4a) = 18a \,||\, 2(9a)+2(4a)+2(a) = 28a)$. Where the number of robots at each initial and target zones is larger than the number of robots over the void, the structural stability has been settled. When the structure has not reached the opposite landing, the structure will maintain its stability at the (initial zone || over the void) as $(2(a) + 2(4a) + 2(9a) = 28a \,||\, 2(4a) + 2(a) + 2(a) = 12a)$.

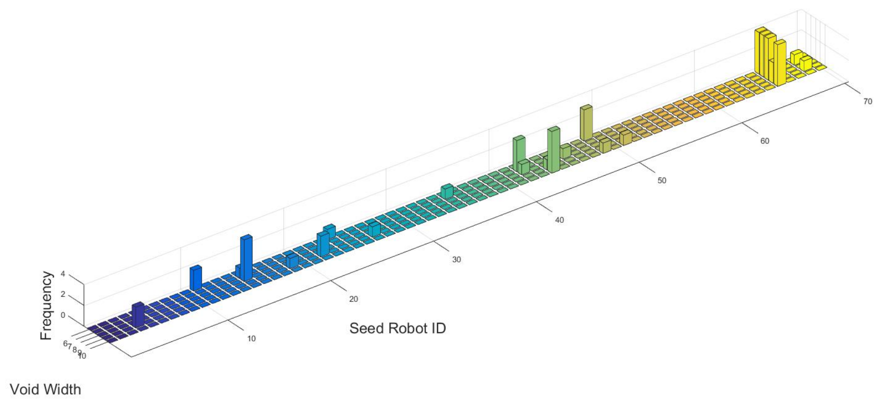
d) Figure 5.2 (d), the structure for sixteen to twenty robot wide void is the extended version of the one for eleven to fifteen robot wide void. If, previously, in the final structure the middle of the arch-like bridge consists of only five rows of two robots, in this structure the middle part consists of ten rows of two robots. The structural stability of the final structure is maintained by the relation of (initial zone || over the void || and target zone) with $(2(a) + 2(4a) + 2(9a) = 28a \,||\, 2(4a)+2(a)+2(a)+2(4a) = 20a \,||\, 2(9a)+2(4a)+2(a) = 28a)$. When the structure is still trying to reach the opposite landing, the structural stability is maintained by the relation of (initial zone || over the void) with $(2(a) + 2(4a) + 2(9a) = 28a \,||\, 2(4a) + 2(4a) + 2(a) + 2(a) = 20a)$.

e) For twenty one to twenty five robot wide void, the structure is extended both in the horizontal and vertical, Figure 5.2 (e). When the front row has not detected the landing area after fifteen rows are over the void, the structure recruits an **a**. Then an **a** also joins the structure at each side as well as the top layer to cover it. When the opposite ground has not been reached, the structure maintains its stability by keeping the relation (initial zone || over the void) with $(2(a) + 2(4a) + 2(9a) + 2(16a) = 60a \,||\, 2(9a) + 2(4a) + 2(4a) + 2(a) + 2(a) = 38a)$. Then, the complete structure will be maintained by the relation of (initial zone || over the void || and target zone) and $(2(a) + 2(4a) + 2(9a) + 2(16a) = 60a \,||\, 2(9a)+2(4a)+2(a)+2(4a)+2(9a) = 54a \,||\, 2(16a)+2(9a)+2(4a)+29a) = 60a)$.

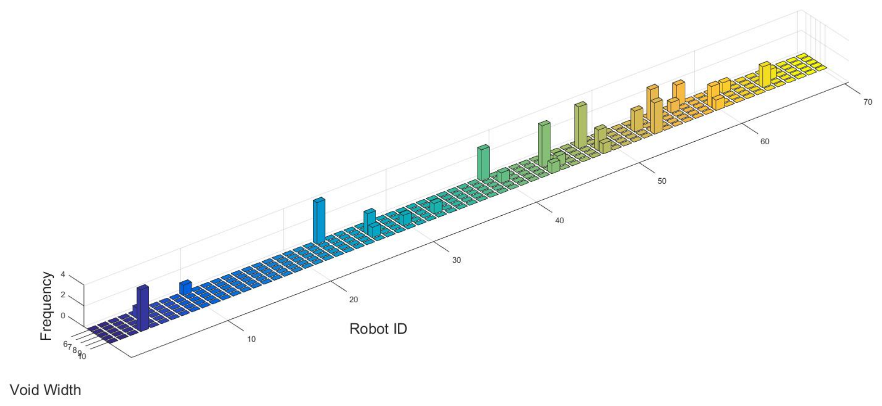
f) This structure is the extended version of twenty one to twenty five robot wide void that works for twenty six to thirty robot wide void. Similar to the structure of sixteen to twenty robot wide void, the middle of the arch-like bridge consists of ten rows of two robots, Figure 5.2 (f). The stability when the structure has not reached the opposite landing is maintained by the relation $(2(a)+2(4a)+2(9a)+2(16a) = 60a \,||\, 2(9a) + 2(9a) + 2(4a) + 2(4a) + 2(a) + 2(a) = 56a)$ of (initial zone || over the void). And the complete structure is maintained by the relation $(2(a)+2(4a)+2(9a)+2(16a) = 60a \,||\, 2(9a)+2(4a)+2(a)+2(a)+2(4a)+2(9a) = 56a \,||\, 2(16a)+2(9a)+2(4a)+2(a) = 60a)$ of (initial zone || over the void || and target zone).

This strategy only works until the span of the structure over the void is thirty robot wide void. Beyond this number, the structural stability when it has not reached the target landing will not be maintained, as the number of robots is more than the number of them on the initial ground. The number of robots at the initial zone can be added, but it requires more complex strategy. However, this strategy is only simulated until ten robot wide void.

5.6 Results



(a)



(b)

Figure 5.9: The frequency of robots that become (a) the seed or (b) detect the target for complex structure.

Simulation for the proposed complex structure has been conducted for six to ten robot wide void. Ten simulations have been run for each void width where the size of all of the swarms is seventy robots. Their initial positions are random before exploring the initial zone. Snapshots from the simulation showing a swarm crossing a ten robot wide void can be seen in Figure 5.14 as an illustration of how the swarm crosses a larger void by building a two-line structure to strengthen the connection and reduce the deflection.

Figure 5.9(a) shows that all robots can become the seed that detects the void for the first time and initiates the construction of the structure. They also have the same opportunity to find the target and inform other robots to stop the exploration, as shown in Figure 5.9(b). And, as can be seen, becoming the seed does not make a robot detect the target, although, in several cases, both the seed and the target detector can be the same robot. Figure 5.10 shows the position of where the seed started the construction.

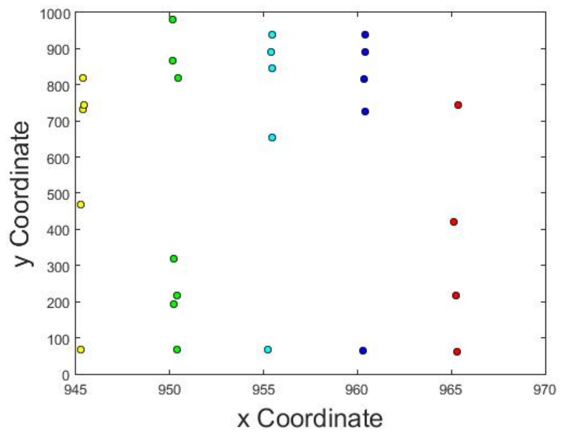


Figure 5.10: The seed position at the edge of the void for complex structure.

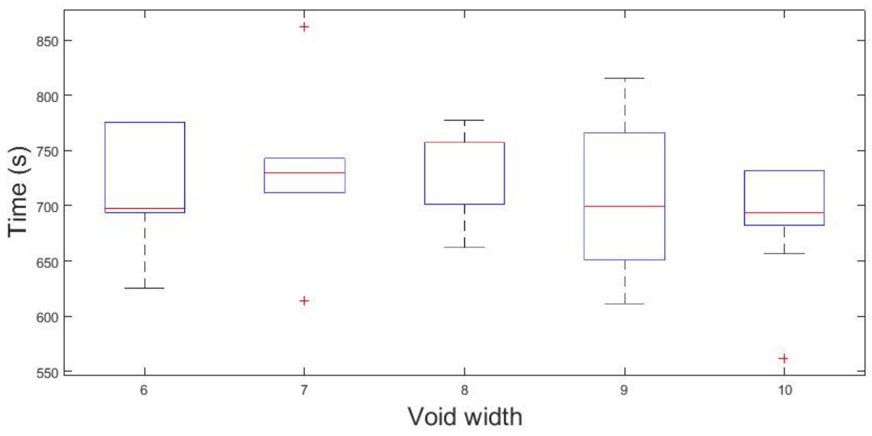


Figure 5.11: The simulation time for complex structure. The swarm size in all cases was seventy robots.

The simulation time, shown in Figure 5.11, consists of three main ranges of time: the initial exploration time, the construction time and the target exploration time, as can be seen in Figure 5.12. When the void is wider, that brings the consequence of a narrow initial zone, and the initial exploration time is decreased, as can be seen in Figure 5.12(a). As for target exploration time, as can be seen in Figure 5.12(c), the size of the ground did not affect the exploration time, as robots wandered to find the target by using Levy walk.

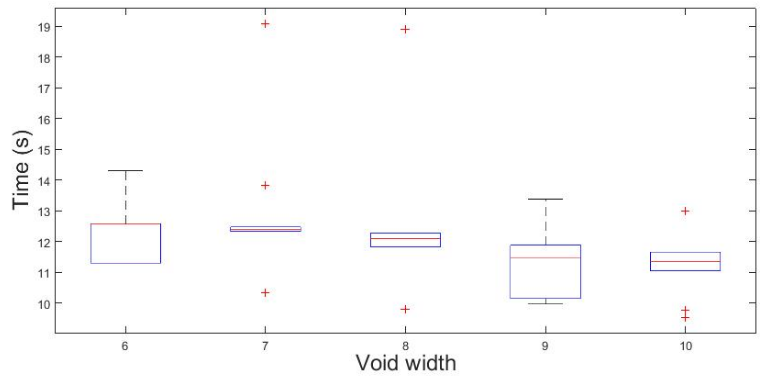
To be more precise, two additional ranges of time were recorded in construction mode. The first one is assembly time, the time required for robots starting from the seed setting itself orthogonal to the edge of the void, until the whole two-line structure has been built and it is ready to deliver other robots from initial ground to the opposite one. Another is disassembly time, the time required for the structure to start disassembling when other robots have been delivered. As can be seen in Figure 5.13, both assembly and disassembly time increases as the void becomes wider.

5.7 Discussion

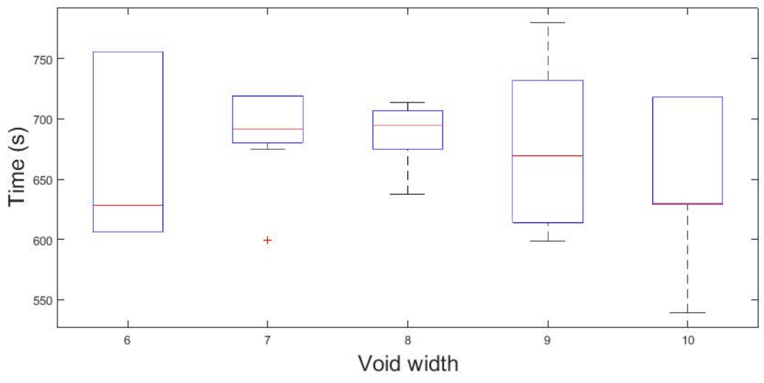
Results show that the bridge launching mechanism can work for a two-line structure as well as a simple line structure. The whole structure can be built allowing robots to be delivered from the initial zone to the opposite zone where the target is located. As dynamic variables are also applied in this algorithm, all robots have the same opportunity to locate the void and the target. And it has been shown that the robot that finds the target is not the seed that initiated the construction. The distributed placement proves that robots have no prior knowledge of void location. So, it will be detected once a robot reaching it first as the seed position at the edge of the void is also distributed.

The simulation time is influenced by the exploration time at both initial zone and target zone, and also the construction time. With all swarms having the same size, there is no distinguished trend of the total simulation time. However, the more narrow exploration area affects the initial exploration time, as robots are faster in locating the void, but this does not work for target exploration time. There is a significant difference between locating the void and the target. The void has an edge, 1000cm long, while the target is only a 5cm circular spot. So, detecting the target depends on the random coverage of the swarm at the target zone.

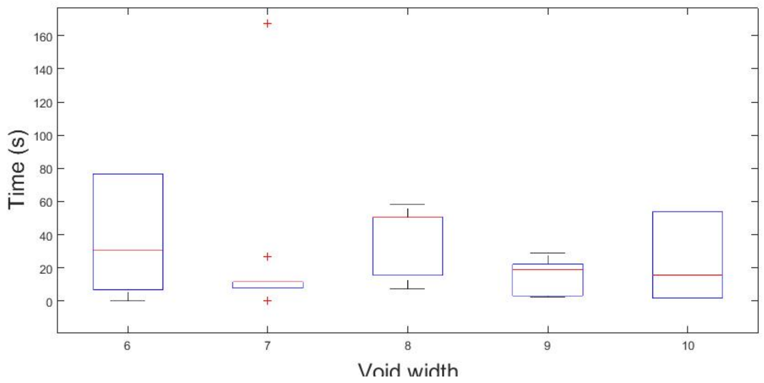
If the construction time is considered, it decreases slightly as the void is increased from seven to ten robot wide void, but increases significantly from six to seven robot wide void. The more narrow exploration area helped the swarm to construct faster. However, the increasing time can be affected by the time taken for robots to build the structure from the beginning until the whole construction and the time from when the structure started to disassemble until finished. The longer structure requires a longer time to construct and to disperse.



(a)

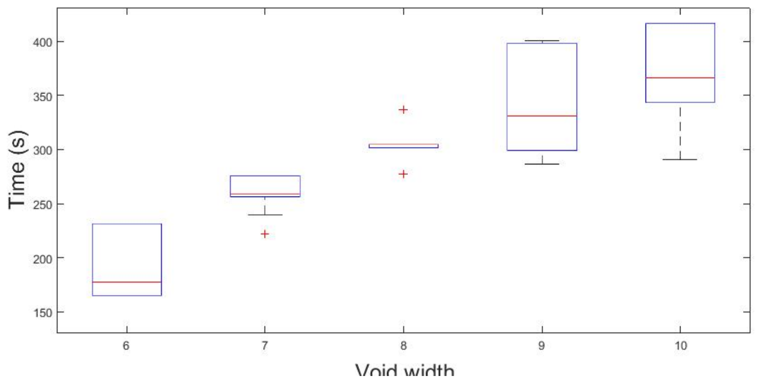


(b)

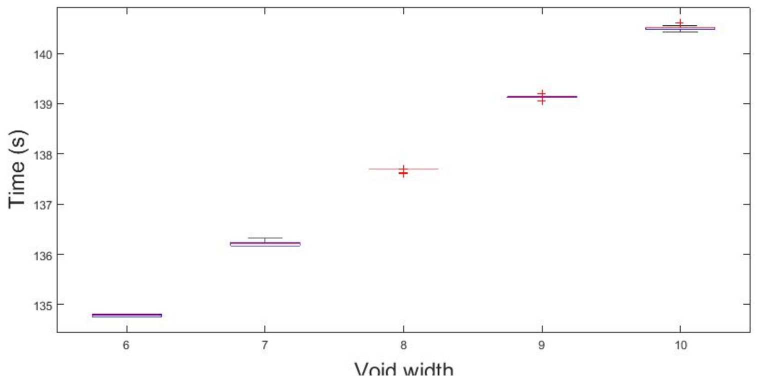


(c)

Figure 5.12: The detail of simulation time for complex structure. The swarm size was seventy robots for all simulations. (a) The exploration time at the initial zone. (b) The construction time to build the structure and bridging two zones. (c) The exploration time at the target zone.



(a)



(b)

Figure 5.13: The detail of construction time for complex structure. The swarm size was seventy robots for all simulations. (a) The time required to assemble robots into the whole construction. (b) The time required for robots to disassemble when there are no free robots at the initial zone.

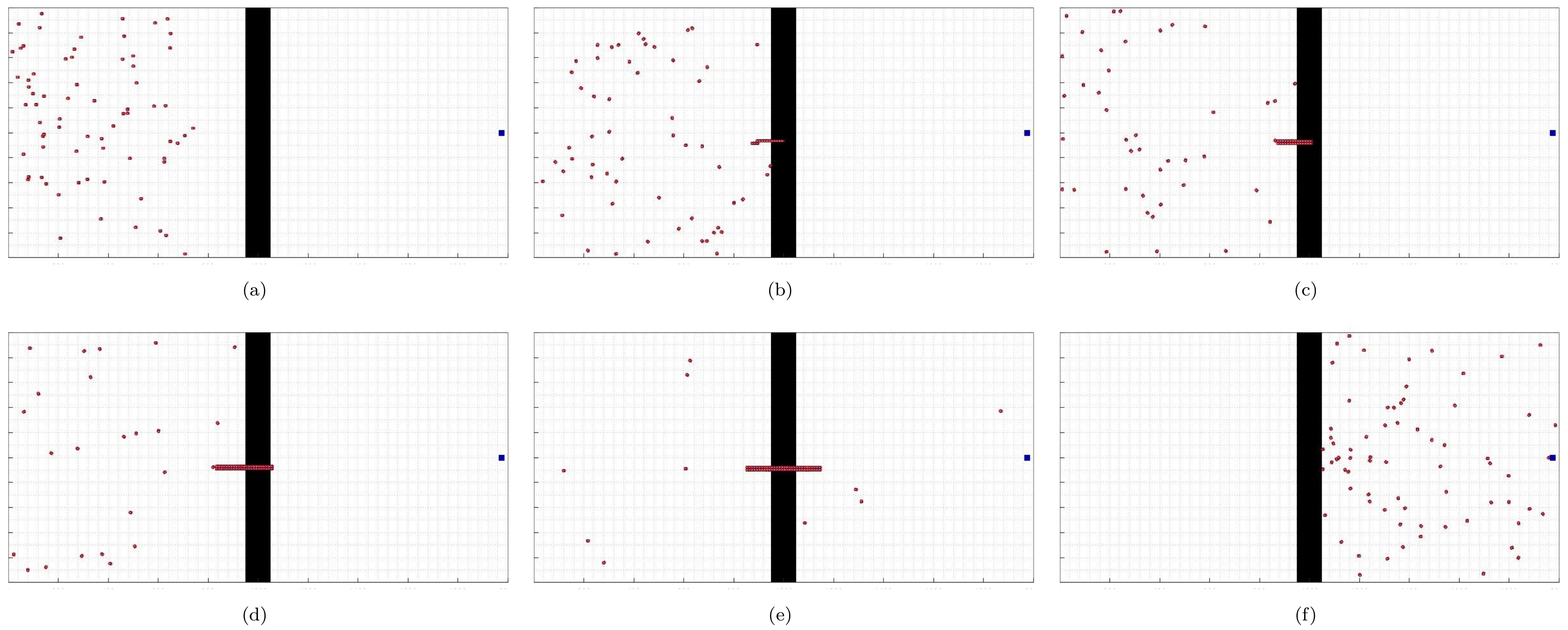


Figure 5.14: Robots attempting to cross a ten robot wide void with a two-line structure. (a) Seventy robots were ready to wander. (b) The front robot in the single line structure has not detected the opposite landing, so the structure started to recruit robots to join as the second line at the right side. (c) The structure moved forward reaching the opposite ground by recruiting robots at both lines to help itself maintain the stability. (d) The structure has managed to reach the opposite landing and still recruited robots to maintain the stability. (e) The structure has managed its stability and it was delivering robots to the other side by continuing to recruit robots and moving forward. (f) A robot has found the target.

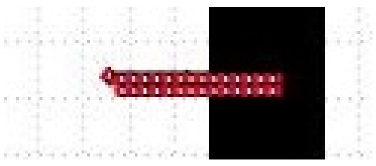


Figure 5.15: Two-line structure in detail. The second line joins at the right side of the first line after a line structure hasn't detected the target zone.

Chapter 6

Outline Specification for a Void Crossing Robot

In this chapter, a number of proposed robot designs suitable for crossing a void are discussed. The chapter firstly discusses the requirements based on challenges and considered aspects of constructing a simple line structure to cross a small void in the real-world applications. A discussion of the use of a simplified 3D printed robot model to demonstrate the coupling mechanism is provided. The chapter concludes with a discussion of robots designs required to cross the larger void.

6.1 Introduction

In considering the development of a swarm robot suitable for use in the construction of a structure a number of issues need to be considered.

1. How the swarm is to detect the void and then move to the seed robot to construct the structure.
2. In this work we are considering small mobile robots that can form a connection autonomously. This raises a number of points including: (i) the alignment of the robots, (ii) the mass of robots, (iii) the intra swarm communication and (iv) the energy consumed by the connection mechanism.

Subsequently, it is considered to use all connection mechanisms of the proposed robot platform to extend self-assembly in a 3D manner as well as the connection. However, for a simple line structure the exploited connection mechanism is only the front-rear connection, which is using a pin and hole. A number of simple tests using 3D printed simplified platforms are discussed to prove that the proposed algorithm can be used in the real world.

In considering coupling two main options are available, magnetic or mechanical. While the magnetic appears to have a number of advantages, particularly of simplicity, it was rejected for two main reasons.

1. If electromagnetics are used, there is a significant power requirement, particularly as the robots have very limited energy storage capability.
2. If permanent magnets are used, the making of a robot's connection is relatively easy, however seperation is very difficult, this could be addressed by either the application of an external force such as in the M-Blocks (Romanishin et al., 2013) which is highly disruptive to the robots' position or the application of a continued magnetic field from a secondary electromagnet, which is both energy inefficient and impact on the performance of the system.

In the coupling considered in this work, the use of a powered mechanical coupling is considered optimal, power being consumed only when the mechanism is either click or unclick.

Compared to robots reviewed in Section 2.3, the important requirement for swarm robots performing void crossing is the rigid connection that support the longer span of the structure constructed. Instead using relatively weak mechanisms in gripping other robots, the proposed swarm robot uses modified grooved pins and engagement latches. Those connections are placed distributively on front-rear side of robots for distributive weight purpose to reduce the deflection. Later, additional mechanisms for left-right and top-down connections are considered using the simple sliding and twist-and-lock mechanisms that strengthen the main front-rear connection. The robot shape is also considered to support the rigid and long span structure, as well as its light weight and relatively small dimension. Its communication system is also considered to not using visual mechanism as processing the image is quite heavy for the relatively simple robots, the use of light based (IR) and radio communication systems is considered.

6.2 Requirements

In performing self-assembly for small void crossing, swarm robots must meet a number of challenges that are related to real-world applications. From the initial explorations to the exploration of the arena in both zones, and the construction in-between two explorations to build the simple line structure to bridge two zones, each task has its challenge.

When all swarm robots are exploring the initial zone, they have to locate the void. Here, the first challenge is how to differentiate the void from the average floor. An assumption of a smooth surface can be taken. Taking the case of rough terrain, swarm robots must be able to detect the void from the ground by its depth. By putting a ground proximity sensor at the front of each robot, a certain distance can be set so that a robot will recognise the void when the sensor measurement exceeds a set distance.

Once a robot detects the void, it has to inform the swarm that they can begin the construction. This robot becomes the seed and its mode is switched from exploration to construction mode. For this purpose, a broadcasting communication is required so that other robots stop searching for the void; instead, they have to avoid the void. Exploration can still conducted by free robots, but it is also to help give space for robots that have already joined the construction process. The swarm size is an important factor for the broadcast communication system. With scalable swarm size, it is better to use fewer communication nodes. However, when there is a technical limitation, a rebroadcasting system can be used.

After broadcasting that the void has been found, the seed robot has to set itself approximately perpendicular to the void and then remains stationary. An assumption that the edge of the void is nearly even can be taken. But there is another possibility by putting a pair of proximity sensors at the bottom-front of the robot, so that it will set itself facing the void and the opposite ground. Both right and left wheels will not slip to the void as well as the front caster, and the seed will recruit a free robot to join the construction.

Recruiting a free robot to join the structure is the next challenge. The seed can broadcast the message of recruiting. The fastest reply from a free robot is received by the seed indicating it is the closest free robot to join the structure. Then, a message to the address of the free robot is sent by the seed through one-to-one communication, pointing out that the robot has to change its initial exploration mode into the construction mode, and join the structure. Keeping the communication, the recruited robot has to move closer to the seed.

The challenge for the free robot is to locate the seed. The communication between the seed and the recruited robot is kept to guide the recruited robot moves towards the seed. There are two pieces of information required, the distance between them and the seed direction. Without GPS and static beacons, the only opportunity to gain the information depends on the swarm robot communication system.

Then, the most challenging aspect is setting the orientation of the recruited robot to the seed and the coupling between them to connect with each other. Once the recruited robot reaches a certain distance to the seed in a certain direction, it has to set itself so its front side will be facing the seed's rear side. The seed robot has to transmit location signal from its connectors that can be received by the recruited directly on its connectors, to guide them to connect accurately. Once the direction is set, the recruited robot has to move towards the seed for the connection purpose. When the distance between them is minimum, the locking system is initiated. For this purpose, sets of signal transmitters have to be embedded on the rear connectors of each robot, and sets of receivers on the front connectors, as any robot can be the seed or recruited robots.

Once the recruited robot joins the structure, the recruiting robot is the last robot in the structure. Other robots at the front have already changed their mode into construction mode, so they will ignore recruit message from the last robot. Once the structure maintains its stability and it is moving forward, the seed has to detect the opposite ground using its ground sensor, differentiating the void and the ground. It keeps informing other robots in the structure to recruit free robots to maintain the stability and reach the opposite ground. When the opposite ground is detected, it still informs the structure for the recruitment until the whole bridge structure has formed. When the number of robots is more than the required number to construct the whole structure, the seed disassembles from the structure, guided by signal transmitters and receivers on connectors, and changes the mode into target exploration mode. Then, the previously second robot acts as the seed, and this process continues until all robots disassemble at the target zone.

When all robots have disassembled and changed their mode into the target exploration mode, the real target searching is started. As, currently, the target is a static beacon, in which the system to be found is just determined by a specific radius, all robots can recognise the target.

In the case when the swarm size does not fulfil the structural stability in the construction process, the seed informs all robots in the structure to retreat and disassemble, until the structure has disassembled. As well as giving that information, the seed keeps detecting the void and the ground until the initial ground is detected and the last three front robots are ready to disassemble.

The coupling mechanism also plays an important role as it determines the strong and stable construction to prevent the structure breaking or falling into the void. The strong and stable structure also helps it reach the wider void. The shape of the swarm robot also affects the strong structure.

To conclude, the requirements that each swarm robot to be capable of self-assembly, are:

1. A swarm robot has to be autonomous, mobile, and simple. For the locomotion, differential-drive steering can be applied. Although this system brings an unstable locomotion, two casters can help in maintaining its stability. All robots have their own controller and they move freely in the environment, helped by their

own perception system. An assumption is taken wherein they have unlimited self-propelled components.

2. Swarm robots have an ability to manoeuvre in the environment, so collision sensors have to be equipped into the robot.
3. All robots in the swarm are homogeneous. They have the same features and they can be mass produced at a low cost to an identical standard.
4. Swarm robots for the application should be of a relatively small size and low weight.
5. A specific shape for swarm robots has to be carefully considered to support the chain-type self-assembly to construct the rigid structure with a certain span, and to decrease any deflection of the structure.
6. A specific proximity sensor that can differentiate between the ground and the void, mainly based on the depth of the detected ground, although swarm robots work on uneven terrain.
7. A broadcasting communication system so that the seed can recruit a free robot to join the structure, and inform other robots in the structure to keep moving forward or retreat. This system is also used by other robots to recruit and when they act as the seed robot after the previous seed robot has disassembled from the structure. This system can also be used for the target, broadcasting its position.
8. A specific communication system that can guide the recruited robot and locates the recruiter until a certain distance is required.
9. A specific communication system between the recruited robot and the recruiter to orientate the recruited and connect with the recruiter to assist in the coupling process.
10. Low impact connectors for the coupling mechanism, supported by the ability to couple through alignment for the coupling and orientation.

Based on the requirements of swarm robot performing self-assembly for void crossing, the proposed swarm robot has dissimilar features to robots reviewed in Chapter 2. Its shape has been carefully chosen considering the long span and rigidity when it assembles to other robots in the structure, which is cuboid and octagonal prism. It occupies differential-drive steering as the most simple locomotion and is equipped by two casters with suspension for its stability and to work on uneven terrain. In the beginning the proposed robot is only equipped by broadcasting communication using radio frequency system. In the next development the communication system applies IR that can be used as both one-to-one and one-to-many communication systems. As for the final version, both radio frequency and IR communication systems are used, as well as IR for assisting in coupling process. The most distinguishable feature of swarm robot is its connection mechanisms that have been carefully considered based on the requirement of rigid and long span structure, strong connection, and the relatively simple mechanism for supportive connection.

The main connection mechanism at front-rear side is applying grooved pins and engagement latches. A pair of grooved pins is placed at the front surface and a pair of engagement latches is placed at the rear surface for the distributive weight purpose, so that the deflection of assembled robots can be reduced. Instead of using a plate with many pins or latches, each connector only occupies three pins or latches, as it is already strong but minimise the space for the relatively small robot. In the development, the sliding mechanism for left-right connection makes a robot easy to connect to the structure but it supports the strong connection for wider structure. And the twist-and-lock mechanism for top-down connection will hold the robot below and strengthen the main connection.

6.3 Possible Design Solutions

Several designs are proposed to answer the challenge of self-assembly in swarm robotics for crossing a void: Cube Robot, OctBot and CuBot. CuBot is the revised version of Cube Robot and has been realised using 3D printing. The OctBot base, mechanical and electronic systems, and its connectors have been proven with successful.

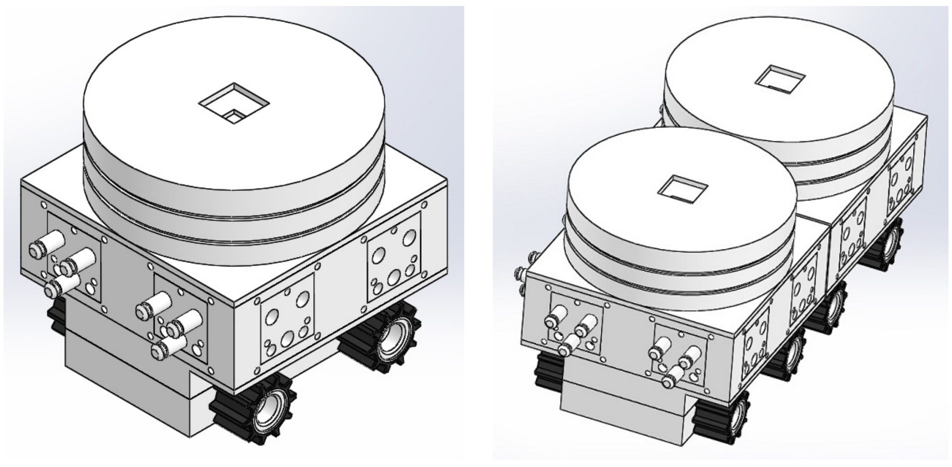
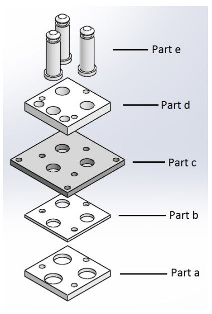

(a) (b)

Figure 6.1: Cube Robot design. (a) The initial proposal for a swarm robot capable of forming structure. It consists of three main parts: the locomotion at the bottom, the connection in the middle, and the sensor at the top. (b) Two robots connect to each other, configuring as a line formation. The male connectors of the rear robot connect to the female connectors of the front robot.

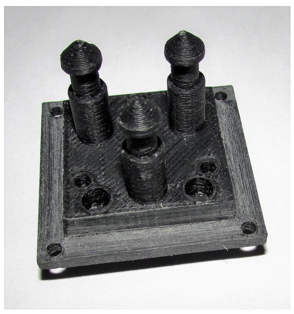
6.3.1 Cube Robot

Cube Robot, a robot that has a cube-like shape, is the first model proposed to answer the challenge of bridging two separated grounds using assembled swarm robots. Its shape and connection mechanism were considered based on span and deflection aspects.

6.3.1.1 Connection Mechanism



(a)



(b)

Figure 6.2: The proposed connection pins for Cube Robot, which consists of five parts. Part a is the locking plate. Part b is the plate to lock the grooved pins. Part c is a PCB to place IR proximity sensors for pairing guidance. Part d is the locking plate. Part e consists of three grooved pins.

The connection mechanism of a Cube Robot consists of grooved pins (Figure 6.2) and engagement latches (Figure 6.3). Each pair of grooved pin sets is placed on the front and the right sides of the robot. And each pair of engagement latch sets is placed on the rear and the left sides of the robot. This arrangement enables the connection mechanism for not only in one-dimension (front-rear side), but also two-dimension connection (front-rear and right-left sides). In the earlier plan, a robot can be held by two robots at the rear of it. Half of it is connected to half of one robot, and the other half is connected to another. This is the main purpose of putting a pair of connection sets on each side of a robot.

The other purpose of putting a pair of connection sets on each side is to decrease the deflection of the structure of assembled robots compared to only one set. The weight is not centralised at the centre of the side, so that the side base will not be bending. The triangular arrangement of grooved pins and engagement latches aims to maintain the strong coupling compared to two pins and two blades of engagement latches, but they do not require more space and complexity than four or more pins and blades.

Part c of grooved pins and part b of engagement latches are electronic layers that contain two IR proximity sensors for assisting the coupling process. The transmitter parts are put on the engagement latch sets, transmitting the signal to guide the recruited robot to align. The receiver parts are put on the grooved pins so that the recruited robot

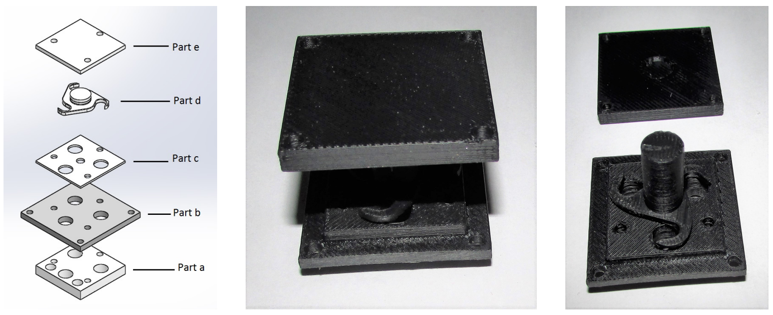


Figure 6.3: The proposed latches for Cube Robot, which consists of five parts. Part a is the locking plate. Part b is a PCB to place IR proximity sensors for pairing guidance. Part c is the separation plate for PCB and latches. Part d is a triangular blade to lock pions when they are pairing. Part e is the locking plate.

has to find a transmitted signal from the recruiter to align. Once it finds those signals, the recruited have to move towards the recruiter until their distance is minimum. This process enables the grooved pins to enter the hole of the engagement latch set. When the distance is minimum, the engagement latches rotate, locking the grooved pins.

However, an additional mechanism has to be equipped for the arrangement of pairs of grooved pins at the front and right sides, and pairs of engagement latches at the rear and left sides. When three robots have to be connected, a robot is held by two robots at the rear; the second rear robot will meet a difficulty to connect, as two sides of it have to assemble with the corner of the constructed structure by the previous two assembled robots, as can be seen in Figure 6.4(a). The front-left connector of the rear robot is not aligned with the rear-right connector of the front robot. And its left connectors are not aligned with the first rear robot's right connectors. An additional controlled swing mechanism can be equipped for each connector to help all required connectors to align, as can be seen in Figure 6.4(b). However, this additional mechanism requires additional energy consumption, as each connector has to be equipped with a motor to drive the controlled swing, and controlling the angle of swinging brings greater complexity for the whole system. Another possibility is the controlled push-pull mechanism applied to each set. But this mechanism brings not only energy consumption problem, but also space and complexity problems. If energy consumption, robot dimension, complexity and low-cost platforms are not considered, both additional mechanisms can be applied.

6.3.1.2 Electronic System

The required systems of the proposed Cube Robot are as follows.

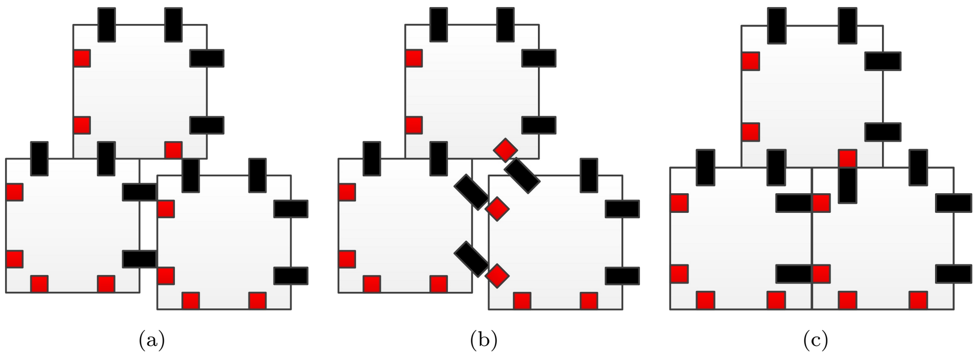


Figure 6.4: The emerged problem of a Cube Robot's connection mechanism. (a) The second rear robot meets the difficulty to connect to two other robots as they form a corner. (b) Additional controlled swing mechanism for each set connector to enable the connection process. (c) A push-pull mechanism for each connector set to provide the space for alignment.

1. *The perception and locomotion units.* There are three ground sensors that are placed at the base of each robot to detect the depth of the void. To detect objects to be avoided, sixteen IR proximity sensors are mounted at the top of the robot. An accelerometer is added to measure the acceleration of the robot.
2. *The communication unit.* The main communication unit is an RFID module. Additionally, eight light sensors and 8 × 3 colour LEDs are placed in a circular position above the IR proximity sensors that can be used to detect emitted light colours of other robots for the purpose of communication or setting the assembly rule.

6.3.1.3 Overall Structure

After evaluating a number of polygons, cube/cuboid shape is selected. The specification of evaluation to design a Cube Robot consists of: the strength of each connection, minimum gap between connected robots to achieve a strong connection and the efficiency of space so that individuals also rely on their shape to support each other, and the length of the span of assembled robots to reach the previously inaccessible zone.

Considering the minimum gap between assembled robots using homogeneous polygons for the strong connection, there are only three shapes that meet the requirement: triangle, square and hexagon. The simplicity of sensor placement is further considered and there are only four polygons that meet the requirement: square, hexagon, octagon and cylinder. From both requirements, only square and hexagon can be used as the main shape of the swarm robot. Subsequently, both polygons were projected into 3D, forming cube/cuboid and right regular-hexagonal prisms. By comparing both polygons, it is clear that a structure constructed of a number of cubes/cuboids has a longer span than the one that is built using a right regular-hexagonal prism. After considering all aspects, it has been decided that cube/cuboid is used as the main shape of the swarm robot.

The dimension of a Cube Robot is expected not to exceed $120mm \times 120mm \times 120mm$. The Ackerman steering is used, as this mechanism makes the robot stable to move, although its control is not as simple as differential-drive steering. The bottom base of the robot is a block on which to place four wheels. The middle base is a cuboid one on which to place all blocks of grooved pins and engagement latches. The top base is a cylinder on which to place proximity light sensors, colour LEDs and the main controller. The model can be seen in Figure 6.1(a) and two connected robots can be seen in Figure 6.1(b).

However, after two robots connected, there is still a 1.7mm gap between them, because the designed grooved pins exceed the space of the engagement latches. This void is enough to drive the deflection that works on only the grooved pins. And, because the weight that works on grooved pins is large, it can cause a fracture of the structure. In addition to the energy consumption and the arrangement of grooved pins and engagement latches blocks, the use of Ackerman steering is complex; thus it is better to change the steering into differential-drive steering.

6.3.2 OctBot

6.3.2.1 Connection Mechanism

A simpler connection mechanism has been applied on an OctBot using a ring and capture mechanism that occupies a worm-screw mechanism to drive the capture mechanism. The mechanism itself is mounted on the front side of the OctBot and, on the rear side, a ring is assembled, as can be seen in Figure 6.5. When a robot recognises another one that is ready to be captured through their aligning proximity sensors, it moves towards the robot in the front until the fingers of the mechanism can be released on the inner side of the ring of the front robot, guided by the proximity sensors. Then, its motor drives the fingers of the capturer to open inside the inner side of the ring; the motor is then deactivated and the fingers hold the ring passively. When those two robots require to disassemble, the motor is in reversed, so that the fingers are closing and the front robot is ready to move away from the rear robot.

However, the gap between two assembled robots cannot be less than 40mm because of the OctBot's geometry and the limitation of the sensors and emitters, and thus might create a significant deflection in a long structure. It is not possible to reduce the length of the capturer as there is a reading limitation of the proximity sensors. Furthermore, the size of the capturer and the ring is not as compact as the expected one. Therefore, although this design is simple and rigid enough, and saves energy consumption, it is better to use another approach for the docking mechanism.

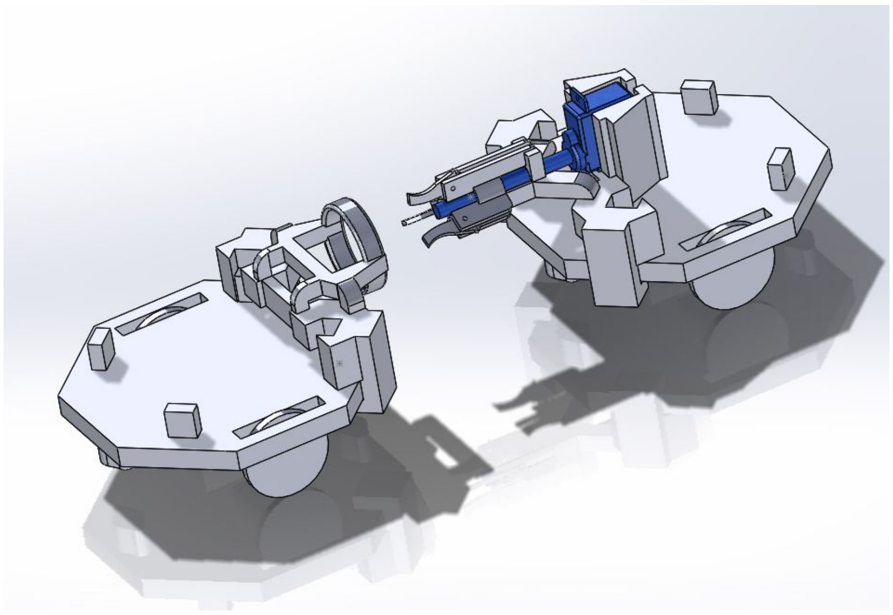


Figure 6.5: OctBot coupling mechanism. It uses a worm-screw mechanism to enable one OctBot to connect to another OctBot. It has been designed with a low cost swarm robot platform in mind; it is a lighweight, compact locking mechanism to allow swarm robots to autonomously form a rigid structure; it has a detection mechanism to ensure two robots are properly docked and communication between robots when they have joined. To capture the three fingers are moved to within the ring and then opened. As they open the robots are pulled together to form a solid structure.

6.3.2.2 Electronic System

The current electronic systems of the OctBot have been implemented into four main parts.

1. *The controller.* The ARM Cortex-M0 has been selected as the main controller for an OctBot. This µC has configurable analogue and digital subsystem on board and allows it to be expanded. I2C is utilised for the internal the communication bus.

2. *The perception and locomotion units.* Eight IR LEDs and four IR photodiodes are positioned in four regions to detect and measure the distance to objects to be avoided. An OctBot is also equipped with a KPS-5130PD7C colour sensor to detect the conditioned arena, consisting of several zones. Another embedded sensor is IMU (FXQOS8700CQ). It is an accelerometer and magnetometer which is used to measure the acceleration and orientation of an OctBot in motion. A standard proximity sensor unit is also equipped to detect very dark patches of floor and an actual edge.

3. *The communication unit.* To communicate to other robots, IR photodiodes are exploited as this type of communication is cheap and simple to be used.

4. *The power unit.* The OctBot is powered by a 3.7V 1200mAh Lithium Polymer (LiPo) battery. It enables an OctBot to operate for approximately eleven hours in standard condition and over two hours at peak current consumption.

6.3.2.3 Proposed Design

Considering the simplicity for the movement and sensing the environment during navigation, the shape of an OctBot is an octagonal prism. Although the connection between a number of OctBots will not be as strong as assembled cuboids, the connection is still considered strong compared to assembled cylinders. And the span of the built structure is longer than the structure formed by assembled hexagonal prisms.

With dimension not exceeding $100mm \times 100mm \times 30mm$, excluding the connectors, the locomotion of an OctBot is by using differential-drive steering, considering its simplicity, and there are two adjustable casters to support the stability and to enable the OctBot navigate on uneven terrain. These casters are also used to recharge when the OctBot remains on a specifically designed charging station. As aforementioned, parallel with the connection mechanism, an edge detector is embedded using an IR proximity sensor to detect the darkness of patches on the floor or the void.

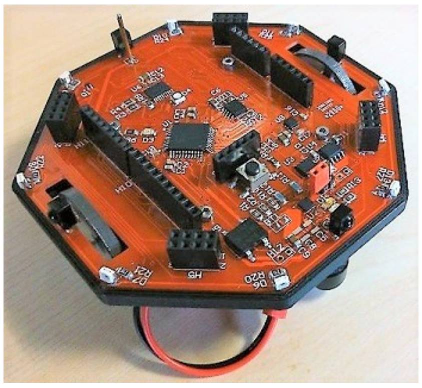


Figure 6.6: The basic OctBot, an octagonal low cost platform to aid swarm robotics research.

The considered minimum energy consumption of its electronic and mechanical systems has been the main advantages of the OctBot, especially the connection mechanism. However, similar to assembled Cube Robots, there is still a gap between two connected robots because of the OctBot's connection mechanism design, which exceeds the octagonal base. With a gap that is wider than two connected Cube Robots, deflection is more likely to occur without the support of the main base of the robot. The connection mechanism of the OctBot only works on an axis, whereby the possible formed structure is only in an axis. However, in the further construction for a large crossing, occupying two or three axes is required to construct a structure considering its span and strength.

6.3.3 CuBot

The cuboid shape supports the strength of the structure constructed by assembled CuBots and its three connection mechanisms support the construction of 3D structure. With a dimension that does not exceed $100mm \times 100mm \times 85mm$, it is more compact than a Cube Robot, but its connection mechanisms rely more on mechanical mechanisms to reduce the energy consumption, similar to an OctBot. When two CuBots connect, there will be no gap between the connection, which decreases the level of deflection of the constructed structure. Using differential-drive steering, the CuBot's locomotion is simpler than a Cube Robot and this type of steering also reduces energy consumption. And, by adapting the flexible casters of the OctBot, a CuBot is able to navigate on uneven terrain as well as maintain its stability. The whole model of a CuBot can be seen in Figure 6.7.

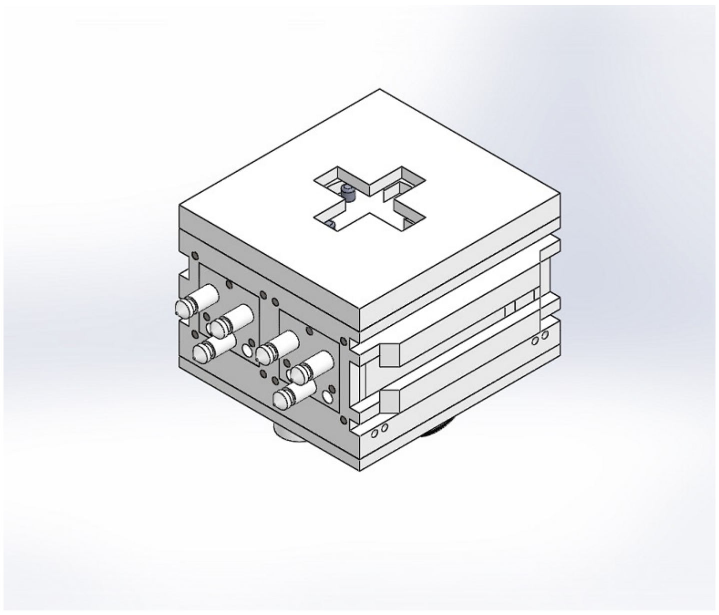


Figure 6.7: CuBot, a cuboid platform model for use within a multidimensional structure.

6.3.3.1 Connection Mechanism

As the proposed dynamic structure for large crossing occupies a 3D structure, there is a requirement that a swarm robot is able to connect to other robots at every side: front and rear, right and left, and top and bottom. However, for the simple line structure, a CuBot only uses a front-rear connection mechanism. A CuBot uses three connection mechanisms, as follows.

1. Grooved pins and engagement latches (Figure 6.8), the same as the Cube Robot's docking mechanism, using active latches. Two blocks of grooved pins are placed at the front of the robot and two blocks of engagement latches are mounted at the rear of the robot. This arrangement has been chosen to maintain the strong connection between robots in a single line structure. However, if octagon is preferred as the shape of the robot for simpler movement, this arrangement can be reduced by only using a block of grooved pins at the front and a block of engagement latches at the rear, thereby reducing the coupling strength. In addition, to reduce the energy consumption, the engagement latches only works when it moves, locking the grooved pins, and, when it moves back, the grooved pins are ready to be released for the disassembling process.

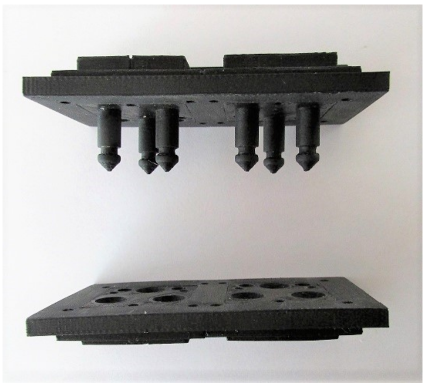


Figure 6.8: CuBot's connection mechanisms for front and rear sides, consisting of grooved pins and engagement latches.

2. Rail system, or left-right connection mechanism, Figure 6.9. Figure 6.9(a) shows the shape of each right and left parts, respectively. Both have two holes at both ends at the bottom for communication mechanism purposes. Figure 6.9(b) shows when both parts have been connected. As can be seen, there is a small gap between the two connected parts as the consequence of the 3D printing results. Figure 6.9(c) shows how the right and the left parts connect. The right part (below) is the part of a robot that has joined in the structure, so that it is static. The left part (above) is the part of a recruited robot that has to join at the right side of the simple line structure. It has to move forward until it is side-by-side with the right part of the recruiter. Its front communication system will communicate with the rear communication system of the recruiter to assist the coupling. After both pairs have met, the recruited robot moves forward until its front and rear pairs of communication systems are in line with the front and rear pairs of the recruited robot. All communication systems of the left-right connection mechanism are based on IR proximity sensors.

3. Twist and lock (Figure 6.10) for the top-down connection mechanism. Figure 6.10(b) is a thin top part that functions as the cover of the robot body and also as the entrance for the upper robot to insert the connector. Figure 6.10(c) is the part under

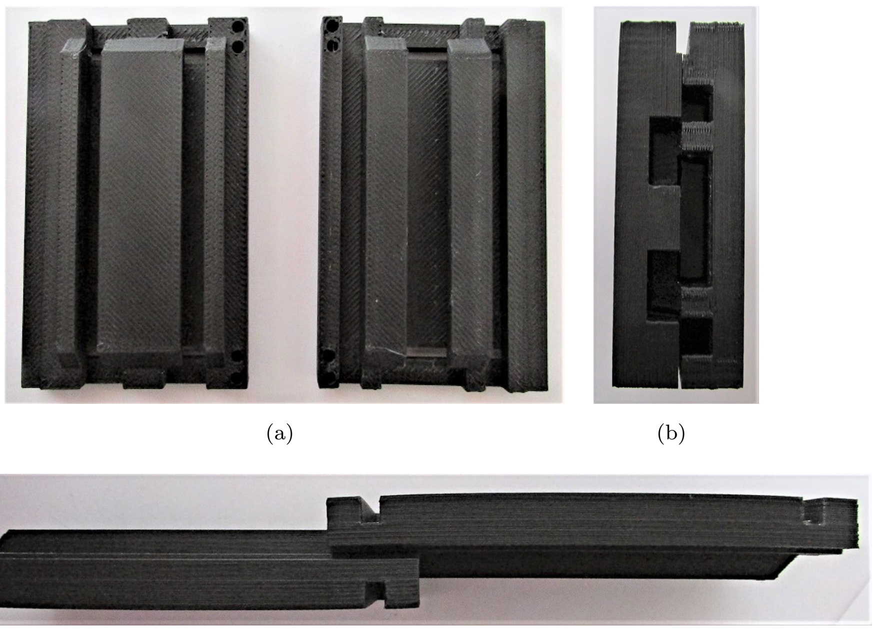

(a) (b) (c)

Figure 6.9: Left-right connection mechanism. (a) The right and the left parts of a robot. (b) The connected right and left parts. (c) The process of sliding, when a robot has to connect at its left side with another robot.

the cover to place the connector of the upper robot. In future development, both the cover and this part have to become a solid part for a strong connection. At the middle, there will be a communication system, an IR receiver. Figure 6.10(d) is the key to hold the connector from the upper robot after it has been rotated to prevent a loose connection. Figure 6.10(a) is the connector of the upper robot where, at the middle, there is an IR transmitter to assist the coupling with the robot below. When it is in pairing, it will be pushed to enter to the cover of below robot. After it has entered, it will rotate clockwise until maximum and will be secured by the key.

With these three mechanisms, the energy consumption of the Cube Robot model has been reduced, as there are only two active blocks instead of four blocks. The motor of engagement latches is also not active when the grooved pins have been engaged. One mechanism only occupies the shape for connection and the other uses a passive mechanical system.

6.3.3.2 Electronic System

The electronic systems of a CuBot have been proposed as follows.

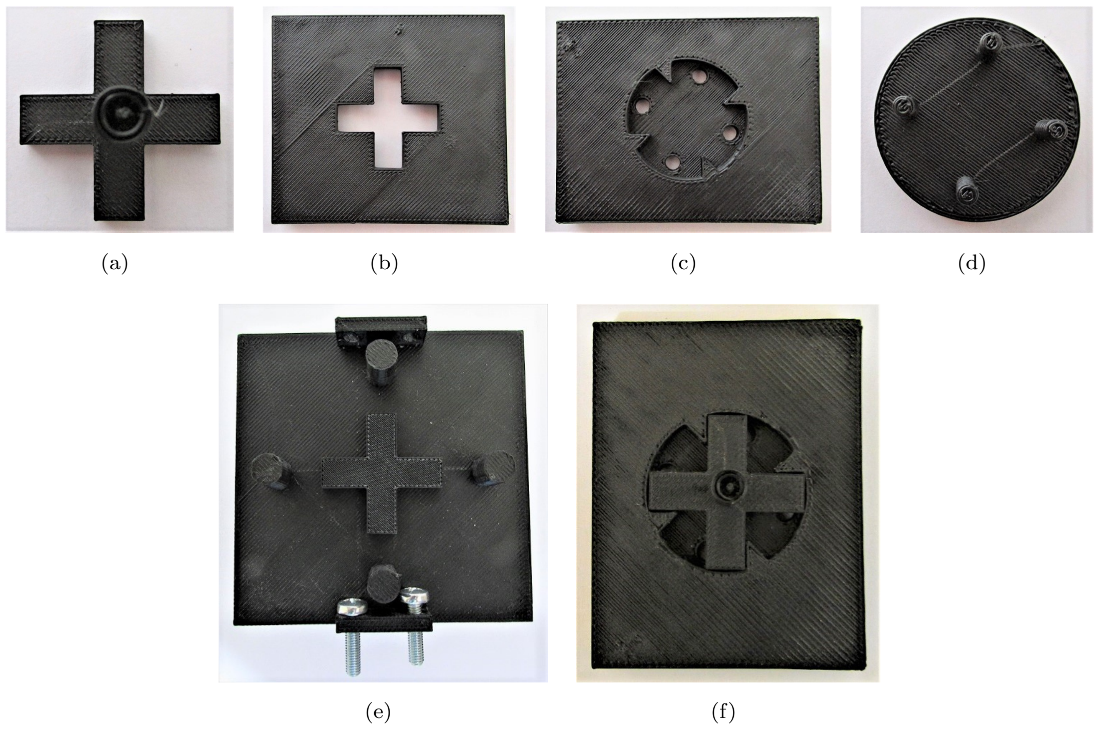


Figure 6.10: CuBot's mechanisms for top-bottom connection to support 3D construction. The cross (a) at the bottom of a robot that will enter a cross-shaped hole of a robot below (b). The cross will touch (c) and rotate clockwise to be secured by (d). (e) The position of the cross below the platform. (f) When the cross enters the hole and hit the locking system.

1. *The main controller.* The ARM Cortex-M3 has been selected as the main controller as it has 512kB flash memory and its energy consumption is low. It has CAN bus as the communication bus.

2. *The perception and locomotion units.* Eight pairs of IR LEDs and IR photodiodes are positioned circular to detect surrounding objects. A pair of edge detectors is placed at the bottom front edge of the robot as they have to detect the void and position themselves in the orthogonal. An accelerometer is embedded to measure the acceleration of the OctBot during navigation. Two pairs of proximity sensor are placed at the corner of the right and left sides of robot to detect the other robot when two robots connect side-by-side. As for matching the connection between two robots at top-bottom position, an IR LED is equipped at the cross key-like, and for the cross hole-like part, it is equipped with an IR photodiode.

3. *The communication unit.* A ZigBee module is equipped to the CuBot for the purpose of broadcast communication. IR photodiodes are also exploited for communication during the coupling process and are placed on every connector.

6.3.3.3 Realisation of the Coupling

A series of initial realisation has been conducted using platforms. Figure 6.11 shows the simulation of a front-rear connection mechanism. When a robot recruits a free robot, it will stay static, as can be seen in Figure 6.11(a). It sends signals in assisting the recruited robot. The recruited one has to locate itself finding the sent signals by moving back and forth once it is in the recruiter's radius (Figure 6.11(b) and Figure 6.11(c)). When its position is in line with the recruiter, as can be seen in Figure 6.11(d), it will move forward until the connectors have entered the recruited connectors and been locked to secure the connection.

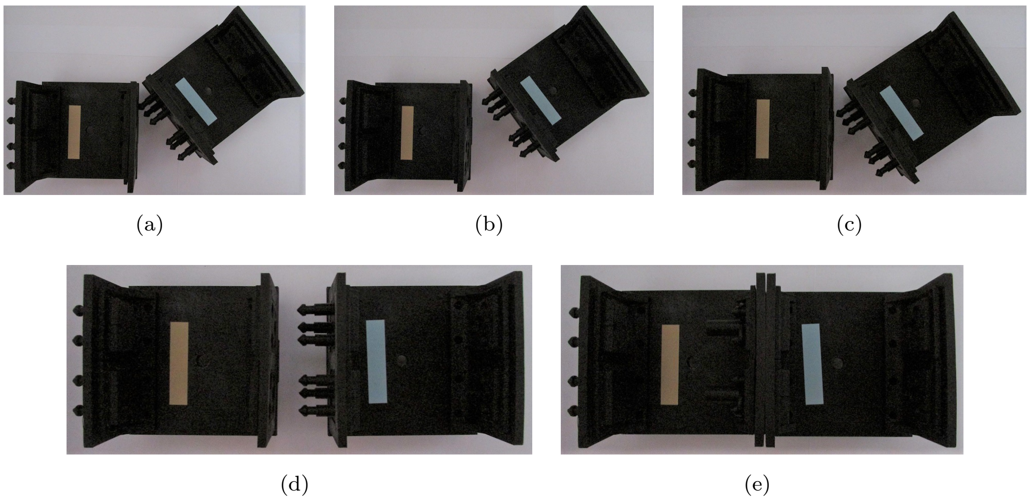


Figure 6.11: Initial realisation of the coupling for a front-rear connection mechanism.

Figure 6.12 shows the simulation of a left-right connection mechanism. Figure 6.12(a) is the left connector of the recruited robot when it has been connected to the front-rear and bottom parts. Figure 6.12(b) is the right connector of the recruiter. Figure 6.12(c) shows the process of connection at the beginning, when the pair of rear communication systems of the recruiter sends signals to assist the front pair of communication systems of the recruited one. When both the robots have been paired, the recruited moves forward and the connection is not only sliding, but secure, as can be seen in Figure 6.12(d). When both front and rear communication systems of both robots have been paired, the recruited stops moving, as can be seen in Figure 6.12(e). They are now ready to recruit other robots.

Figure 6.13 shows the simulation of a top-down connection mechanism. Figure 6.13(a) shows how the connector of the upper robot is placed at the bottom part of the robot. When it has been paired to the connector of the robot below, it will enter the cross-shaped hole, as can be seen in Figure 6.13(b). Below the cross-shaped cover, there is a platform that will hold the upper connector, as can be seen in Figure 6.13(c). Recognising it cannot be pushed again, and the gap is minimum between the upper

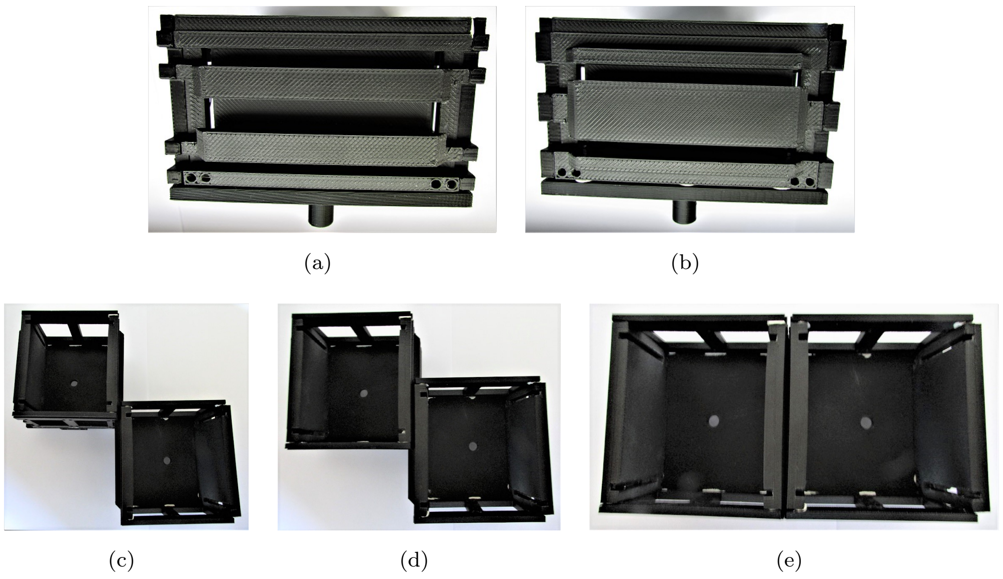


Figure 6.12: Initial realisation of the coupling for a left-right connection mechanism.

connector and the platform, the upper connector will be rotated until maximum, as can be seen in Figure 6.13(d). Then, it will be secured by a locking system so that the fixed connection will be as in Figure 6.13(e).

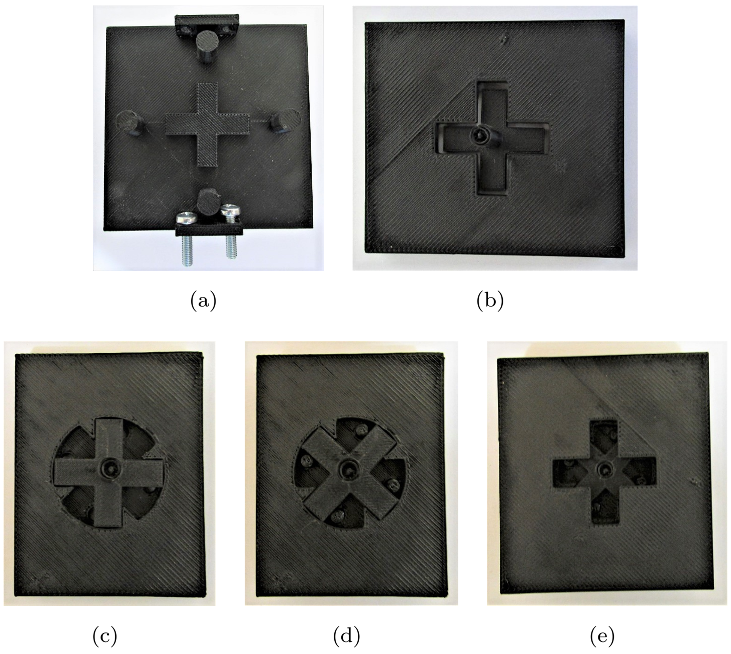


Figure 6.13: Initial realisation of the coupling for a top-down connection mechanism.

6.4 Design Analysis

Considering the requirements for self-assembly swarm robot systems, a number of possibilities have been designed. One design has been built completely, both platform and electronic system, and the other two designs have been 3D printed. For each design, a number of aspects have been considered carefully, including the shape of the robot (Table 6.1), the steering mechanism (Table 6.2), communication and localisation (Table 6.3), and connection mechanism (Table 6.4). In the following design analysis (Voland, 1999), those aspects are evaluated and compared to decide which design is to be chosen through discussion with peers. The largest total, highlighted in red, indicates the best solution, which is obtained from summing all the ratio of multiplication of weighting factors with rating factors.

Comparing a possible number of common robot shapes, four shapes have been considered as the shape of the swarm robot. Although cylinder has many advantages, its weak connection because of its small area of connection is the reason it has not been chosen, as a strong connection is an important aspect for self-assembly. The other option is octagonal prism. Although its connection is not as strong as hexagonal prism or cuboid, it is easy to manoeuvre without collision or move back and forth to avoid collision. To align, it is quite simple compares to cuboid. When connected, the span of the structure is as long as the assembled cylinders and cuboids, and because of it is simple to connect to other robots in four directions in the horizontal.

Comparing a number of steering mechanisms, differential-drive steering has the most advantages. Although it is not stable on uneven terrain, it can be equipped with two ball casters to maintain the balance. In the case of occupying a long robot shape, skid steering can be considered, as it has the same mechanism as differential-drive steering, but it works on track wheels.

Comparing communication mechanisms, radio frequency-based communication has more advantages compared to light-based communication. Its coverage is broader, the data rate is faster and it is able to manage the noise or obstacles such as walls. Comparing three mechanisms of radio frequency communication, although a ZigBee has fewer advantages compared to WiFi, its ability to choose between broadcast and point-to-point communication, in addition to its coverage and reliability, makes it is the most likely device to equip.

In practice, a ZigBee could be used for both broadcast and point-to-point communication when needed. The broadcast communication is used when the seed informs other swarm robots that the void has been located, searches for the closest robot to be recruited, informs others that the opposite ground has been located, informs others to retreat, when the rear robot of the structure searches for the closest robot to be recruited, and when any robot has located the target. For a certain distance, a number of robots other

Table 6.1: Comparison of the basic robot shapes.

	Goal				
	Rigid Connection	*Span*	*Alignment*	*Manouverability*	
	Weighting Factor				
Design Alternatives	*100*	*100*	*60*	*30*	**Total**
Circular	1	10	10	10	2000
Octagon	5	10	8	8	2220
Hexagon	10	3	3	5	1630
Square	10	10	5	3	2390

Table 6.2: Comparison of steering mechanisms.

	Goal					
	Design simplicity	*Manouverability*	*Energy efficiency*	*Speed*	*Stability*	
	Weighting Factor					
Design Alternatives	*85*	*100*	*85*	*85*	*50*	**Total**
Differential-drive	10	10	8	8	3	3360
Skid	10	9	8	7	5	3275
Ackerman	8	7	8	4	50	2650
Tricycle	8	7	8	5	3	2635
Synchronous	4	10	8	5	3	2595
Omnidirectional-drive	3	1	6	10	5	1965
Articulated	3	3	8	3	3	1640
Independent	1	5	4	10	10	2275

than the seed have to act as the broadcaster to prevent loss of communication because of technical limitation.

The point-to-point communication is used when the closest robot to the seed, or to the rear robot of the structure, informs the other party that it has received the recruit signal from the recruiter. Then, both parties communicate with each other and, by using this mode to guide, the recruited moves towards the recruiter. The information that is given to each other is their distance and direction. The recruiter has to guide the recruited until it reaches a certain distance and faces the rear side of the recruiter to make them ready for the coupling process. This communication mode is also used when the seed informs the rear robot of the structure to continue recruiting, and when it disassembles and reaches a certain distance from the subsequent seed robot.

A communication using IR proximity sensor is also used to assist the coupling process. The recruiter transmits the IR signals and the recruited has to detect those signals to a couple of certain receivers. The recruited has to position itself, adjusting to the transmitted signals for the alignment. Once they are aligned, the recruited has to move towards the recruiter until their distance is minimum and each connectors ready for coupling.

Hooks and pins and hole mechanisms have the stronger coupling compared to gripper and probe and drogue mechanisms. This is because they can be placed in several locations on a surface to share the weight. In practice, both systems can also be implemented in smaller size compared to the two others. Based on several evidences, the power consumption of the gripper is the highest, as a strong coupling depends on the power. For the connection speed, once two robots are facing each other and ready to couple, pin and hole mechanism has the fastest coupling process as it also depends on its mechanics system. Hooks and pin and hole mechanisms also have compactness; where two robots have coupled there is only a small space left. However, in the two other mechanisms there is quite a large space between two robots after coupling. As for the risk of slipping during the coupling process, pin and hole mechanism has the settled mechanism to prevent slippage.

Another aspect to consider is the size and the weight of the swarm robot, which affects the alignment and coupling mechanism. Choosing a large robot size will have the effect of greater weight of the robot. This aspect influences the large deflection occurred in the structure, which makes the span shorter. However, large robot size gives more space for a coupling mechanism, which means a strong coupling can be formed. This is also an opportunity to construct a complex structure for a large crossing that has a long span. However, as a robot swarm resembles the swarm of a biological system, the size of the robot has to be made as small as possible.

Small size of robot results in less weight of swarm robots and creates less deflection of the structure. Therefore, a long span can be achieved. However, small size means less

Table 6.3: Comparison of communication methods.

	Goal						
	Coverage	*Non-complex system*	*Frequency/data rate*	*Broadcast ability*	*Energy efficiency*	*Reliability*	
	Weighting Factor						
Design Alternatives	*100*	*20*	*100*	*100*	*40*	*85*	**Total**
Bluetooth	5	10	8	7	10	7	3195
WiFi	10	10	10	9	3	8	3900
ZigBee	8	10	7	10	5	8	3580
Visible	3	3	3	7	7	5	2065
Infra Red	3	10	3	5	7	3	1835

Table 6.4: Comparison of mechanical connection mechanisms. Applied (a) hooks is in ATRON (Jorgensen et al., 2004), Sambots (Wei et al., 2010) and M^3Express (Wolfe et al., 2012); (b) gripper is in s-bots (Mondada et al., 2004) and marXbots (Bonani et al., 2010b); (c) pin and hole is in (Castano et al., 2000), (Yim et al., 2003) and (Liedke and Worn, 2011); and (d) probe and drogue is in OctBot.

	Goal						
	Strength	*Size*	*Power consumption*	*Connection speed*	*Compactness*	*Rigidity*	
	Weighting Factor						
Design Alternatives	*100*	*50*	*20*	*20*	*50*	*90*	**Total**
Hooks (a)	9	10	10	8	10	5	2710
Gripper (b)	7	6	5	9	6	3	1850
Pin and hole (c)	9	10	10	10	10	10	3200
Probe and drogue (d)	80	5	10	6	6	9	2570

space for the coupling mechanism, so that it will have a weak coupling. In terms of impact, the span of the structure will be short. Considering the problem of robot size and weight, it is better to choose an optimal size and weight of the swarm robot. It has to be not too large or too small, so that the structure of assembled robots has a particular deflection and particular span. To manage the large crossing problem, each robot has a particular space for a coupling mechanism to support the particular coupling strength for achieving a particular span.

Sensor placements have to be considered carefully, which affects the safety gap and detecting objects surrounding the robot. In the vertical, it is important to place sensors according to the safety gap between robot and objects underneath in order to prevent a collision between them that can cause the failure of the robot or getting stuck because of the object. With its sensor placed in a particular position below, the robot is able to detect objects underneath or avoid falling into the void. In the horizontal, sensor placements influence the ability to detect surrounding objects in any height to prevent its whole body or a part colliding with other objects.

6.5 Stability of the Proposed Structure

By using a number of connected simplified platforms, it has been proven that the algorithm of crossing a small void works. The set relation of robot number over the void and the ground can maintain structural stability, so that the structure will not fall into the void. However, there are a number of emerged problems, such as the robot's manoeuvrability and the deflection occurred.

Taking a cuboid shape is not as advantageous as taking an octagon or cylinder shape. The coupling process is not as smooth as robots with the two other shapes, because there are edges that limit the sensing and the movement. However, cuboid shape brings advantages of convenient sensing for detecting the edge of the void, and a strong connection mechanism, as a pair of connectors can be placed on a surface.

6.5.1 Example Systems

A number of platforms have been 3D printed using ABS plastic material, in Figure 6.14. The platform dimension is 10cm length, 10cm width and 5mm height. The platform is designed to provide similar ground clearance to the robot. Coupling is by a nut and bolt for simplicity.

Referring to the challenge of structural stability and to prove the approach of simple line structure in Chapter 4, a number of case studies have been taken for crossing one robot wide and two robot wide voids. For one robot wide void there are two demonstrated

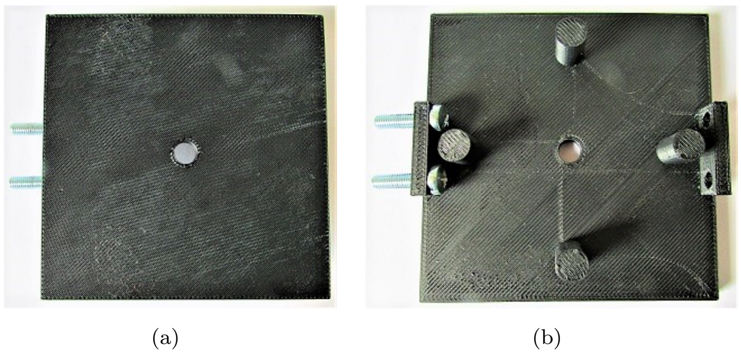


Figure 6.14: A simple platform has been used to test the structural stability and deflection aspect of assembled swarm robots. (a) Top view of the platform. (b) Bottom view of the platform.

scenarios using a swarm consisting of three platforms and four platforms, respectively. For two robot wide void, the scenario is demonstrated using six platforms. The point of taking case studies using a real practical platform here is to prove that the relation $N = 3n_g$ works, the structure can maintain its stability and the deflection is minimum. Therefore, a number of snapshots only focus on the crucial time of these highlighted points.

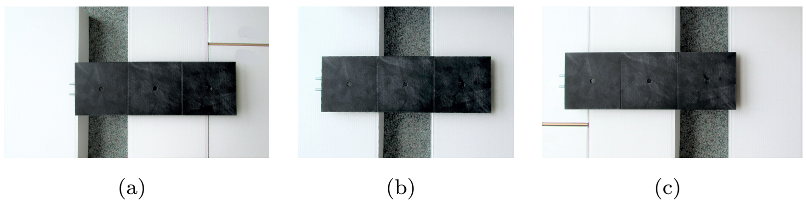


Figure 6.15: Using the dummy platforms to illustrate crossing a one robot wide void.

In a one robot void wide crossing, as illustrated in Figure 6.15, the first crucial time is when the seed robot is pushed by two other following robots over the void, as illustrated in Figure 6.15(a). The structure has to maintain its stability so that it does not fall into the void. Here, the relation $n_p > n_s$ has been proven. The next crucial time is when a robot over the void is held by two other robots that are on the ground to prove the relation of $N = 3n_g$, as illustrated in Figure 6.15(b). The last is when most of the structure is on the ground, but the rear robot is still over the void. The structural stability can be maintained and is related to the relation of $n_o > n_g$, as illustrated in Figure 6.15(c).

The case of four robots crossing a 10cm void, as illustrated in Figure 6.16, there are extended important points to show. After the structure consisting of three robots has settled, forming the bridge, as illustrated in Figure 6.16(b), it has to maintain its stability when a robot joins to the rear of the structure. A good grips to the ground plays an important role as well as less force from the joining robot when it is coupling, as

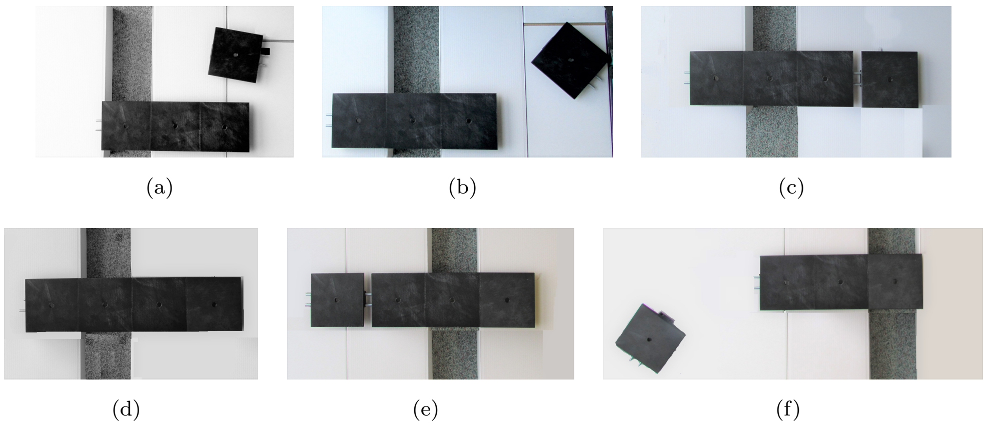


Figure 6.16: The illustration of four dummy platforms crossing a one robot wide void.

illustrated in Figure 6.16(c). Then, the structural stability has to be maintained during the transition to push the front robot to the target zone, as can be seen in Figure 6.16(d) and Figure 6.16(e). A strong gripping to the ground is also required when the front robot disassembles from the structure, as illustrated in Figure 6.16(f), to prevent friction that can affect the structural stability.

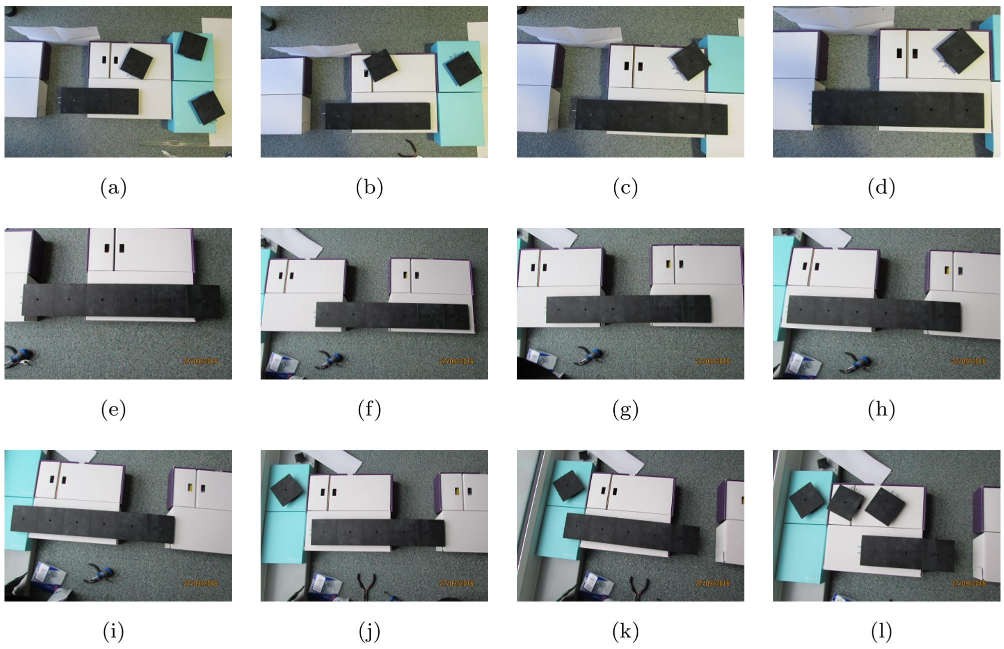
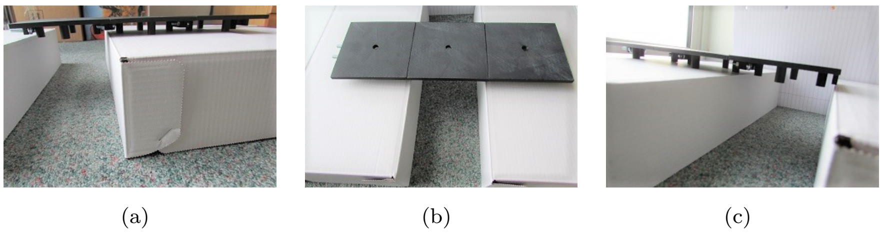


Figure 6.17: A six platforms structure crossing for two robot wide void.

In performing void crossing for two robot wide void using six robots, as can be seen in Figure 6.17, all structural stability during the process of delivering all robots to the target zone has been maintained. The first stability has been maintained when the seed is pushed by two other robots, as illustrated in Figure 6.17(a). It has to be maintained when two other robots join the structure and the factor of the robot's grip to the ground plays an important role, as can be seen in Figure 6.17(b) and Figure 6.17(c). A robot is then pushed towards the opposite ground as the relation of $n_p > n_g$ is maintained, as can be seen in Figure 6.17(d). The structure requires a robot to further maintain its stability during the bridge mode, where $n_p = n_g = n_o$, as can be seen in Figure 6.17(g). After the required number of robots reach the target ground, the stability has to be maintained by applying $n_o > n_g$, as illustrated in Figure 6.17(j) to Figure 6.17(l).

6.5.2 Deflection

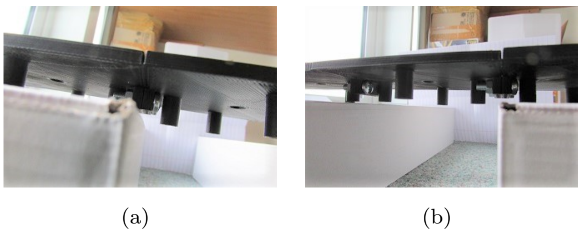
Referring to the decision of changing the structure based every five robot wide void, as related to Chapter 4 and Chapter 5, a number of tests have been conducted. The deflection of the structure can affect the structural stability and the effort to reach the opposite ground. With a flat structure, robots will not require additional sub-systems to reach and step on the opposite ground. Three cases were taken to see whether there is deflection for the structure constructed by assembled platforms, these are three, four, and five platform structures.



(a) (b) (c)

Figure 6.18: The deflection problem of three connected platforms.

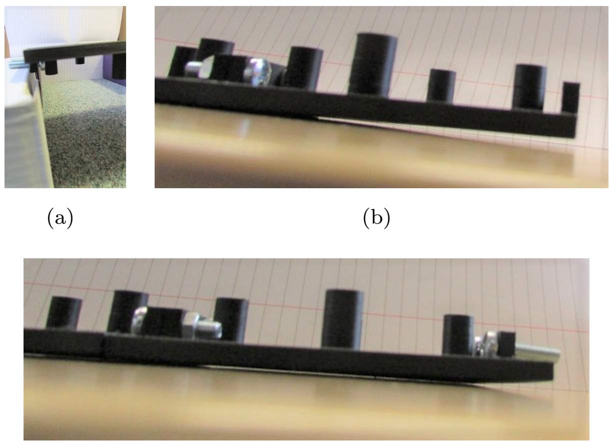
For three assembled platforms, there is only a slight deflection occurred, as can be seen in Figure 6.18. This deflection did not affect the structure when it had to reach the opposite ground.



(a) (b)

Figure 6.19: The deflection problem of four connected platforms.

A slight deflection was occurred at the four assembled platforms, but it did not affect the structure when it was reaching for the opposite ground. The connection between two platforms was set very tight. However, because of the platform imprecision, there is still a small gap between two platforms, as can be seen in Figure 6.19. This small gap results in a deflection that impacts on the whole structure.


(a) (b)

(c)

Figure 6.20: The deflection problem of five connected platforms that caused the end of the structure deflected at 5mm.

A deflection was occurred for five connected platforms because the connection was only at one horizontal point, which caused up to 5mm deflection at the end of the structure, as can be seen in Figure 6.20. For two robot wide void, this problem can be overcome because the front caster wheel will help the structure to climb the cliff to reach the opposite ground. The assumption that 1mm deflection will be occurred for one platform is taken from the case of 5mm deflection for five connected platforms. For the longest structure case, when it has to cross a five robot wide void and requires at least fifteen platforms connected together, the deflection might reach 15mm. This number equals to the height of the main wheels outside the base of the robot where the risk of hitting the cliff and the inability of climbing the cliff to reach the opposite ground are highly possible. Moreover, the diameter of caster wheels is only 10mm, which height outside the base is only one third of the diameter. Therefore, it has been decided that the structure has to be changed every five robot wide void to prevent deflection.

6.5.3 Issues with Practical Systems

Referring to the practical systems, the solution to which has been answered in this chapter, several issues are still emerged. The gap between two robots through their connectors cannot be avoided. There is still a small gap between connectors as a result of 3D printings not being as precise as expected. However, using a pair of connectors in which each has a triangular arrangement of pins or holes, the coupling is stronger so as to prevent deflection in the vertical. The deflection only occurs in the horizontal plane, which does not have a large effect on the deflection of the structure.

In answering the bending challenge, a number of parts have been 3D printed, specifically, the left-right and top-down connection mechanisms. However, the mechanism to lift a robot up has not been designed, but a concept has been considered.

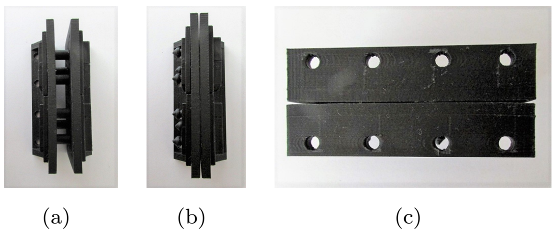

(a) (b) (c)

Figure 6.21: Gap issue of practical systems that can result in deflection.

Printing a part should be conducted in a certain direction; mostly, the wide area is put on the platform of the machine. When a thin and long surface is put on the platform, there is a high possibility of resulting in a bended part. There is also no guarantee that the perfect printed part will not be broken in use. Because of the direction in printing, a crucial part, such as the pin of a connector, can be accidentally snapped in use. The broken part can be stuck again with acetone or super-glue, but the risk of snapping again is still possible. Making the part stronger can be done by putting it in a vacuum container filled with acetone at the bottom for a certain range of time. However, this process has the risk of melting all parts, so it was avoided.

This chapter has considered the design and practical consideration of swarm robots capable of forming a structure autonomously with the algorithms discussed in Chapter 4 and 5. It is clear that these robots will be relatively complex and need careful design and efficient manufacturing process.

Chapter 7

Conclusion

In this chapter, conclusions of the work that covers the research questions and their solutions is presented, followed by a discussion of future work.

7.1 Conclusions

In concluding this work, there are several highlighted aspects related to the research questions. These aspects have been addressed to answer the main challenge, implementing an adaptive ability to swarm robotics to allow void crossing autonomously.

A simple self-assembly rule can be achieved by applying the existing construction mechanism; the bridge launching mechanism. Here, robots in the structure recruit free robots that are exploring the initial zone to join at the rear of the structure and, after a preset number is achieved, the structure moves forward. This preset number is implemented by simply adding two rows of robots, then it moves forward for one row. Here, the structural stability is maintained to prevent the structure falling into the void, and all robots are successfully delivered to the opposite zone when their number in the swarm is compatible to the void width. Dynamic variables are also applied to optimise the mechanism. They also help in decreasing the use of each robot's memory.

The simplest structure, the single line structure, was used first for all approaches, as this structure guarantees a simple and fast process. As long as the rule is maintained, the structure is stable and able to deliver the swarm in reaching the opposite zone. However, a problem of deflection emerged, so that this single line structure required to be supported by another line to maintain its strength and stability. Taking a more reliable approach, after reaching a certain void width, the structure developed into a 3D structure which could maintain the stability and minimise the deflection of the structure. Here, the combination of bridge launching mechanism and the principle of a telescopic antenna was used. However, this approach also encounters a span limitation because of a deflection problem at some point. An advanced structure needs to be further applied, but these strategies have more advantages compared to others because of the span reached by the robots.

Designing a suitable swarm robot for this application domain is also challenging as there is a size limitation, but robots must have a strong connection mechanism to support them in constructing a strong and stable structure. By prioritising the strong connection mechanism rather than their manoeuvrability, the square based robot is chosen rather than other shapes. It also gives more space for connector placements.

The performance of the swarm in crossing the void has been measured by recording their variables, simulation time and its details, and also using captured video to see if the strategy worked successfully. The robustness of the strategy is achieved when all robots have the same opportunity to initiate the construction and detect the target. So, if there are robots stuck together or having failure, there are still several robots that can accomplish the task. In addition, when the number of robots does not satisfy the void width, they can retreat to the initial zone. Flexibility is achieved when any number of robots can be added into the swarm and they can perform the same, in addition to the challenge of any size of void to be crossed. Scalability can be achieved by putting any number of robots in the swarm performing the same task for a certain void width.

This approach has considered the general requirements of swarm robotics: homogeneous robots, low cost platforms, and in addition there is no predefined leader in the swarm. These bring the advantage of decreasing the risk of failed operation because the swarm has no dependency to a leader or a specific type of robots to maintain the operation. A strategy of retreating to the initial zone has a purpose to prevent the loss of robots during operation so that the swarm can do other operations later. The proposed relatively simple self-assembly rule helps minimising the computation a robot should do for a fast operation, reducing the complexity for scalability aspect in swarm robotics. Proposing such strategy that relies on a rigid and long span structure can bring an advantage of crossing a wider void compared to the size of a swarm robot, and this strategy is also supported by the scalability. In a specific case of practical search and rescue operation, this approach will help the faster response because of its minimum computation, reach a zone separated by a wide void because of the structure and scalability, and the strategy can be run repeatably to cross other voids and reach a wider operation areas while the loss of robots can be minimised. This approach can also be applied as the base to construct a more complex structure e.g. a shelter with a more complex strategy and a set of rules.

7.1.1 Research Contributions

The reviewed literature indicates that the best solution for bridging two separated zones where there is a void in-between for swarm robotics is by using the self-organised assembly approach. However, many approaches did not consider the scalability aspect. Several of them have also met an additional adaptive ability problem when the environment changes.

Research questions that have been proposed in obtaining the approach are as follows.

RQ1, defining simple self-assembly rules. As discussed in Chapter 4, the process to find simple rules for self-assembly in swarm robotics crossing a void required several long processes. Several algorithms have been proposed and proved the evolution process to find the most simple rules with specific requirements of adaptive ability, scalability and structural stability resulted in the algorithms identified as cases A, B, C and D for simple line structure.

RQ2, defining the shape of the structure. After simple rules are implemented, the next step is to consider the effective structure to maintain the structural stability when the void is wider but there is a span limitation because of the deflection. The challenge is that it has to be implementable in practical systems. This research question is addressed in Chapter 5.

RQ3, designing the swarm robot. Several designs have been realised and analysed. Considering a number of aspects, a cube-like shape with connection mechanisms at the front-rear, left-right, and top-down has been chosen to implement the self-assembly approach in swarm robotics for crossing a void. This research question is addressed in Chapter 6.

RQ4, defining metrics. The purpose of developing a Matlab-based simulation environment is to support the proposed approach implementation. The metrics that would be used to measure the performance of a robotic swarm crossing a void have been decided in this development process, as discussed in Chapter 3.

7.1.2 Simulation Environment

As discussed in Chapter 3 to address in implementing the approach of adaptive ability for swarm robots in crossing a void, the first important thing is how to simulate the approach. A number of simulation environments have been tried to simulate scalable swarm robots. However, features in those simulation environments didn't meet the requirement.

The most important considered thing for a simulation environment is whether it can accommodate swarm robotics. There are many simulation environments for robotics, but most of them can only accommodate a single robot. There are also a number of simulation environments for multi-robot systems (Claytronics, 2014; Open-Source-Robotics-Foundation, 2014; SeSAm, 2012), but they are not specific for swarm robotics.

Simulation for swarm robotics requires two main things: decentralised control and scalability. With decentralised control, each robot in a swarm is autonomous, it can act on its own without receiving an order from another. In the developed simulation environment, it has been proved that all robots can explore on their own, detect the void and the target, initiate the construction, recruit to and be recruited by other robots. Indeed, in the construction, the seed and all robots at the front of the structure lead other robots to reach the opposite ground, but all robots have the equal opportunity to lead. Scalability also plays an important role in swarm robotics. A swarm will not determine its size, as any robot can be added to or leave the swarm. Robots have to work collaboratively regardless the size of the swarm. In the developed simulation environment, users can add any number of robots as easily as typing the number, without copying and dragging robots one by one. In several existing simulation environments, these features are already provided. However, there are also several things that are not provided or need to be added.

In several simulation environments, it is not easy to define the geometry of swarm robots as users require. Many of them are developed specifically for certain robots, so it is difficult for users who use other types of robots to simulate their approach. As the proposed approach of the thesis emphasises self-assembly with shape and connection mechanism considerations, defining a specific geometry for the robot is a must. Additionally, the geometry of a robot can be modified for further purpose.

As the approach is only simulated in a simple way to show how exploration and self-assembly work, it can be conducted only in a 2D manner. The connection mechanism itself is not simulated, but the required time can be estimated on how the robot was recruited, positioned itself and connected to the recruiter. Several simulation environments apply 3D simulation, which makes the running simulation too complex and very slow, while the detail can be decreased to only demonstrate the important element of the approach. Here, the challenge has been answered by implementing the simulation by applying a top view of the arena and there is no requirement to add a specific HD graphic card to support a 3D environment.

The other challenge is that the simulation should be able to be recorded. For several simulation environments, this feature is not provided, so it is difficult to capture pictures or record the whole simulation for evidence. In several of them, it is because the simulation itself is too complicatedly heavy to run effectively, or the resulted record will be too large because of the HD graphic. With 2D simulation and enabled graphic data recording, the evidence can be saved.

While all challenges have been answered, slow simulation when more robots are added to the swarm cannot be avoided. In all simulation environments, this problem is occurred, and, in several environments the simulation even stopped or crashed. It has been found that adding thousands of robots to the arena will not make the simulation stop. It will still run, but slowly.

7.1.3 The Structural Stability of the Proposed Algorithms

Referring to **RQ1** and **RQ2**, solutions to which have been proposed in Chapter 4 and Chapter 5, the approach of crossing the void using swarm robotics deploys robots and enables them to perform self-organised assembly. The strategy mainly applies the launching mechanism of a bridge, but the structure is dynamic, so it is important to maintain the structure stability by applying a certain relation between robots on the ground and over the void. It is also important to securely connect each robot to another.

The approach of void crossing is divided into three strategies: i) one robot wide or a simple line structure for a void that size is one to five robot wide; ii) two robot wide or two-line structure for void sized six to ten robot wide; and iii) 3D structure for a void sized more than ten robot wide. There are several reasons in determining the strategy.

For the cantilever beam, the cross-sectional area $A = dh$ (mm^2) is calculated, where d (mm) and h (mm) are the depth and the height of a swarm robot. The load per unit length $W = A\gamma g$ (N/mm) is calculated where γ (kg/mm^3) is density and g (m/s^2) is gravity. Therefore, the deflection of the bridge δ (mm) can be calculated as shown in Equation 7.1 (Gere and Goodno, 2009).

$$\delta = \frac{WL^4}{8EI} \tag{7.1}$$

where L (mm) is the length of the bridge, E (N/mm^2) is Young's Modulus, and I (mm^4) is the area moment of inertia. The deflection and the length of the cantilever can be compared. If deflection is greater than half diameter of the robot's wheels, then the swarm robots satisfactory climb on to the destination side of the void.

In calculation, the deflection of a connected ABS plastic cube will result in only a small deflection, so that it is alright to cross a void the size of which is more than ten robot wide with only a single line structure. However, connecting five 3D printed platforms resulted in 5mm deflection, it is estimated that the deflection of fifteen connected platforms will be 15mm or the same as the height of wheels outside the platform, it is, therefore, safe to limit the maximum void width at five robot wide for single line structure.

Based on the reason of the deflection at single line structure, the same reason has been taken to apply the structure of two robot wide line and 3D structures. In addition,

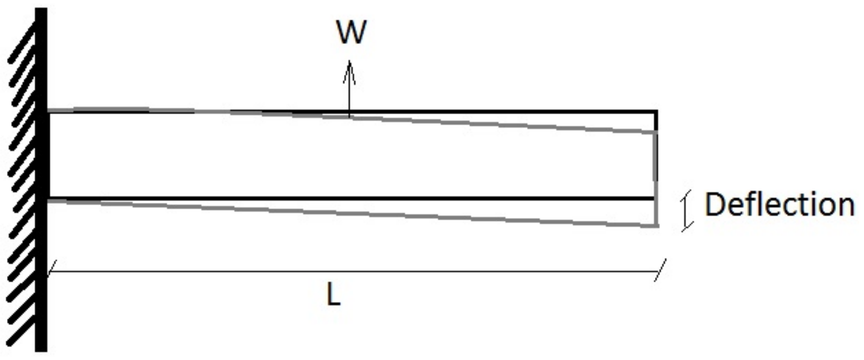


Figure 7.1: Deflection occurred on a cantilever.

taking the based number of five for each shape of structure makes it easier to implement into the simulation.

In both fixed and random initial position of the swarm, where the size of the swarm is scalable, the time required to accomplish the task increased when the number of robots in the swarm increased. However, as the exploration time at the initial zone decreased as more robots increases the probability to detect the void faster, the required time to construct the structure therefore increases, as more robots require more time for recruitment and coupling. Meanwhile, the scalability does not affect the exploration time at the target zone.

However, the trend of scalable swarm happened differently for fixed swarm size, as can be seen in Section 5.6. The fixed size emphasises the effect of the void width on the accomplishment time. For the exploration time at the initial zone, the time decreased when the void became wider, or the exploration area decreased. Therefore, the probability to detect the void increases when the area decreases. As for the mean construction time, at some point it increased, but then the trend decreased, although the assembly and disassembly time increased when the void was wider. The unexpected non-linear trend happened because of the random position of robots, which caused more required time for recruitment. Similar to scalable swarm, there is also no increased or decreased trend for exploration time at the target zone.

The proposed approach applying dynamic bridge launching mechanism answers the challenge of adaptive ability where a swarm has to find the target, but there is a void that separates the location between the target and their initial positions. They are able to cross a certain size of void without *a priori* knowledge of the void width and position. They also do not have any knowledge of their number in the swarm.

Indeed, the approach is in applying a simple rule, embedded to the robots to determine if they have to continue the construction or retreat, or decide the shape of the structure after detecting whether there is a landing ground at the opposite. However, despite applying and implementing more complex methods, a simple method is better for swarm robotics because the limitation of the requirement of simple robots, although they can work collaboratively. With this simple approach, the robot's processor will not be used for a heavy program. Instead, it can be used for optimised sensors and actuators.

7.2 Future Work

In order to achieve optimal performance and to ensure that the crossing of the void is fully reliable using both simulations and practical systems, additional work will be required.

1. The whole 3D structure has to be implemented and the results recorded. It is not only limited to eleven to fifteen robot wide void but also up to twenty six to thirty robot wide void. As proposed and partly implemented in Chapter 4 and Chapter 5, the structure can be expanded up to thirty robot void width, but the implementation is only up to a two-line structure that covers up to ten robot wide void. The performance can be predicted, such as simulation time, where the wider the void, the longer the simulation with the same swarm size. However, it is better to know the time required to simulate the strategy, as there is a probability that the trend might be different when the swarm size reaches thousands.

2. The electronic system to the 3D platforms needs to be implemented so that the swarm robot can work with full features. Practical systems have their own challenge and, if the approach can be implemented in practice, more advantages can be obtained to answer the actual adaptive ability in real world. A number of sub-electronic systems have been proposed to be embedded into the platform (see Chapter 6) and it is better to see if they work well in the practical systems.

3. An approach using an evolutionary learning process to optimise the sensing and actuating systems needs to be implemented so that swarm robots can perform better. A number of approaches used evolutionary systems in deciding the action to take for the swarm based on the obtained sensing, such as the swarm has to avoid or cross a void. Other approaches proposed the evolutionary learning process so that the swarm can learn and evolve its own rules in accomplishing a task. Therefore, by implementing such an approach, there is a possibility to gain more effective and simpler rules to apply.

4. An approach combining self-organised assembly and fault detection for more advanced structure needs to be developed. A strategy depending on only two connected robots certainly will meet a limitation in span. Taking a look at the real

bridge that has piers to support the span, robots in failure can be used for supporting pillars so that the span of the structure becomes longer. Therefore, a fault detection task is required to detect which robots are in failure, and how to bring and pile them up in constructing the piers across the void. In addition, the simulation environment might be modified to observe the side view of the constructed structure.

Appendix A

Extended Results of Small Void Crossing

The following contents show the extended results that are not included in Chapter 4 for cases A, B, C and D. It consists of the frequency of robot ID that became the seed robot, the simulation time, the positions of seed robots when detected the void, the frequency of robot ID that detected the target, and snapshots of simulations.

A.1 Case A

Extended results for case A only consist of the frequency of robot ID became the seed robot and the recorded simulation time including T_s (time for robots exploring the initial zone), T_a (timefor robots assembling forming a structure and crossing the gap if the number of robots is enough;), T_o (time for robots exploring the target zone when all robots have disassembled from the structure), T_d (time for the structure retreat to the initial zone and robots disassembling), T_b (time for robots exploring the initial zone because they failed to cross the gap) and T_t (simulation time).

A.1.1 Frequency of Robot ID Became the Seed Robot for Case A

The following figures, Figure A.1, Figure A.2, Figure A.3 and Figure A.4, show the frequency of robot ID became the seed robot for case A for one, two, four and five robot wide void, for five, seven, eight and ten swarm size. As robot positions were fixed and the larger the swarm the closer to the void for robots having the larger ID, they were detecting the void faster than other robots, Figure A.4.

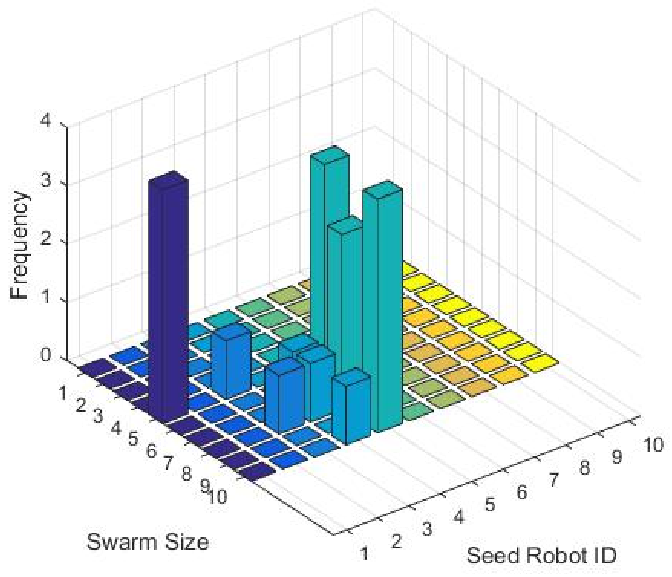


Figure A.1: Frequency with which a specific robot became the seed robot for case A for one robot wide void.

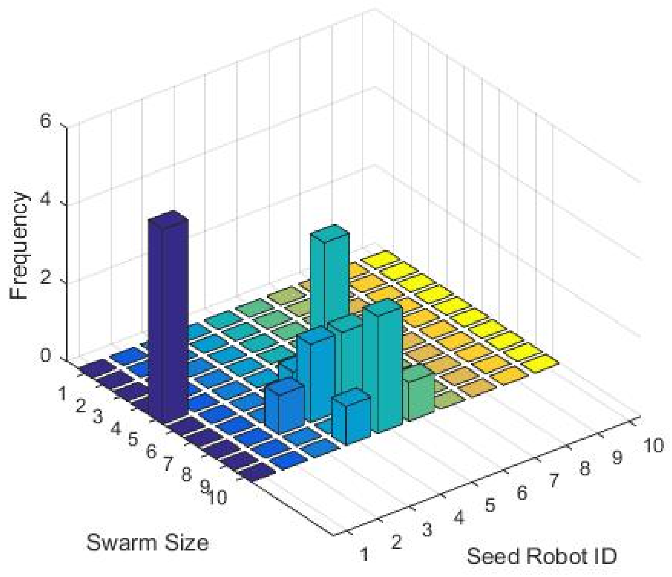


Figure A.2: Frequency with which a specific robot became the seed robot for case A for two robot wide void.

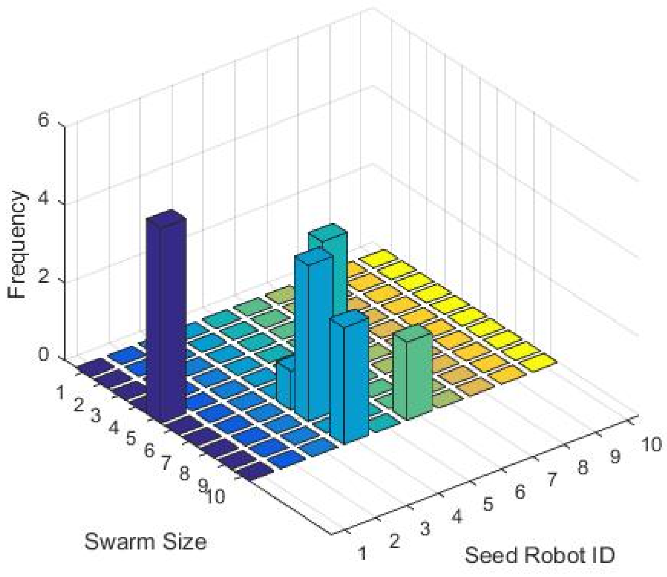


Figure A.3: Frequency with which a specific robot became the seed robot for case A for four robot wide void.

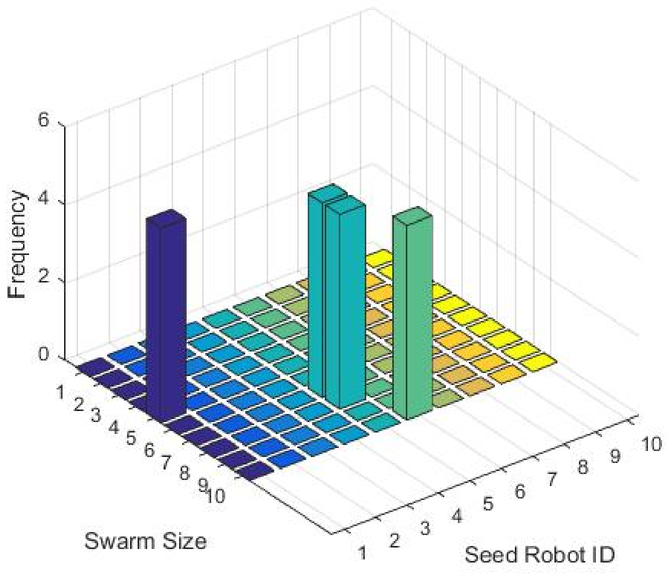


Figure A.4: Frequency with which a specific robot became the seed robot for case A for five robot wide void.

A.1.2 Simulation Time for Case A

The following tables, Table A.1, Table A.2, Table A.3, Table A.4 and Table A.5, show the recorded time (in s) in simulations including T_s, T_a, T_o. T_d, T_b and T_t (in seconds) for case A, for one, two, three, four and five robot wide void. N is the swarm size and RN is the running number.

Table A.1: Recorded time in a one robot wide void for case A.

N	**RN**	T_s	T_a	T_o	T_d	T_b	T_t
5	1	2.33	49.15	116.27	0	0	167.75
	2	2.42	43.74	7.58	0	0	53.74
	3	2.58	56.79	102.16	0	0	161.53
	4	6.13	72.75	45.99	0	0	124.87
	5	2.39	46.01	106.65	0	0	155.05
7	1	3.32	77.58	5.7	0	0	86.6
	2	3.52	78.23	7.99	0	0	89.74
	3	2.46	79.97	2.7	0	0	85.13
	4	2.47	85.94	26.07	0	0	114.48
	5	3.77	87.68	31.49	0	0	122.94
8	1	2.96	115.98	142.29	0	0	261.23
	2	2.51	90.68	1.95	0	0	95.14
	3	6.36	134.64	20.52	0	0	161.52
	4	2.37	93.24	19.13	0	0	114.74
	5	4.08	90.08	23.64	0	0	117.80
10	1	2.38	134.82	11.3	0	0	148.5
	2	4	138.03	71.49	0	0	213.52
	3	2.36	126.53	5.7	0	0	134.59
	4	2.5	131.28	1.17	0	0	134.95
	5	2.75	123.52	38.31	0	0	164.58

Table A.2: Recorded time in a two robot wide void for case A.

N	**RN**	T_s	T_a	T_o	T_d	T_b	T_t
5	1	1.92	26.78	0	14.17	5.01	47.88
	2	2.33	30.02	0	14.26	5.01	51.62
	3	2.13	31.91	0	14.49	5.01	53.54
	4	2.03	30.26	0	14.39	5.01	51.69
	5	10.13	54.67	0	14.25	5.01	84.06
7	1	7.94	61.18	0	29.53	5.01	103.66
	2	1.99	47.07	0	29.58	5.01	83.65
	3	3.85	62.28	0	28.81	5.01	99.95
	4	2.17	55.04	0	29.76	5.01	91.98
	5	5.28	60.98	0	29.51	5.01	100.78
8	1	2.33	90.63	51.03	0	0	143.99
	2	3.22	96.31	6.46	0	0	105.99
	3	3.8	100.55	2.36	0	0	106.71
	4	3.15	92.1	67.21	0	0	162.46
	5	2.71	92.47	51.75	0	0	146.93
10	1	1.95	136.49	15.07	0	0	153.51
	2	2.02	133.84	21.14	0	0	157
	3	2.17	133.01	96.61	0	0	231.79
	4	2.57	125.64	53.26	0	0	181.47
	5	3.2	126.06	3.54	0	0	132.8

Table A.3: Recorded time in a three robot wide void for case A.

N	**RN**	T_s	T_a	T_o	T_d	T_b	T_t
5	1	1.49	26.09	0	14.23	5.01	46.82
	2	1.96	29.5	0	14.12	5.01	50.59
	3	2.19	33.48	0	14.38	5.01	55.06
	4	1.91	28.87	0	14.05	5.01	49.84
	5	1.52	24.29	0	14.45	5.01	45.27
7	1	6.26	57.54	0	28.79	5.01	97.6
	2	1.68	57.88	0	28.68	5.01	93.25
	3	4.8	74.88	0	26.55	5.01	111.24
	4	1.76	51.07	0	27.13	5.01	84.97
	5	1.5	48.6	0	23.93	5.01	79.04
8	1	2.51	64.71	0	23.9	5.01	96.13
	2	1.51	53.97	0	23.82	5.01	84.31
	3	1.51	57.37	0	24.9	5.01	88.79
	4	2.16	52.29	0	23.96	5.01	83.42
	5	7.43	84.69	0	24.28	5.01	121.41
10	1	6.75	96.98	0	39.94	5.01	148.68
	2	1.52	85.24	0	41.26	5.01	133.03
	3	12.29	109.51	0	40.06	5.01	166.87
	4	1.71	83.95	0	41.22	5.01	131.89
	5	1.46	95.86	0	40.13	5.01	142.46

Table A.4: Recorded time in a four robot wide void for case A.

N	**RN**	T_s	T_a	T_o	T_d	T_b	T_t
5	1	1.32	26.25	0	14.13	5.01	46.71
	2	1.27	34.81	0	14.12	5.01	55.21
	3	1.17	28.78	0	14.33	5.01	49.29
	4	1.13	25.89	0	14.48	5.01	46.51
	5	1.13	32.46	0	14.14	5.01	52.74
7	1	1.17	58.96	0	22.85	5.01	87.99
	2	4.97	69.79	0	23.01	5.01	102.78
	3	3.37	62.24	0	23.74	5.01	94.36
	4	1.23	61.26	0	22.77	5.01	90.27
	5	1.08	49.7	0	22.56	5.01	78.35
8	1	1.13	63.09	0	23.78	5.01	93.01
	2	24.87	78.75	0	23.63	5.01	132.26
	3	1.08	55.13	0	23.8	5.01	85.02
	4	4.49	72.49	0	23.86	5.01	105.85
	5	1.16	60.39	0	23.37	5.01	89.93
10	1	1.07	82.76	0	40.83	5.01	129.67
	2	1.23	89.52	0	41.3	5.01	137.06
	3	1.15	98.96	0	39.98	5.01	145.1
	4	1.1	98.26	0	40.01	5.01	144.38
	5	1.11	83.35	0	41.26	5.01	130.73

Table A.5: Recorded time in a five robot wide void for case A.

N	RN	T_s	T_a	T_o	T_d	T_b	T_t
5	1	0.69	31.31	0	14.36	5.01	51.37
	2	0.65	26.07	0	14.25	5.01	45.98
	3	0.72	31.16	0	14.14	5.01	51.03
	4	0.68	28.85	0	14.07	5.01	48.61
	5	0.66	27.49	0	14.28	5.01	47.44
7	1	0.67	51.62	0	21.39	5.01	78.69
	2	1.73	51.49	0	21.29	5.01	79.52
	3	0.65	48.83	0	21.55	5.01	76.04
	4	0.65	53.82	0	21.38	5.01	80.86
	5	0.66	50.18	0	22.41	5.01	78.26
8	1	0.74	64.02	0	22.26	5.01	92.03
	2	0.64	57.3	0	22.63	5.01	85.58
	3	0.82	63.07	0	22.5	5.01	91.4
	4	0.72	66.8	0	22.51	5.01	95.04
	5	0.77	66.68	0	22.56	5.01	95.02
10	1	0.67	96.94	0	40.28	5.01	142.9
	2	0.65	95.16	0	40.41	5.01	141.23
	3	0.66	95.89	0	40.21	5.01	141.77
	4	0.66	95.16	0	40.49	5.01	131.32
	5	0.65	89.02	0	40.89	5.01	135.57

A.2 Case B

The extended results for case B also only consist of the frequency of robot ID became the seed robot and the recorded simulation time T_t, T_s, T_a, T_o, T_d and T_b.

A.2.1 Frequency of Robot ID Became the Seed Robot for Case B

Figure A.5, Figure A.6, Figure A.7 and Figure A.8 show the frequency of robot ID became the seed robot for case A for one, two, four and five robot wide void. The number of robots in the swarm simulated for case B is three, six, nine, twelve, fifteen and eighteen swarm size. The robot positions were fixed so the larger IDs that positioned closer to the void had the larger chance to detect the void.

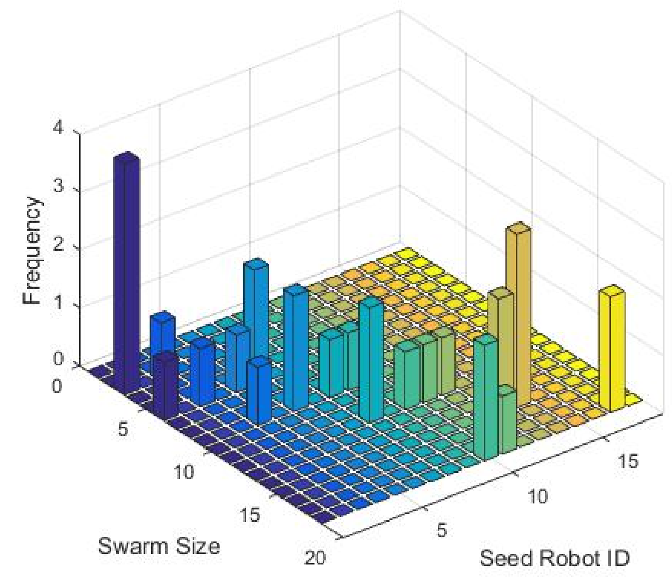


Figure A.5: Frequency with which a specific robot became the seed robot for case B for one robot wide void.

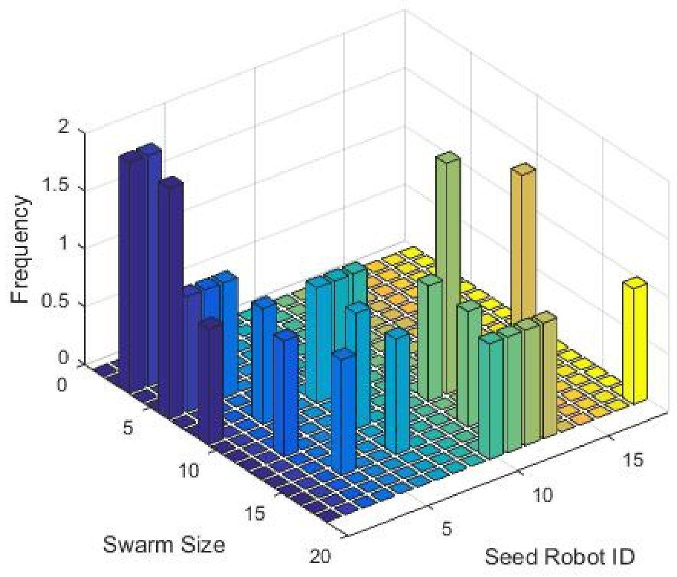


Figure A.6: Frequency with which a specific robot became the seed robot for case B for two robot wide void.

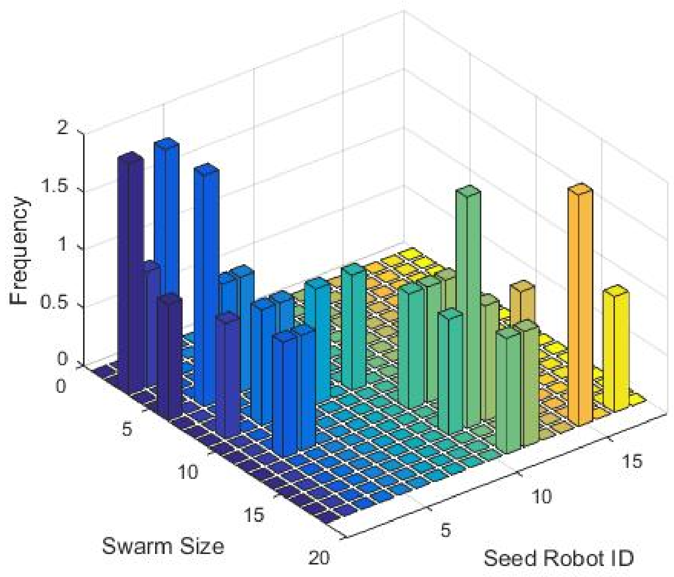


Figure A.7: Frequency with which a specific robot became the seed robot for case B for four robot wide void.

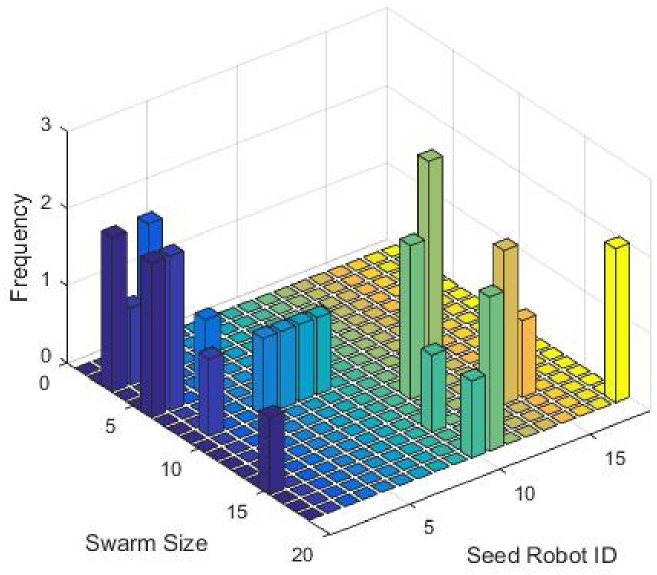


Figure A.8: Frequency with which a specific robot became the seed robot for case B for five robot wide void.

A.2.2 Simulation Time for Case B

Table A.6, Table A.7, Table A.8, Table A.9 and Table A.10 show the recorded time (in s) in simulations including T_s, T_a, T_o. T_d, T_b and T_t (in seconds) for case B, for one, two, three, four and five robot wide void. N is the swarm size and RN is the running number.

Table A.6: Recorded time in a one robot wide void for case B.

N	**RN**	T_s	T_a	T_o	T_d	T_b	T_t
	1	26.55	25.62	61.58	0	0	113.75
	2	16.83	44.86	239.33	0	0	301.02
3	3	22.64	46.35	49.97	0	0	118.96
	4	25.37	36.35	114.47	0	0	176.19
	5	59.13	38.67	69.94	0	0	167.74
	1	30.12	100.95	43.87	0	0	174.94
	2	15.36	93.18	49.12	0	0	157.66
6	3	23.78	89.69	136.99	0	0	250.46
	4	18.1	77.83	27.89	0	0	123.82
	5	53.6	99.37	73.14	0	0	226.11
	1	14.55	150.84	38.75	0	0	204.14
	2	15.42	160.53	11.3	0	0	187.25
9	3	39.99	158.32	118.07	0	0	316.38
	4	22.68	164.26	78.44	0	0	265.38
	5	28.48	139.88	5.6	0	0	173.96
	1	25.2	202.07	33.57	0	0	260.84
	2	13.37	187.23	0.08	0	0	200.68
12	3	39.33	229.26	1.66	0	0	270.25
	4	17.8	207.97	3.38	0	0	229.15
	5	22.36	193.62	62.46	0	0	278.44
	1	14.13	278.99	6.99	0	0	300.11
	2	14.13	278.99	6.99	0	0	300.11
15	3	17.92	271.49	33.74	0	0	323.15
	4	17.41	252.21	2.89	0	0	272.51
	5	14.82	260.77	15.18	0	0	290.77
	1	28.46	286.35	1.43	0	0	316.24
	2	17.3	307.27	99.03	0	0	423.6
18	3	21.07	330.57	56.28	0	0	407.92
	4	14.78	337.29	69.88	0	0	421.95
	5	18.19	314.2	128.56	0	0	460.95

Table A.7: Recorded time in a two robot wide void for case B.

N	**RN**	T_s	T_a	T_o	T_d	T_b	T_t
3	1	36.47	16.44	0	8.73	5.01	66.65
	2	50.04	32.62	0	8.73	5.01	96.4
	3	54.93	28.06	0	8.8	5.01	96.8
	4	53.86	34.4	0	8.73	5.01	102
	5	22.27	31.55	0	8.73	5.01	67.56
6	1	16.79	78.74	0.81	0	0	96.34
	2	19.3	92.15	28.31	0	0	139.76
	3	15.61	81.4	11.36	0	0	108.37
	4	26.75	84.34	5.28	0	0	116.37
	5	18.79	87.36	47.78	0	0	153.93
9	1	27.12	122.7	6.27	0	0	156.09
	2	19.32	129.46	0.79	0	0	149.57
	3	30.03	164.14	25.05	0	0	219.22
	4	21.98	116.73	24.64	0	0	163.35
	5	22.27	160.94	6.45	0	0	189.66
12	1	13.97	210.06	68.41	0	0	292.44
	2	13.91	205.97	80.3	0	0	300.18
	3	17.48	195.73	104.35	0	0	317.56
	4	13.27	194.09	28.2	0	0	235.56
	5	26.86	191.93	22.48	0	0	241.27
15	1	19.28	245.25	49.05	0	0	313.58
	2	13.75	261.71	8.38	0	0	283.84
	3	16.56	254.87	30.76	0	0	302.19
	4	12.84	242.2	24.9	0	0	279.94
	5	12.73	246.49	41.19	0	0	300.41
18	1	13.95	298.71	119.3	0	0	431.96
	2	13.07	299.88	0.01	0	0	312.96
	3	14.77	326.99	5.79	0	0	347.55
	4	14.63	317.46	15.8	0	0	347.89
	5	13.79	311.07	36.75	0	0	361.61

Table A.8: Recorded time in a three robot wide void for case B.

N	**RN**	T_s	T_a	T_o	T_d	T_b	T_t
3	1	58.34	29.17	0	8.74	5.01	101.26
	2	15.39	25.61	0	8.8	5.01	54.81
	3	28.8	29.4	0	8.8	5.01	72.01
	4	14.7	29.64	0	8.66	5.01	58.01
	5	29	46.46	0	8.8	5.01	89.27
6	1	18.97	94.6	0	21.73	5.01	140.31
	2	25	68.03	0	21.87	5.01	119.91
	3	22.19	67.68	0	21.86	5.01	116.74
	4	14.8	59.77	0	22	5.01	101.58
	5	27.18	53.32	0	21.79	5.01	107.3
9	1	25.3	106.41	14.24	0	0	145.95
	2	21.11	155.52	5.15	0	0	181.78
	3	17.13	152.18	59.78	0	0	229.09
	4	22.42	148.05	92.6	0	0	263.07
	5	22.4	121.91	2.79	0	0	147.1
12	1	11.84	170.6	24.34	0	0	206.78
	2	22.64	181.98	1.05	0	0	205.67
	3	11.86	176.29	25.23	0	0	213.38
	4	13.32	180.78	51.72	0	0	245.82
	5	41.34	167.62	100.58	0	0	309.54
15	1	14.71	265.12	77.18	0	0	357.01
	2	13.84	211.66	2.69	0	0	228.19
	3	17.2	271.22	14.08	0	0	302.5
	4	13.25	249.96	132.38	0	0	395.59
	5	15.22	244.94	31.36	0	0	291.52
18	1	13.06	293.14	0.25	0	0	306.45
	2	20.56	288.76	2.7	0	0	312.02
	3	14.35	273.77	18.42	0	0	306.54
	4	13.46	333.14	44.89	0	0	391.49
	5	12.02	295.51	16.86	0	0	324.39

Table A.9: Recorded time in a four robot wide void for case B.

N	**RN**	T_s	T_a	T_o	T_d	T_b	T_t
3	1	29	24.64	0	8.73	5.01	67.38
	2	25.29	19.46	0	8.73	5.01	58.49
	3	41.75	22.35	0	8.73	5.01	77.84
	4	85.02	16.85	0	8.66	5.01	115.54
	5	29.25	31.96	0	8.8	5.01	75.02
6	1	29.66	70.89	0	21.79	5.01	127.35
	2	19.6	72.3	0	21.7	5.01	118.61
	3	47.69	73.41	0	21.86	5.01	147.97
	4	16.72	86.97	0	21.72	5.01	130.42
	5	14.55	80.22	0	21.93	5.01	121.71
9	1	35.33	88.54	0	34.92	5.01	163.8
	2	14.37	119.25	0	35.11	5.01	173.74
	3	26.18	111.16	0	34.85	5.01	177.2
	4	14.97	103.7	0	34.87	5.01	158.55
	5	16.12	104.79	0	34.98	5.01	160.9
12	1	15.77	214.84	32.71	0	0	263.32
	2	23.83	166.72	22.3	0	0	212.85
	3	24.85	186.79	29.71	0	0	241.35
	4	11.69	173.93	114.59	0	0	300.21
	5	18.69	195.85	24.83	0	0	239.37
15	1	17.01	278.8	38.12	0	0	333.93
	2	12.87	219.16	0.07	0	0	232.1
	3	16.22	228.44	2.28	0	0	246.94
	4	12.98	244.44	3.7	0	0	261.12
	5	24.23	243.67	16.11	0	0	284.01
18	1	11.98	321.61	25.85	0	0	359.44
	2	17.51	251.54	43.49	0	0	312.54
	3	15.03	285.76	93.28	0	0	394.07
	4	12.68	276.64	16.63	0	0	305.95
	5	12.6	248.49	41.75	0	0	302.84

Table A.10: Recorded time in a five robot wide void for case B.

N	**RN**	T_s	T_a	T_o	T_d	T_b	T_t
3	1	24.68	19.6	0	8.73	5.01	58.02
	2	15.01	23.61	0	8.74	5.01	52.37
	3	24.79	19.62	0	8.8	5.01	58.22
	4	19.23	17.18	0	8.66	5.01	50.08
	5	44.52	22.57	0	8.73	5.01	80.83
6	1	25.36	86.92	0	21.79	5.01	139.08
	2	19.51	81.15	0	21.65	5.01	127.32
	3	17.53	68.48	0	21.65	5.01	112.67
	4	14.96	78.89	0	21.96	5.01	120.82
	5	19.04	76.5	0	21.73	5.01	122.28
9	1	16.28	109.61	0	34.85	5.01	165.75
	2	13.17	91.33	0	34.78	5.01	144.29
	3	18.41	119.15	0	35	5.01	177.57
	4	16.79	103.96	0	34.91	5.01	160.67
	5	26.85	123.53	0	35.13	5.01	190.52
12	1	20.58	141.18	0	47.82	5.01	214.59
	2	19.84	139.21	0	48.03	5.01	212.09
	3	15.28	178.39	0	47.7	5.01	246.38
	4	17.83	131.79	0	47.98	5.01	202.61
	5	28.26	159.36	0	48.21	5.01	240.84
15	1	17.04	222.96	35.53	0	0	275.53
	2	22.55	228.66	58.88	0	0	310.09
	3	12.59	244.23	8.17	0	0	264.99
	4	12.59	244.23	8.17	0	0	264.99
	5	11.97	225.1	78.54	0	0	315.61
18	1	12.24	270.65	5.84	0	0	288.73
	2	19.94	291.59	2.29	0	0	313.82
	3	14.19	293.41	36.91	0	0	344.51
	4	22.2	273.56	234.48	0	0	530.24
	5	12.24	270.65	5.84	0	0	288.73

A.3 Case C

This section shows results that have been collected for case C, consisting of frequency of robot ID became the seed robot, seed robots positions, frequency of robot detected the target, simulation snapshots, and the simulation time including the exploration time including at the initial and target zone, the assembly and moving forward time, and the retreat and disassembly time.

A.3.1 Frequency of Robot ID Became the Seed Robot for Case C

Figure A.9, Figure A.10, Figure A.11 and Figure A.12 show the frequency of a specific robot under fixed initial positions to become the seed robot for one, two, four and five robot wide void.

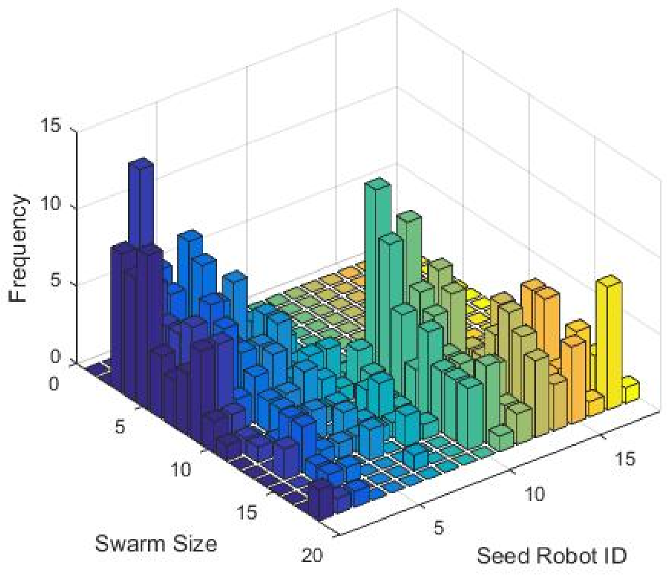


Figure A.9: Frequency of seed ID in fixed initial positions for one robot wide void.

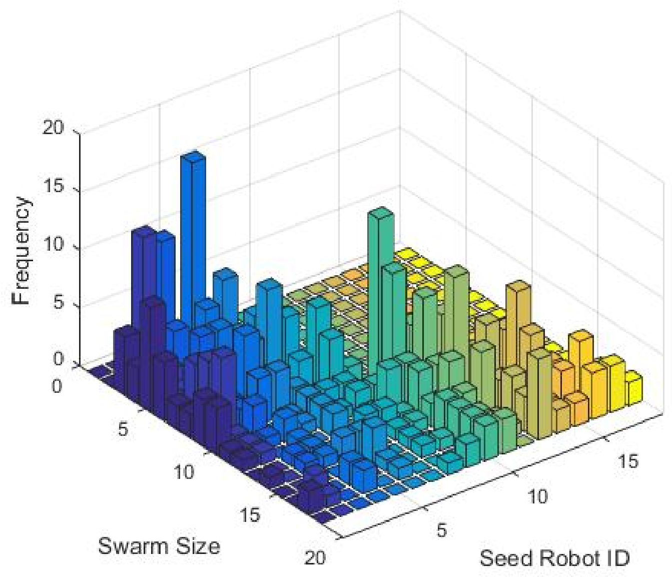


Figure A.10: Frequency of seed ID in fixed initial positions for two robot wide void.

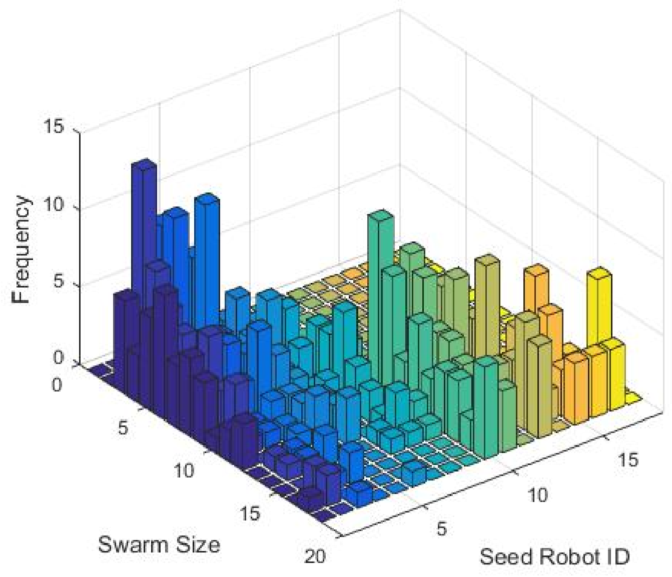


Figure A.11: Frequency of seed ID in fixed initial positions for four robot wide void.

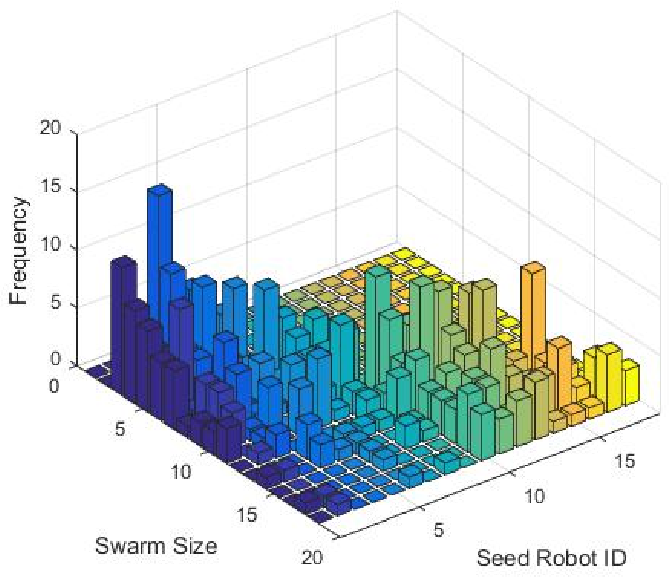


Figure A.12: Frequency of seed ID in fixed initial positions for five robot wide void.

A.3.2 Positions of Seed Robots for Case C: Fixed Initial Positions

Figure A.13, Figure A.14 and Figure A.15 show the positions of seed robots for case C for three to six, eleven to fourteen and fifteen to eighteen robots in the swarm. x-coordinates 220, 225, 230, 235 and 240 show the void size of five, four, two and one robot wide, respectively.

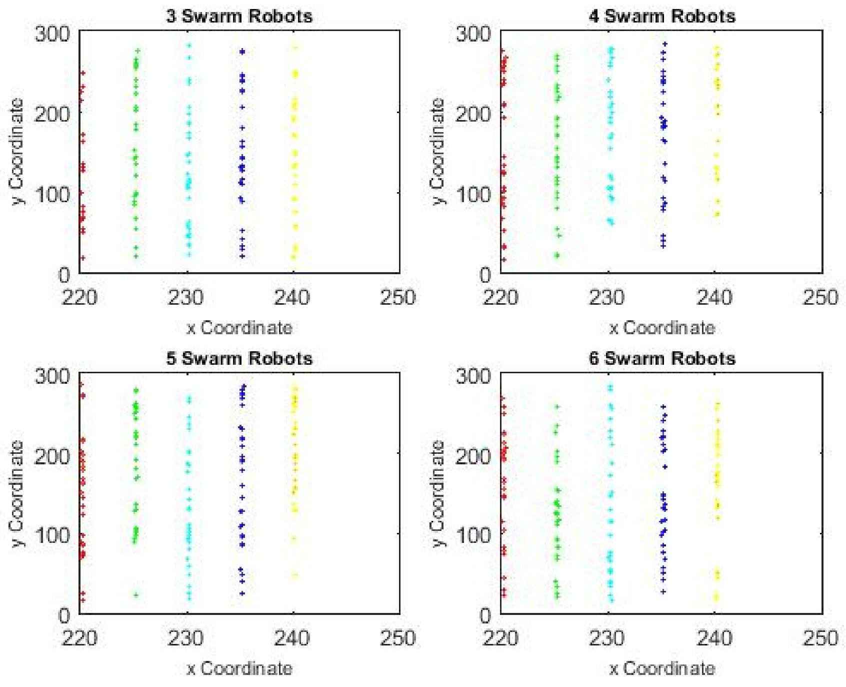


Figure A.13: Positions of seed robots in fixed initial positions for three to six robots.

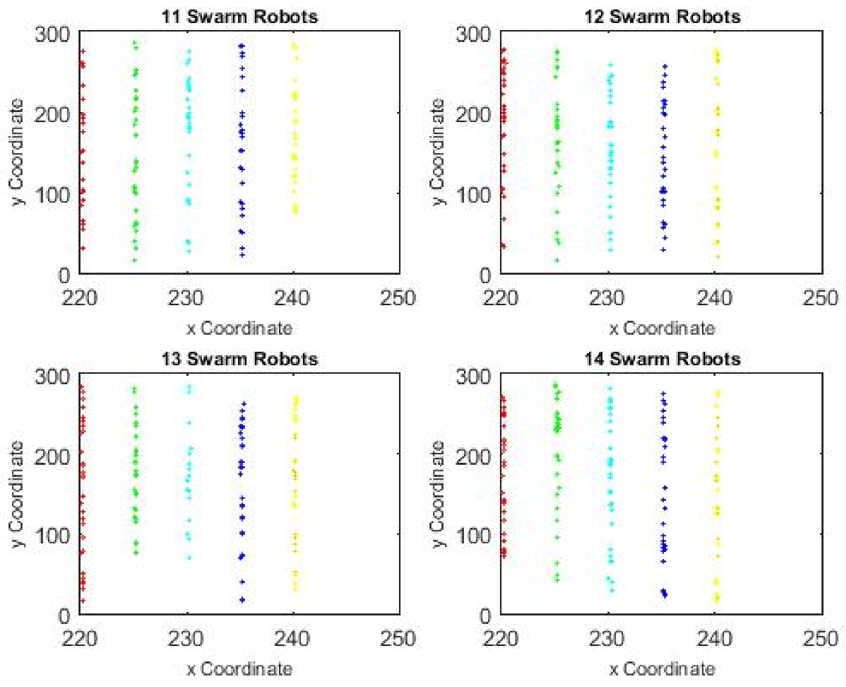


Figure A.14: Positions of seed robots in fixed initial positions for eleven to fourteen robots.

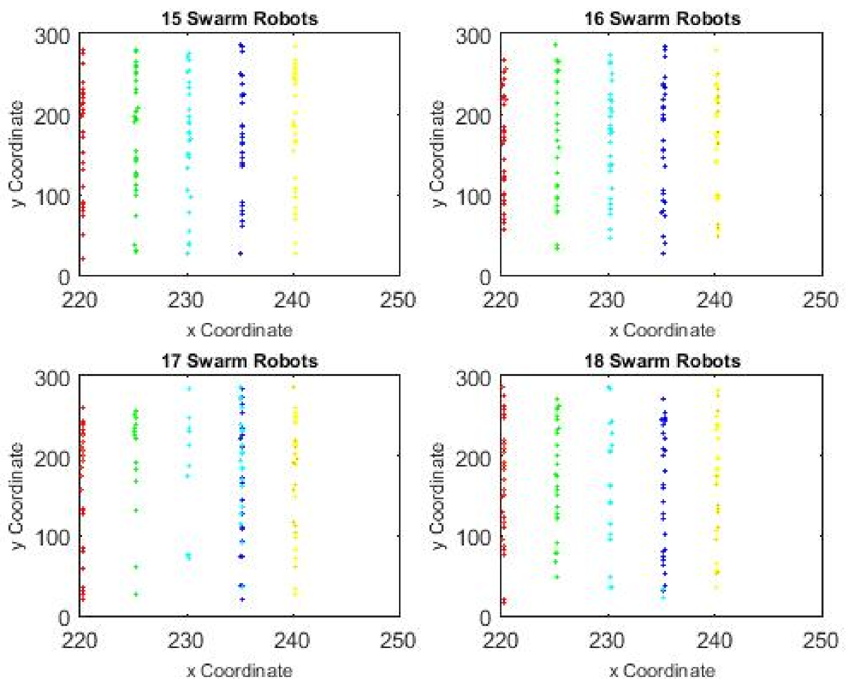


Figure A.15: Positions of seed robots in fixed initial positions for fifteen to eighteen robots.

A.3.3 Frequency of Robot ID Detecting the Target for Case C

Figure A.16, Figure A.17, Figure A.18 and Figure A.19 show the frequency of robot ID that detected the target where robot initial positions were fixed, for one, two, four and five robot wide void.

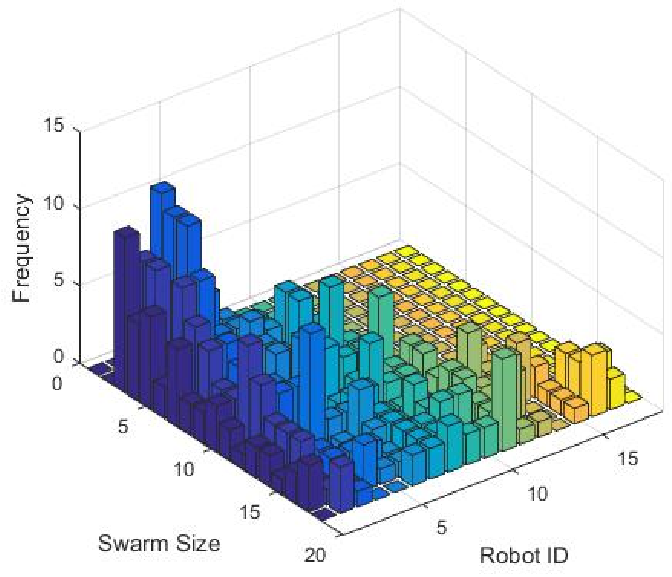


Figure A.16: Frequency of robot ID detecting the target in fixed initial positions for one robot wide void.

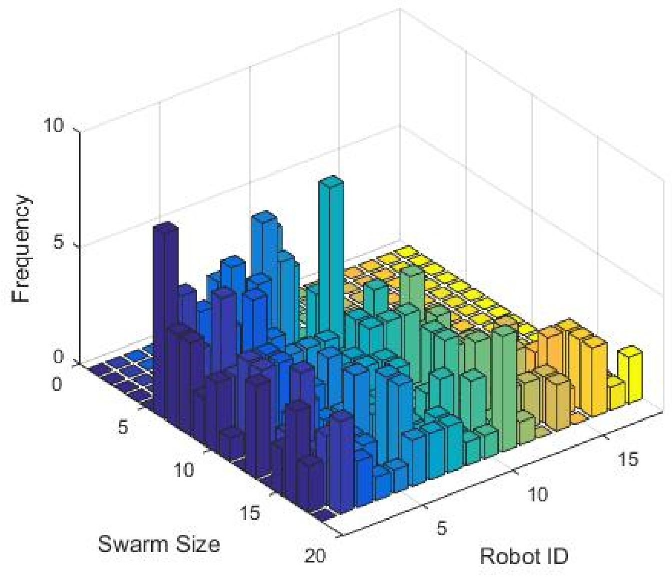


Figure A.17: Frequency of robot ID detecting the target in fixed initial positions for two robot wide void.

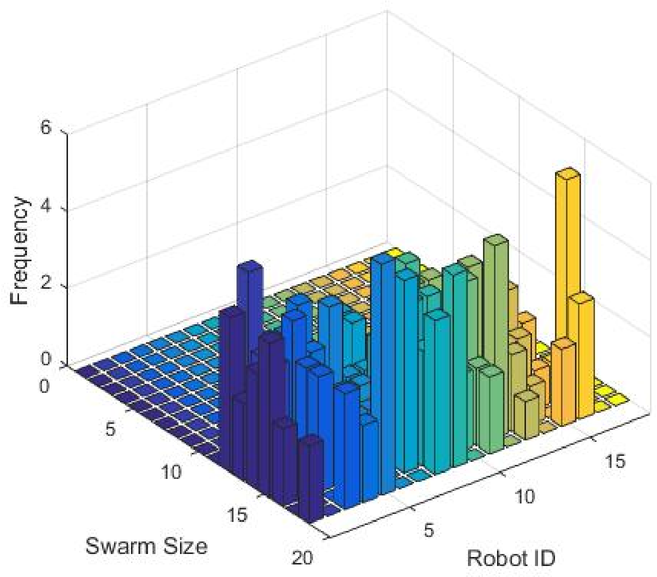


Figure A.18: Frequency of robot ID detecting the target in fixed initial positions for four robot wide void.

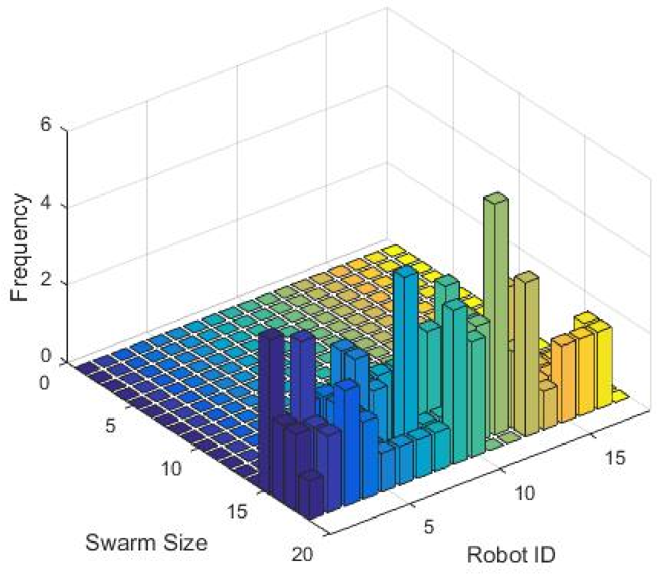


Figure A.19: Frequency of robot ID detecting the target in fixed initial positions for five robot wide void.

A.3.4 Snapshots of Simulations for Case C

Figure A.20, Figure A.22 and Figure A.24 show the simulations when the swarm failed to cross the void for case C, for two, three and four robot void. And Figure A.21, Figure A.23 and Figure A.25 show the simulations of successful task.

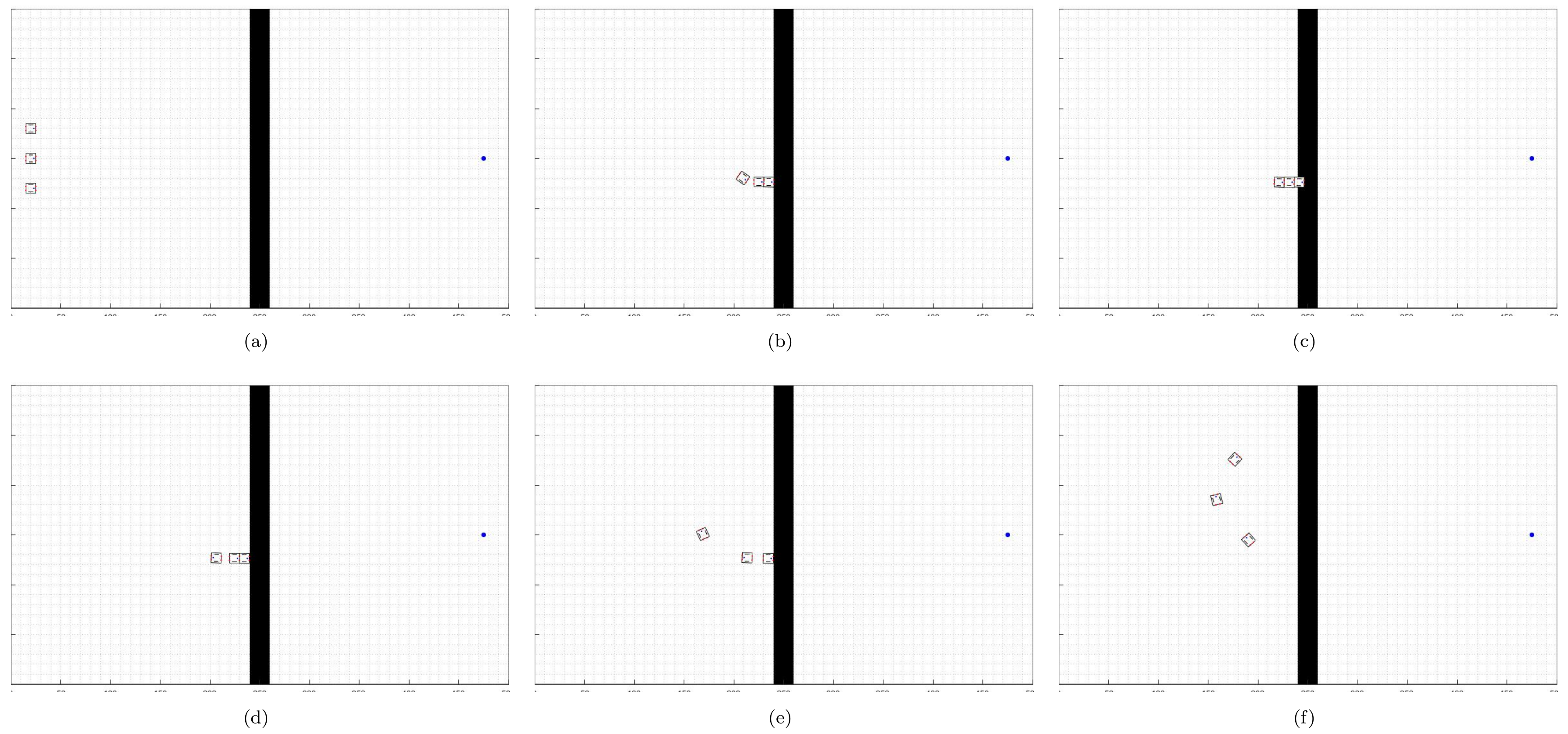


Figure A.20: Three robots failed to cross the two robot wide void in fixed initial position. (a) Three robots are ready to wander the initial zone to find the void. (b) The last robot is joining the structure. (c) The structure moved towards the opposite zone, but it retreats to the initial zone because it is not long enough to reach the opposite ground. (d) The rear robot in the structure decouples from the structure and wander the initial zone. (e) All robots have disassembled. (f) After exploring the initial zone for a while, they stop moving.

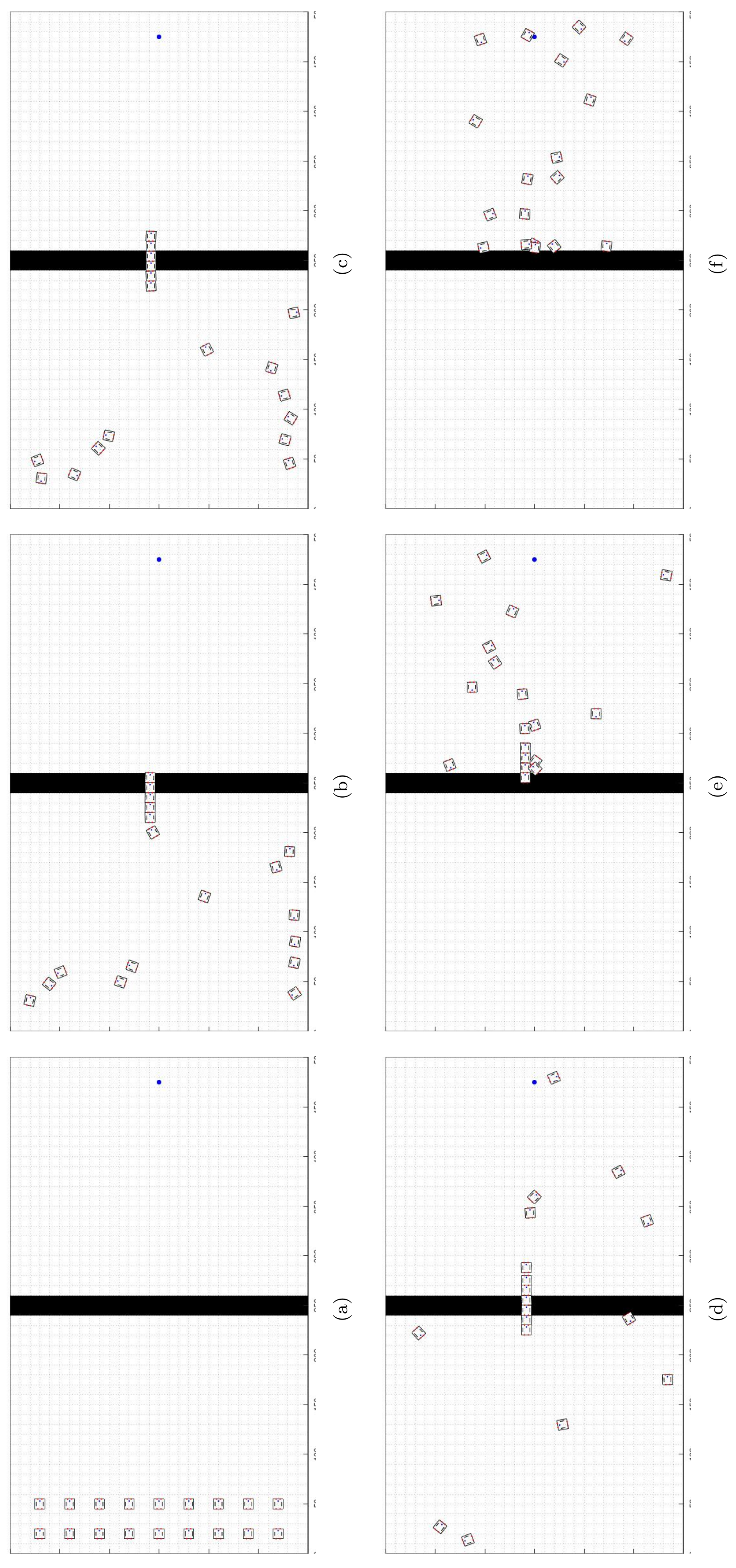
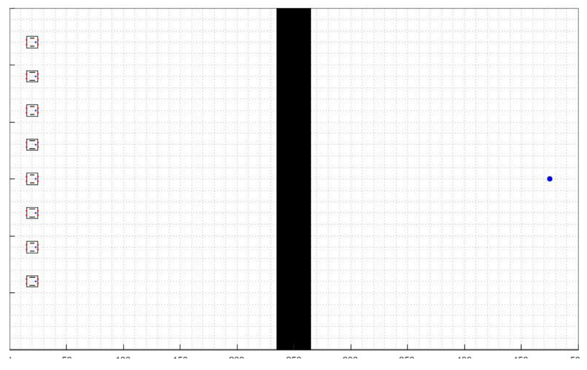
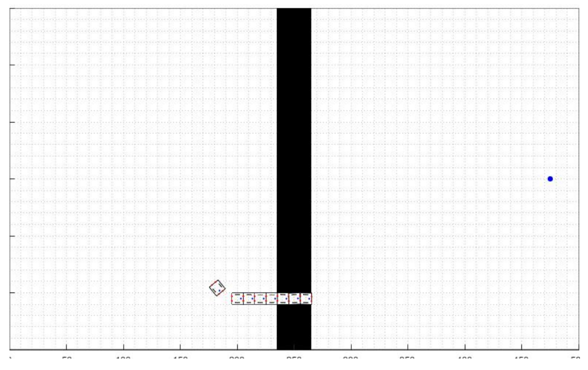
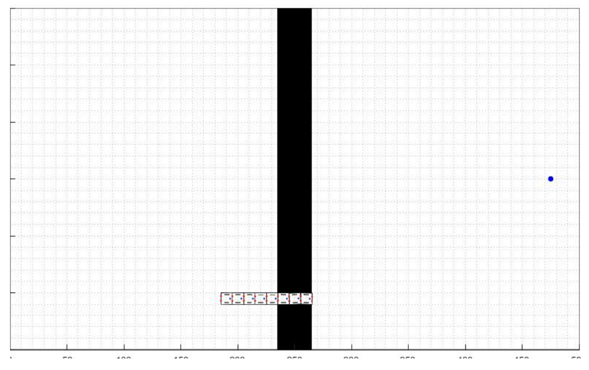
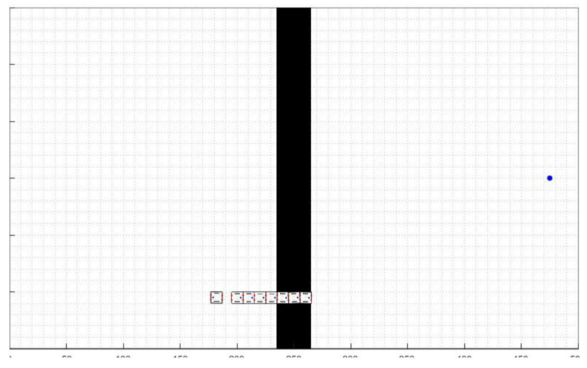


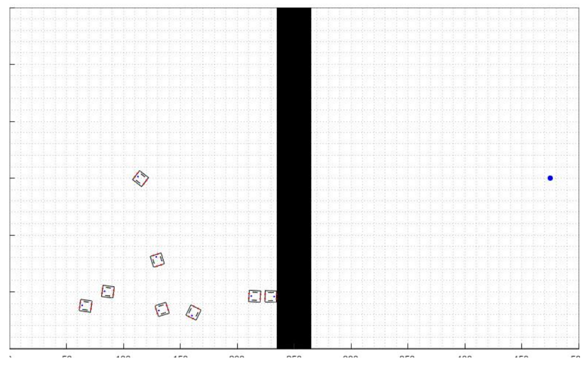
Figure A.21: Eighteen robots are success to cross the two robot wide void in fixed initial position. (a) The initial position of the swarm to explore the arena. (b) A robot is joining the structure while the others wander at the initial zone. (c) When the structural stability has been maintained over the void, it is checking if there is free robot at the initial zone to be recruited. (d) Six robots in the structure is maintained to ensure the structural stability when the front robot disassembles from the structure. (e) When there is no free robot at the initial zone, the structure moves forwards and the front robot disassembles while the structural stability is kept maintained, decoupled robots wander at the opposite zone. (f) A robot has found the target.


(a)


(b)


(c)


(d)


(e)

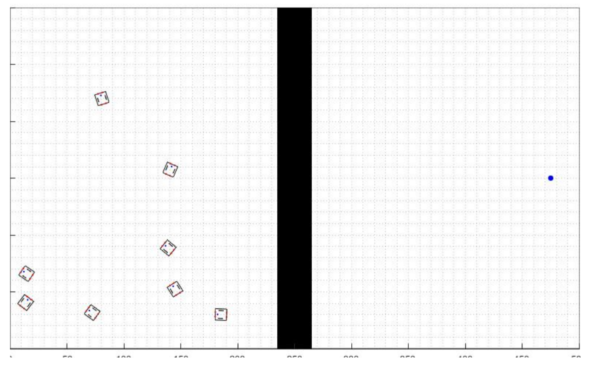

(f)

Figure A.22: Eight robots failed to cross the three robot wide void in fixed initial position. (a) A swarm consists of eight robots is ready to explore and find the void. (b) The last free robot is joining the structure. (c) The front robot is checking if there is the opposite ground and comparing the robot number in the structure to see the structural stability. (d) Because the number of robot is not enough to maintain the structural stability, the rear robot decouples and then the structure will retreat. (e) The last two robots that are still in the structure are decoupling while other robots explore the initial area again. (f) After exploring for a while, they stop moving.

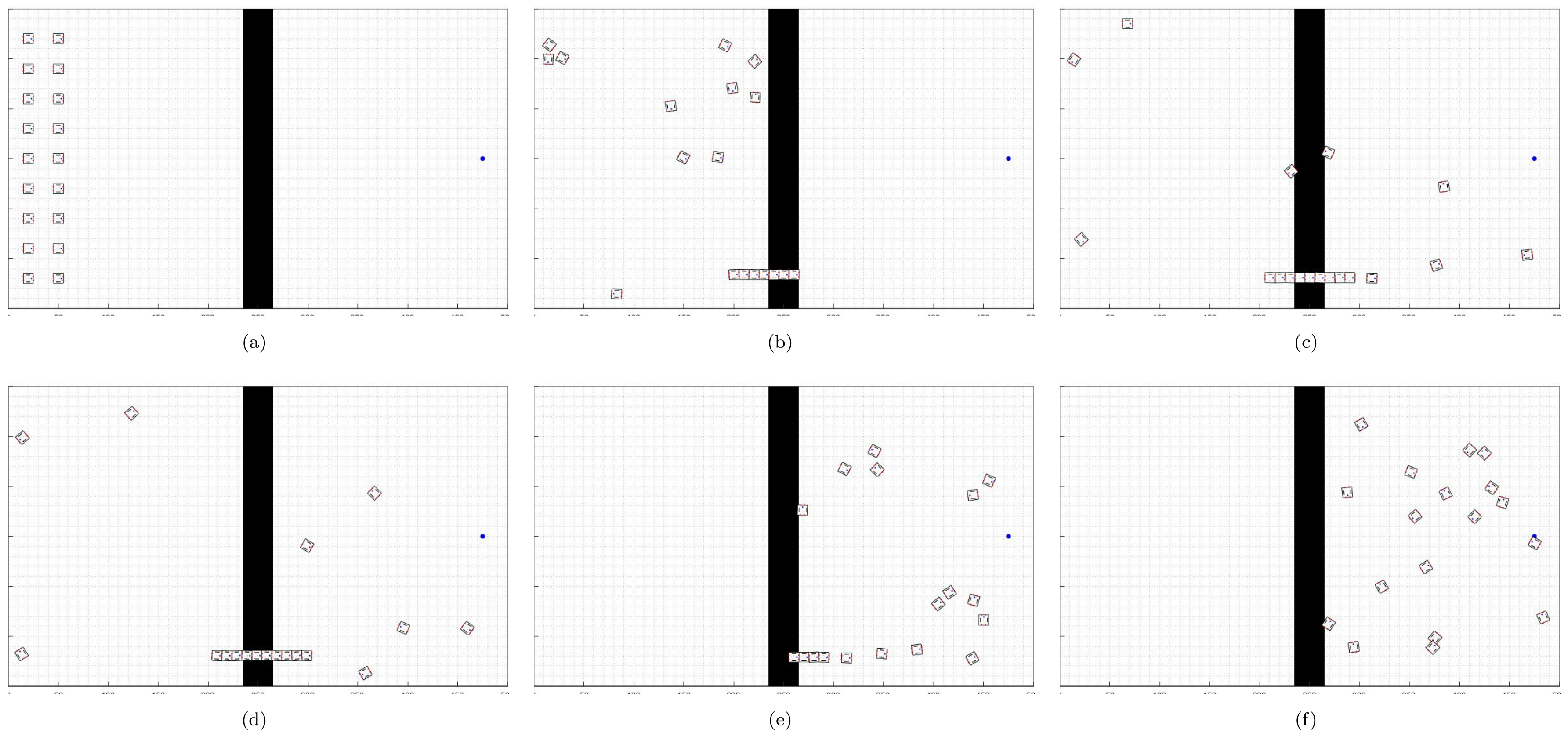


Figure A.23: Eighteen robots are success to cross the three robot wide void in fixed initial position. (a) The swarm in its initial position. (b) The structure is recruiting free robots at the initial area while they are wandering on until the number is satisfying to maintain the structural stability. (c) Nine robots are maintained over the void for the structural stability when the front robot disassembles. (d) Disassembled robots explore the opposite ground while the structure is recruiting free robots at the initial zone. (e) When there is no free robot, the structure moves forward and the front robot decouples. (f) The target has been detected by a robot.

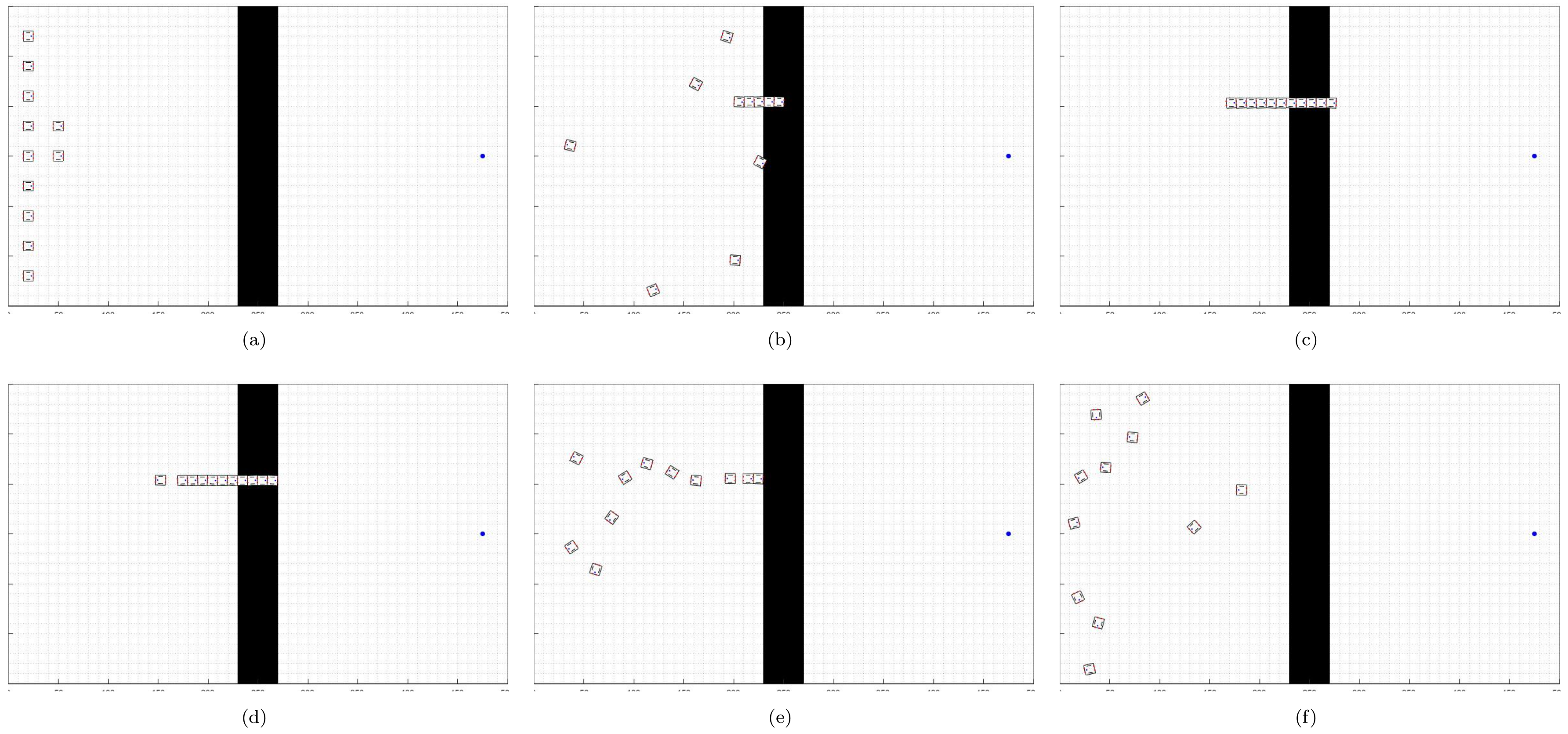


Figure A.24: Eleven robots failed to cross the four robot wide void in fixed initial position. (a) The first nine robots are positioned closer to the edge of the arena and two others are closer to the void. (b) Five robots have assembled in the structure and they are recruiting other free robots to join while moving forward to check if there is the opposite ground. (c) The front robot has reached the opposite ground, but the number of robots in the structure is not enough to maintain the structural stability. (d) The structure retreats and the rear robot disassembles from the structure to wander back at the initial zone. (e) The structure continues to retreat and other rear robots disassemble. (f) All robots stop moving after they explore the initial area again for a while.

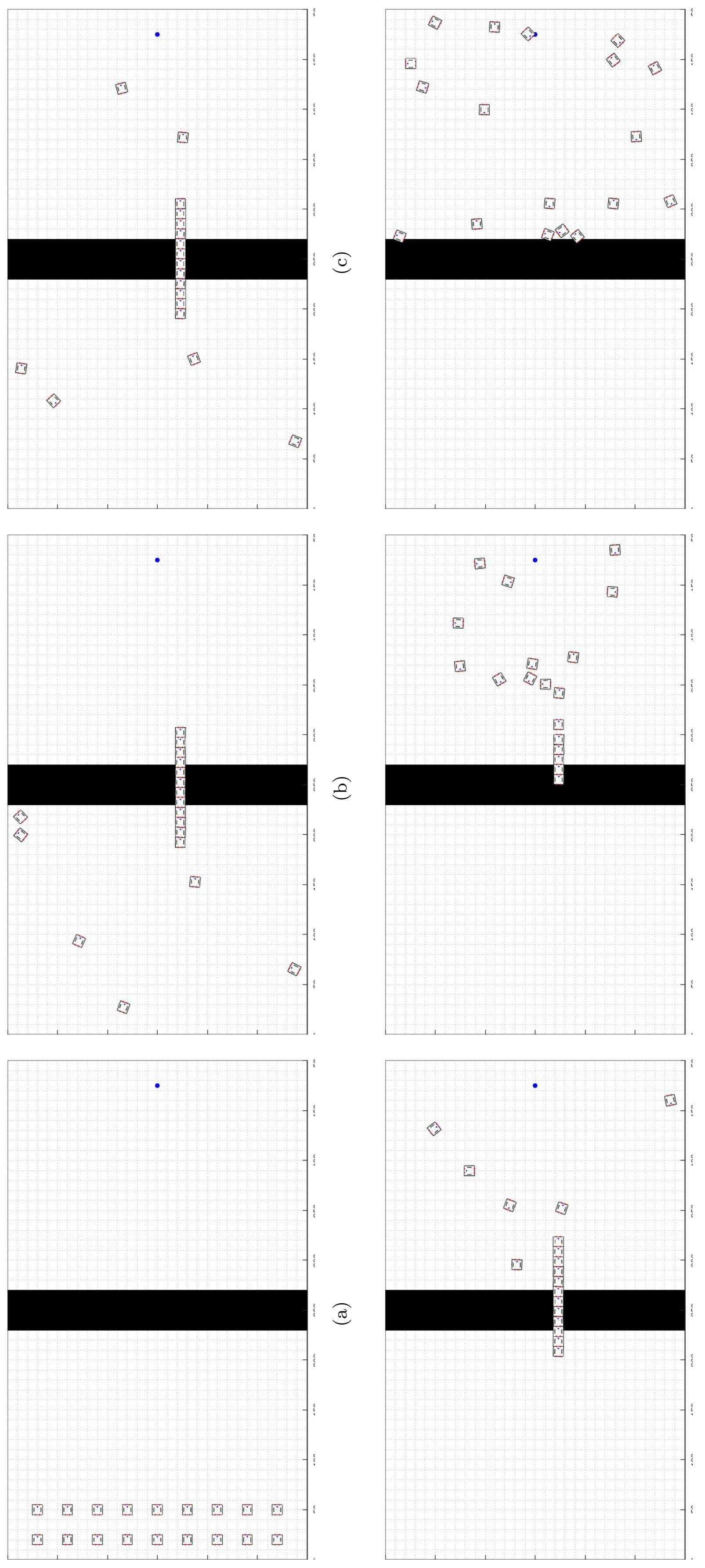


Figure A.25: Eighteen robots are success to cross the four robot wide void in fixed initial position. (a) The swarm at its initial position. (b) The front robot has detected the opposite ground and the recruited robots in the structure is enough to cross the void so that it can move forward. (c) Twelve robots are kept in the structure to maintain the stability by recruiting other free robots while the front robot disassembles from the structure. (d) When there is no free robot at the initial zone, the structure starts moving forward to the opposite ground and the front robot decouples. (e) Decoupled robots wander at the target area while the remain robots in the structure move forward and disassemble. (f) A robot has found the target.

A.3.5 Recorded Time of Simulations for Case C

The following figures show the recorded time of simulations for case C, including T_t, T_s, T_a, T_d and T_o.

A.3.5.1 The Simulation Time (T_t) for Case C

Figure A.26, Figure A.27, Figure A.28 and Figure A.29 show the simulation time for case C for one, two, four and five robot wide void.

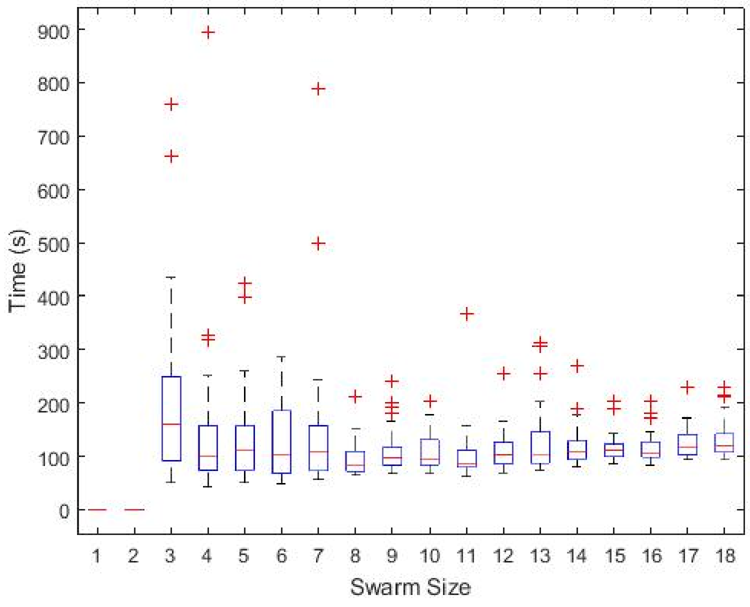


Figure A.26: The simulation time of fixed initial position for one robot wide void.

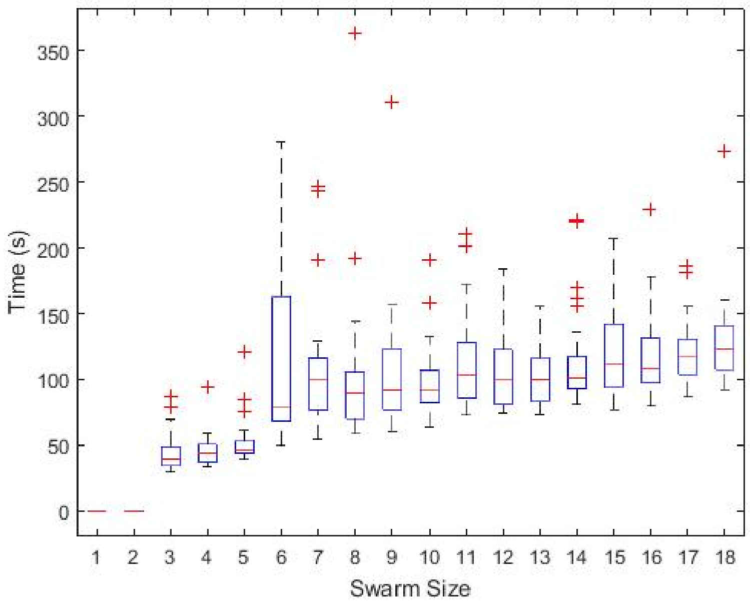


Figure A.27: The simulation time of fixed initial position for two robot wide void.

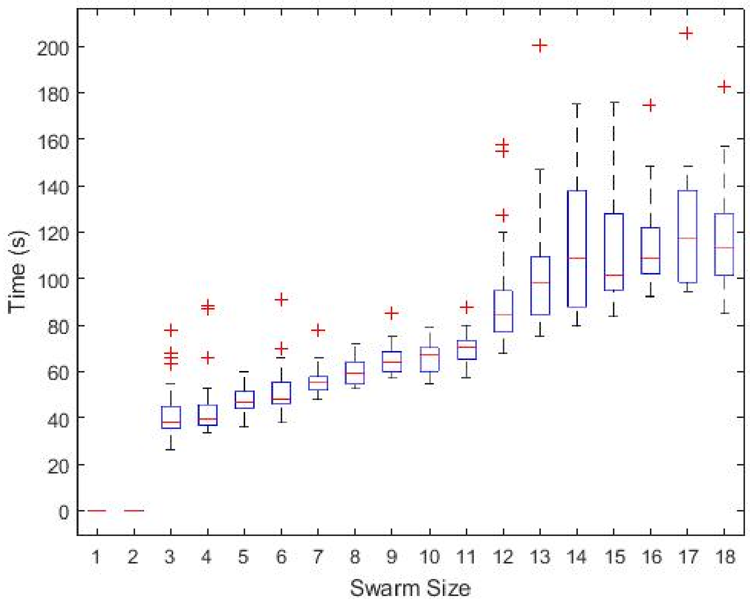


Figure A.28: The simulation time of fixed initial position for four robot wide void.

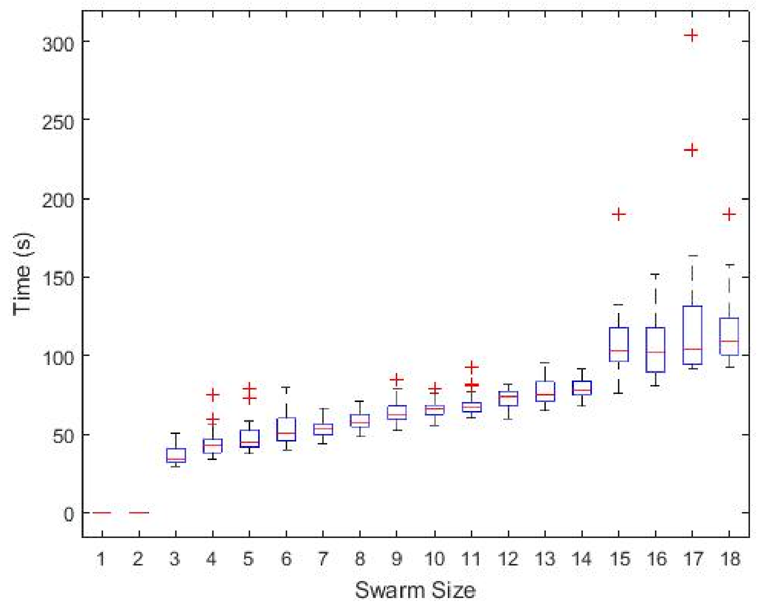


Figure A.29: The simulation time of fixed initial position for five robot wide void.

A.3.5.2 The Exploration Time at the Initial Zone (T_s) for Case C

Figure A.30, Figure A.31, Figure A.32 and Figure A.33 show the exploration time at the initial zone for case C, for one, two, four and five robot wide void.

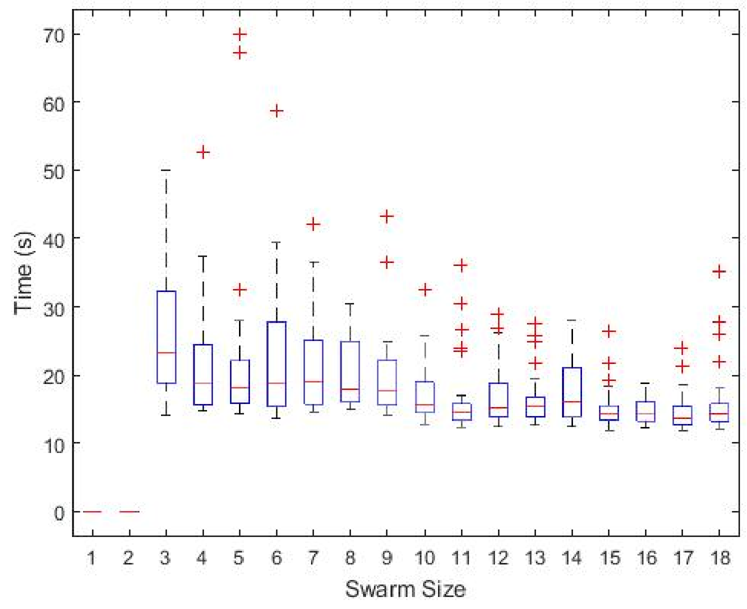


Figure A.30: The exploration time at the initial zone in fixed initial positions for one robot wide void.

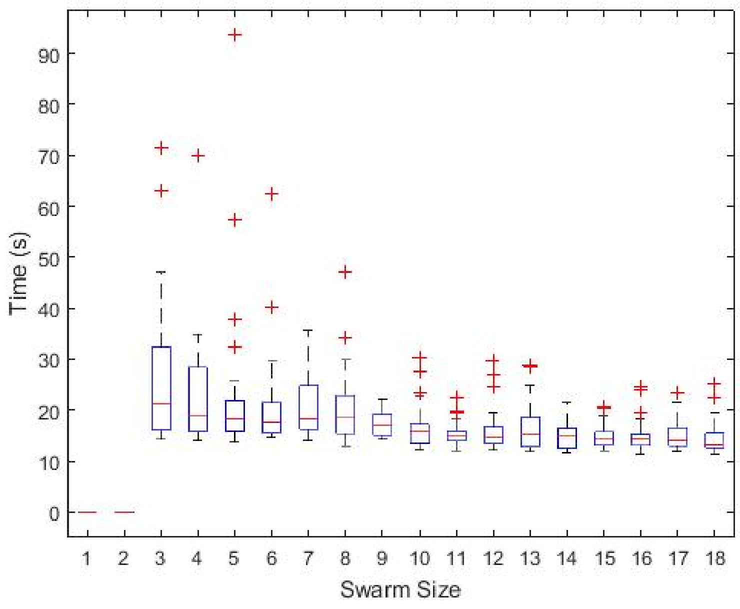


Figure A.31: The exploration time at the initial zone in fixed initial positions for two robot wide void.

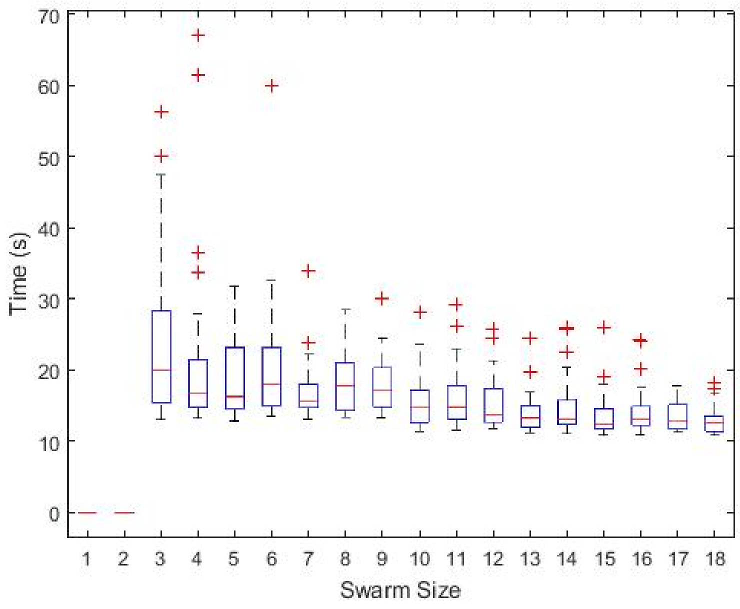


Figure A.32: The exploration time at the initial zone in fixed initial positions for four robot wide void.

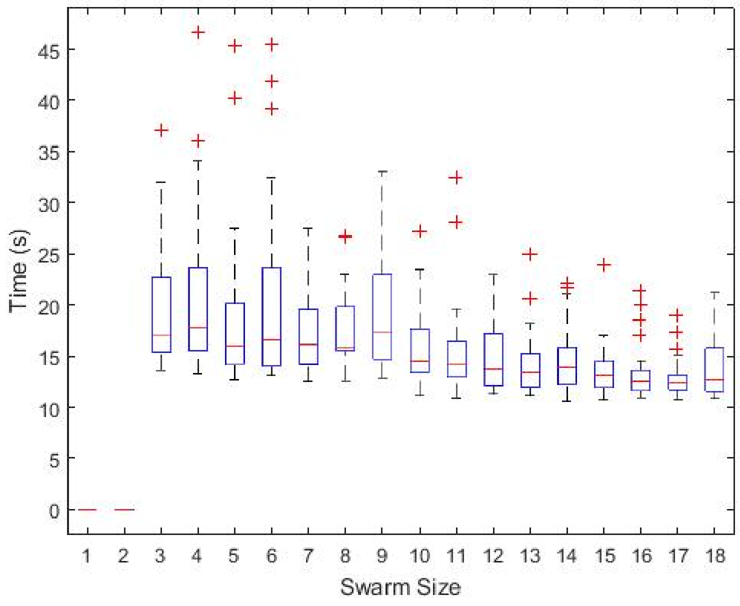


Figure A.33: The exploration time at the initial zone in fixed initial positions for five robot wide void.

A.3.5.3 The Assembly and Moving Forward Time (T_a) for Case C

Figure A.34, Figure A.35, Figure A.36 and Figure A.37 show the assembly and moving forward time for case C, for one, two, four and five robot wide void.

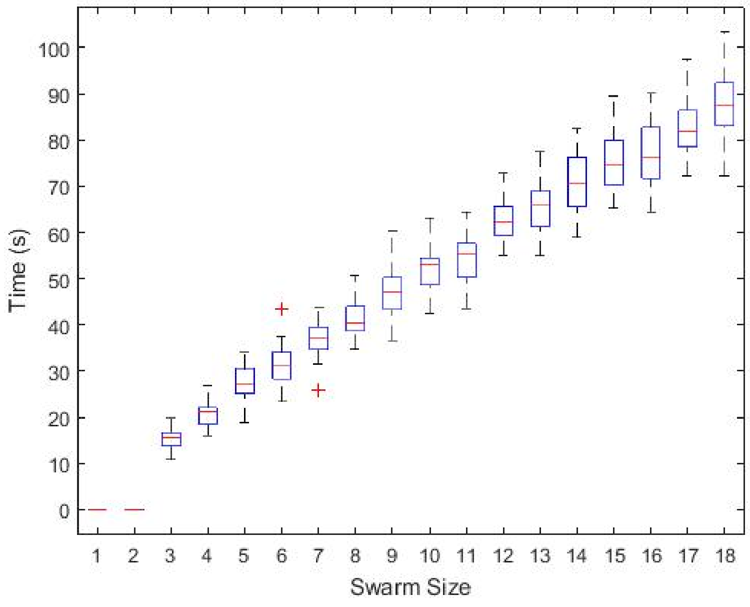


Figure A.34: The assembly and moving forward time in fixed initial positions for one robot wide void.

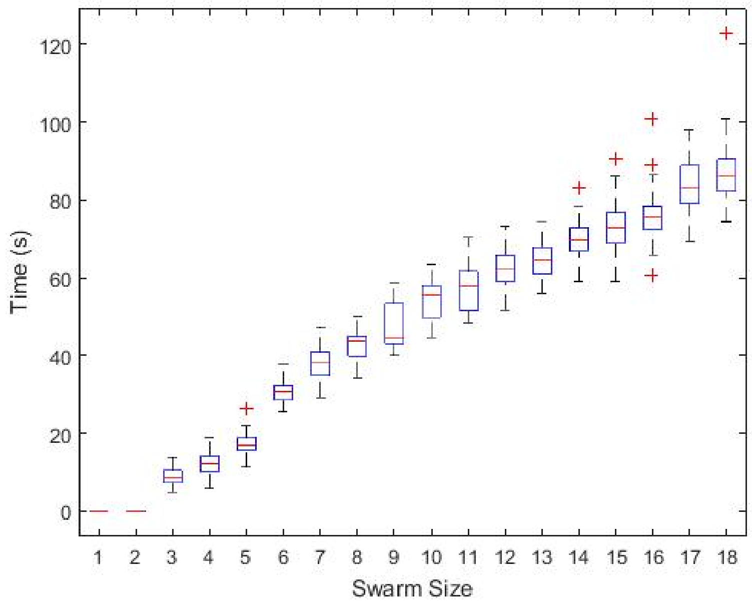


Figure A.35: The assembly and moving forward time in fixed initial positions for two robot wide void.

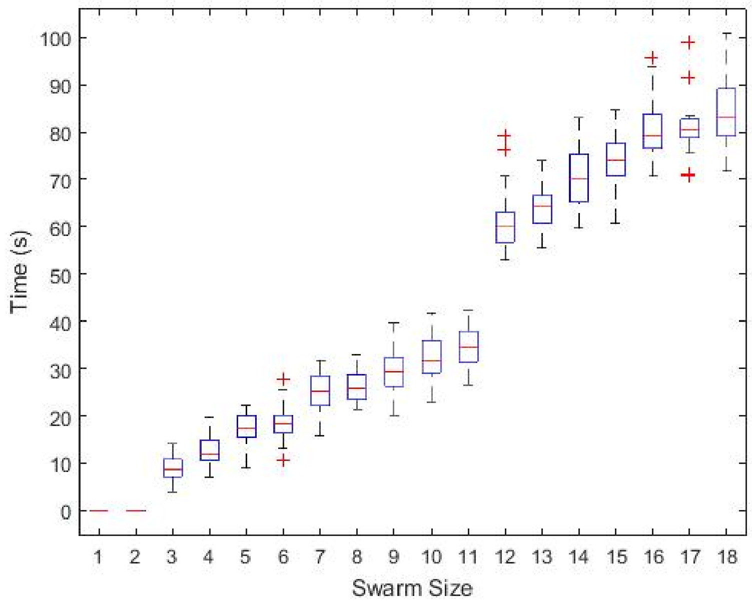


Figure A.36: The assembly and moving forward time in fixed initial positions for four robot wide void.

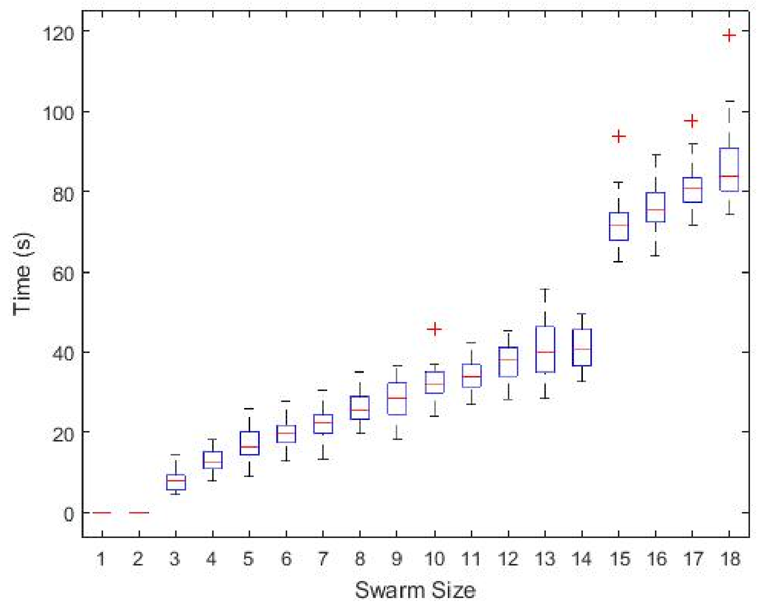


Figure A.37: The assembly and moving forward time in fixed initial positions for five robot wide void.

A.3.5.4 The Retreat and Disasembly Time (T_d) for Case C

Figure A.38, Figure A.39, Figure A.40 and Figure A.41 show the retreat and disassembly time for case C, for one, two, four and five robot wide void.

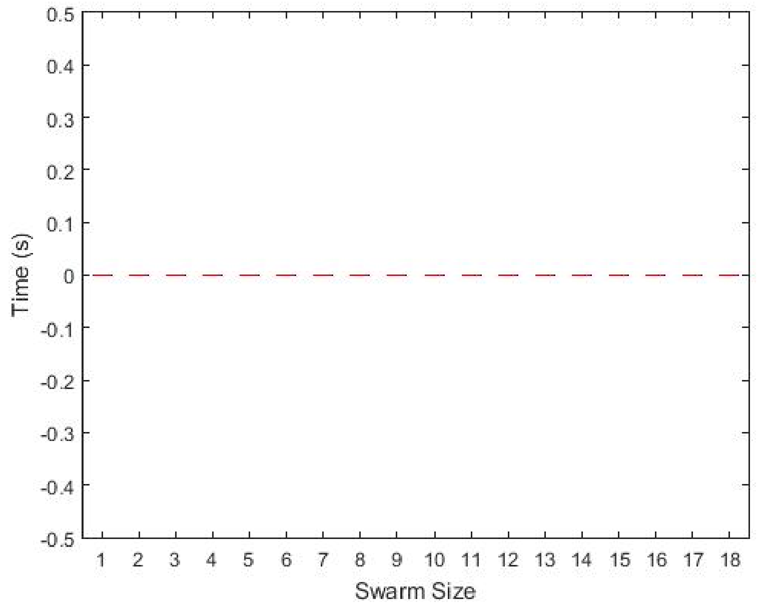


Figure A.38: The retreat and disassembly time in fixed initial positions for one robot wide void.

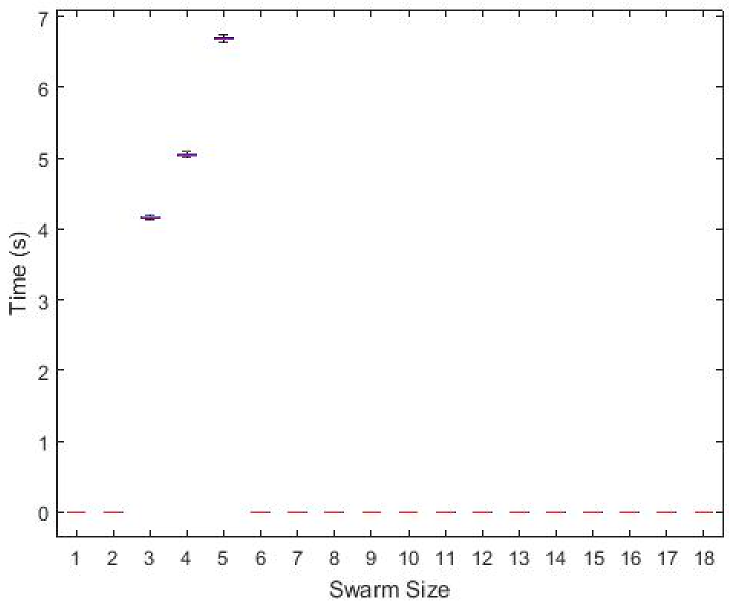


Figure A.39: The retreat and disassembly time in fixed initial positions for two robot wide void.

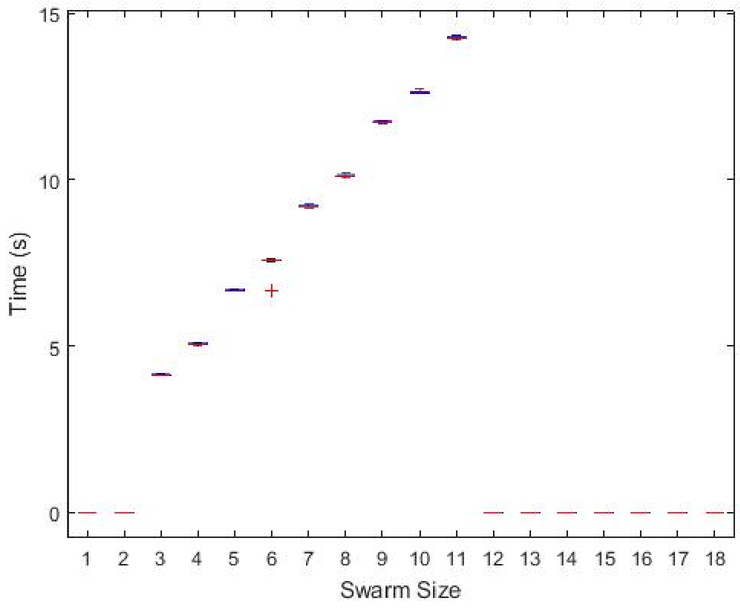


Figure A.40: The retreat and disassembly time in fixed initial positions for four robot wide void.

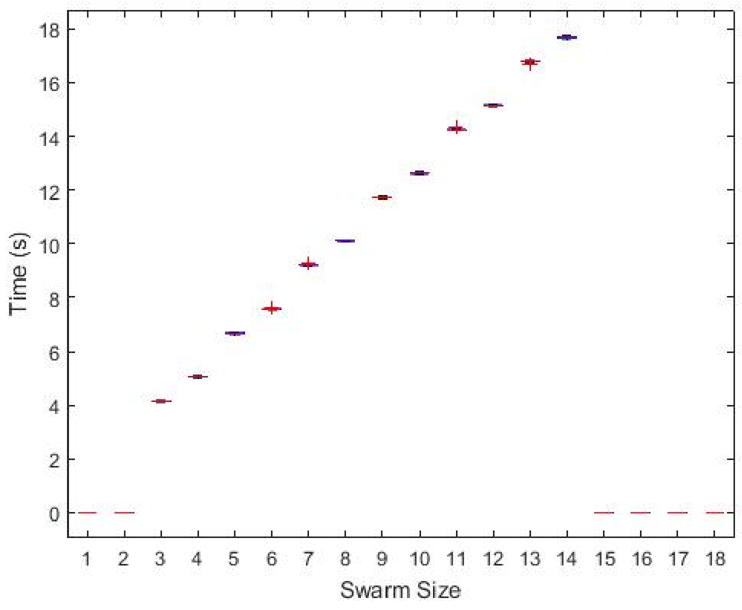


Figure A.41: The retreat and disassembly time in fixed initial positions for five robot wide void.

A.3.5.5 The Exploration Time at the Target Zone (T_o) for Case C

Figure A.42, Figure A.43, Figure A.44 and Figure A.45 show the exploration time at the target zone for case C, for one, two, four and five robot wide void.

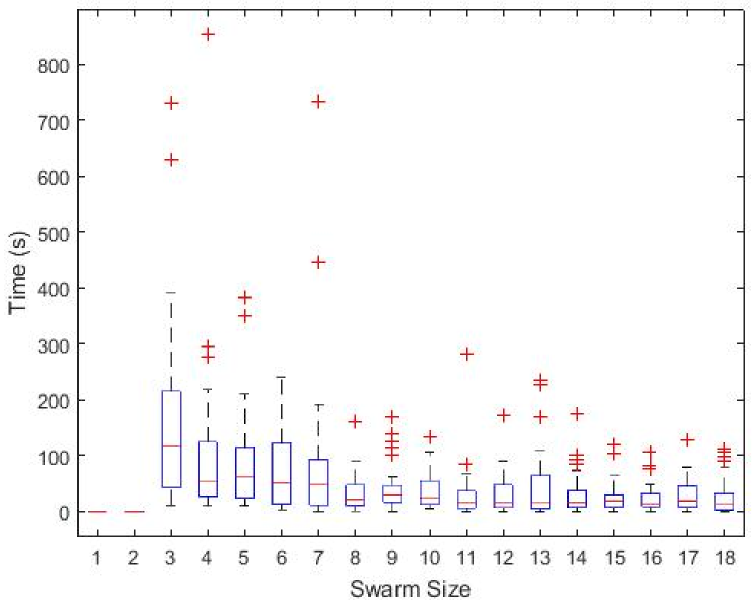


Figure A.42: The exploration time at the target zone in fixed initial positions for one robot wide void.

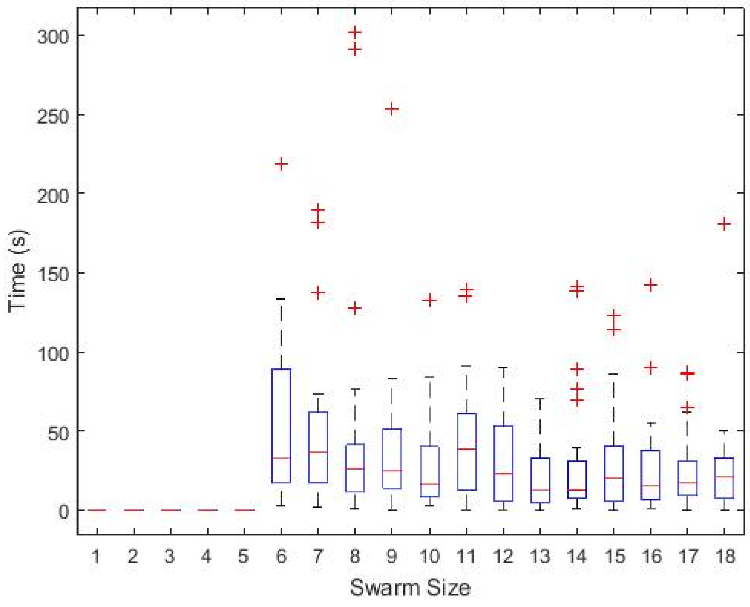


Figure A.43: The exploration time at the target zone in fixed initial positions for two robot wide void.

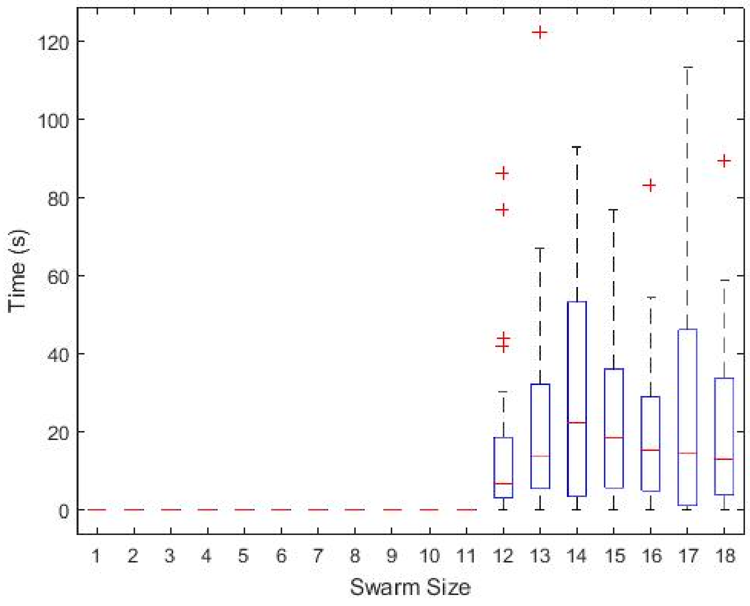


Figure A.44: The exploration time at the target zone in fixed initial positions for four robot wide void.

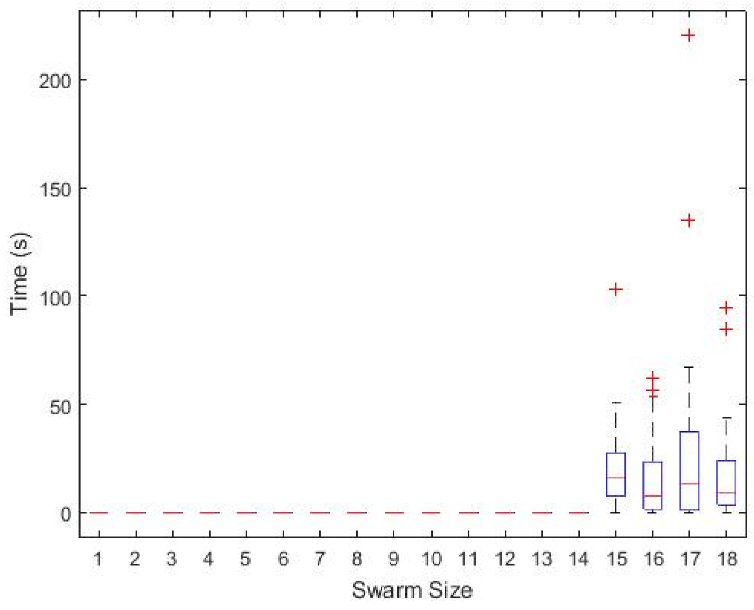


Figure A.45: The exploration time at the target zone in fixed initial positions for five robot wide void.

A.4 Case D

This section shows results that have been collected for case D, consisting of frequency of seed robot, seed robot positions, frequency of robot that detected the target, simulation snapshots, the simulation time, the exploration time including at the initial and target zone, the assembly and moving forward time, and the retreat and disassembly time.

A.4.1 Frequency of Robot ID Became the Seed Robot for Case D

Figure A.46, Figure A.47, Figure A.48 and Figure A.49 show the frequency of robot ID that became the seed robot for case D, for one, two, four and five robot wide void.

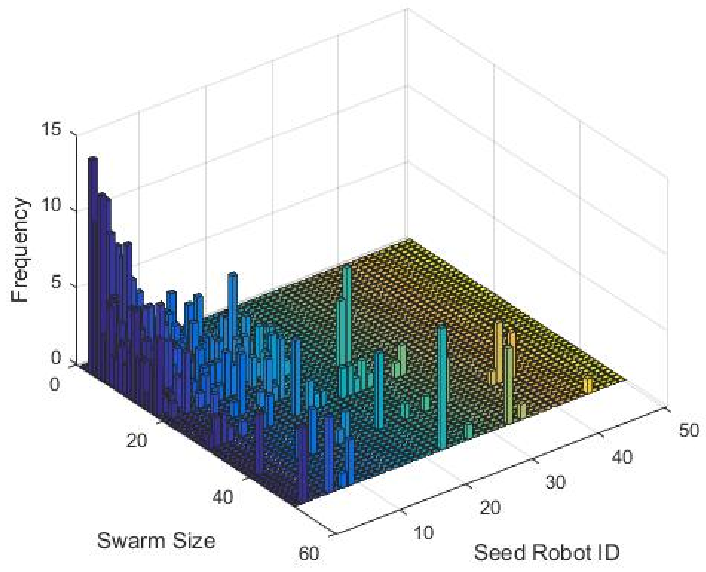


Figure A.46: Frequency of seed ID in random initial positions for one robot wide void.

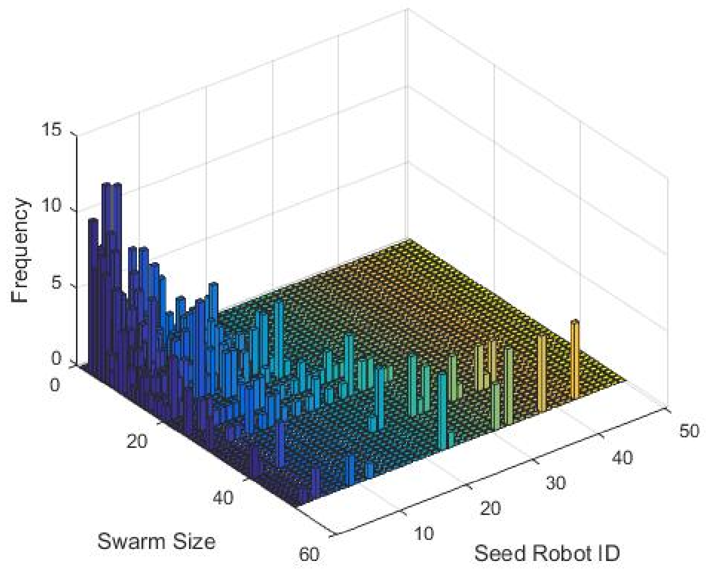


Figure A.47: Frequency of seed ID in random initial positions for two robot wide void.

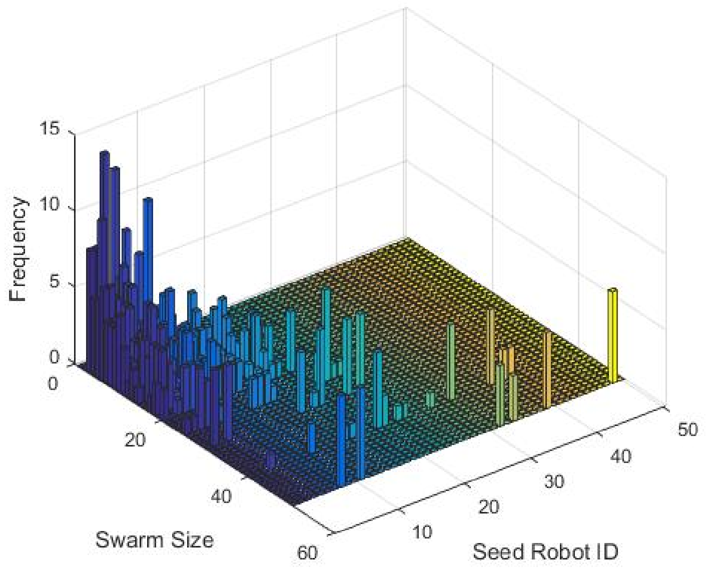


Figure A.48: Frequency of seed ID in random initial positions for four robot wide void.

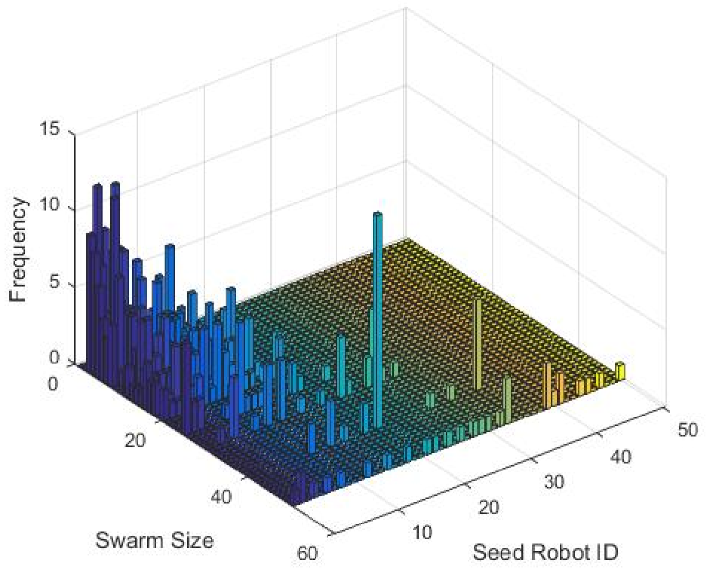


Figure A.49: Frequency of seed ID in random initial positions for five robot wide void.

A.4.2 Positions of Seed Robots for Case D: Random Initial Positions

Figure A.50, Figure A.51, Figure A.52, Figure A.53, Figure A.54 and Figure A.55 show positions of seed robots in random initial positions, for six to eighteen, twenty, twenty five, thirty, forty and fifty robots in the swarm. x-coordinates 220, 225, 230, 235, and 240 show the void size of five, four, three, two and one robot wide voids, respectively.

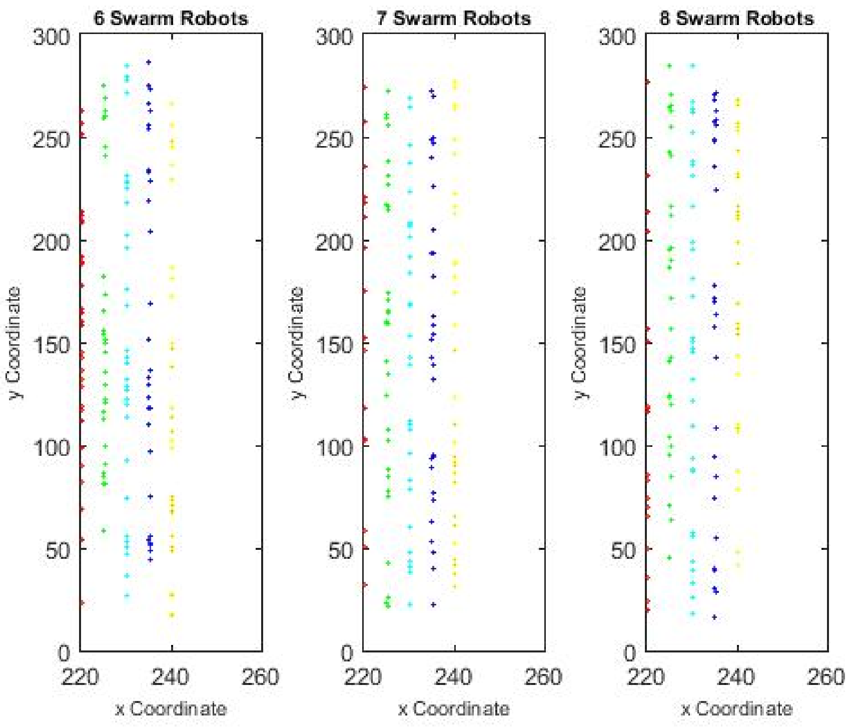


Figure A.50: Positions of seed robots in random initial positions for six to eight robots.

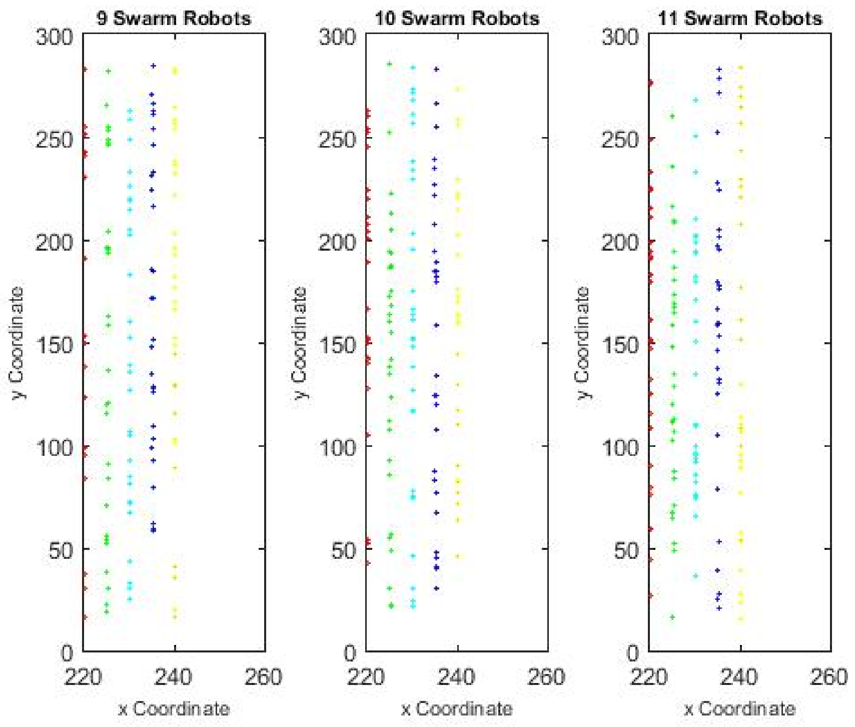


Figure A.51: Positions of seed robots in random initial positions for nine to eleven robots.

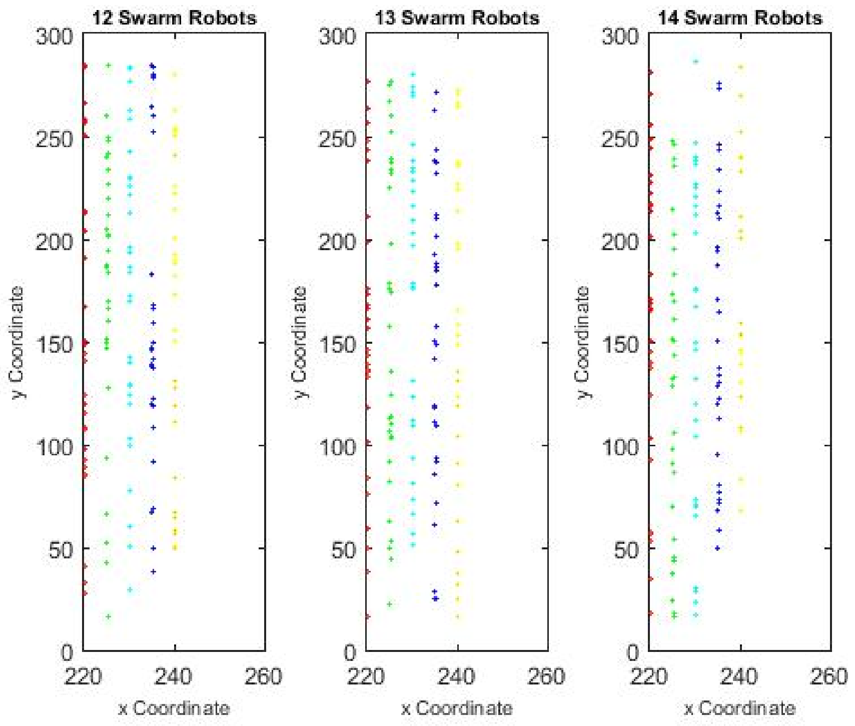


Figure A.52: Positions of seed robots in random initial positions for twelve to fourteen robots.

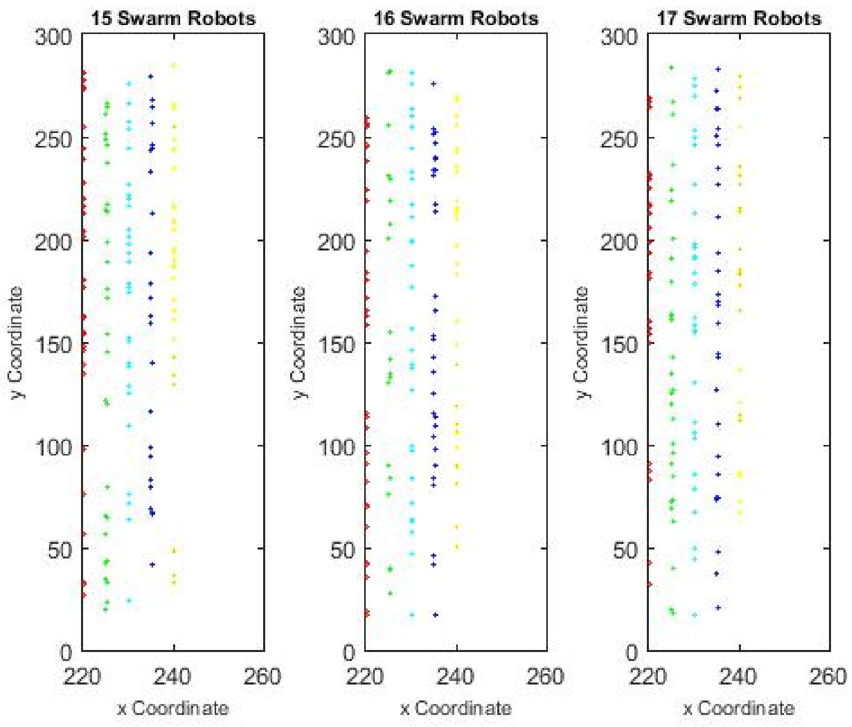


Figure A.53: Positions of seed robots in random initial positions for fifteen to seventeen robots.

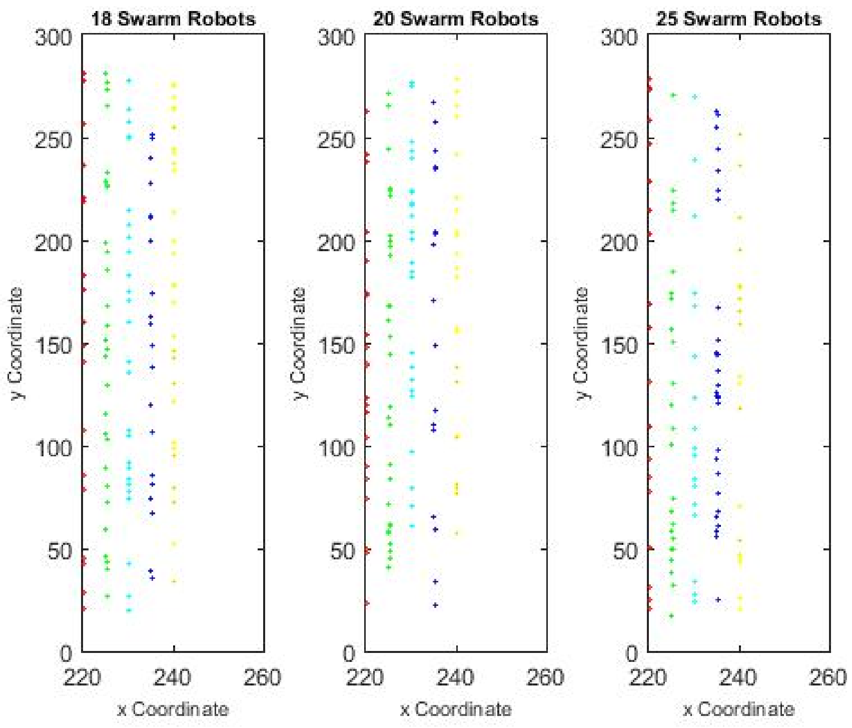


Figure A.54: Positions of seed robots in random initial positions for eighteen, twenty, twenty five robots.

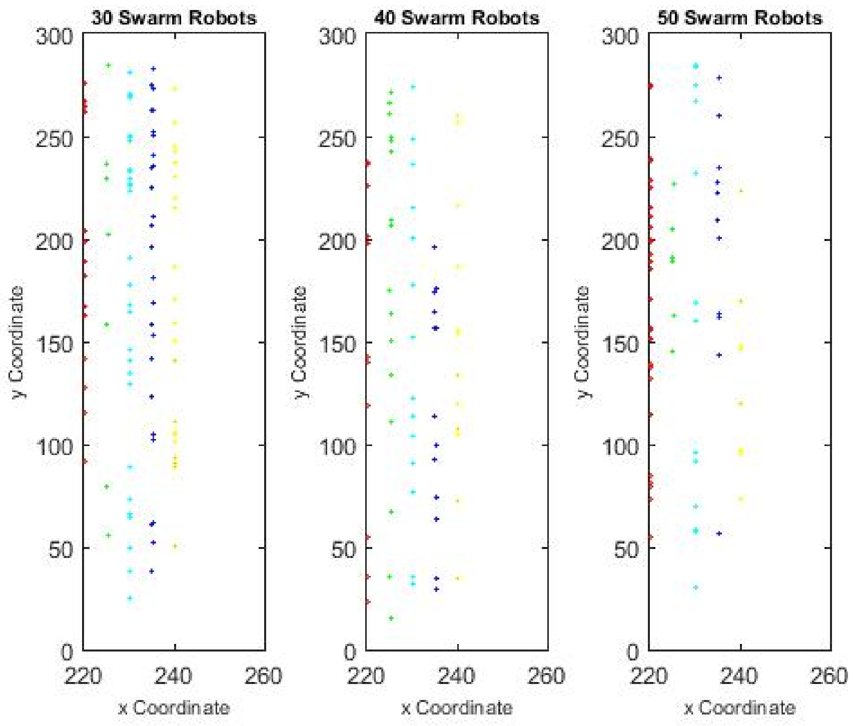


Figure A.55: Positions of seed robots in random initial positions for thirty, forty, fifty robots.

A.4.3 Frequency of Robot ID Detecting the Target for Case D

Figure A.56, Figure A.57, Figure A.58 and Figure A.59 show the frequency of robot ID detecting the target for case D, for one, two, four and five robot wide void.

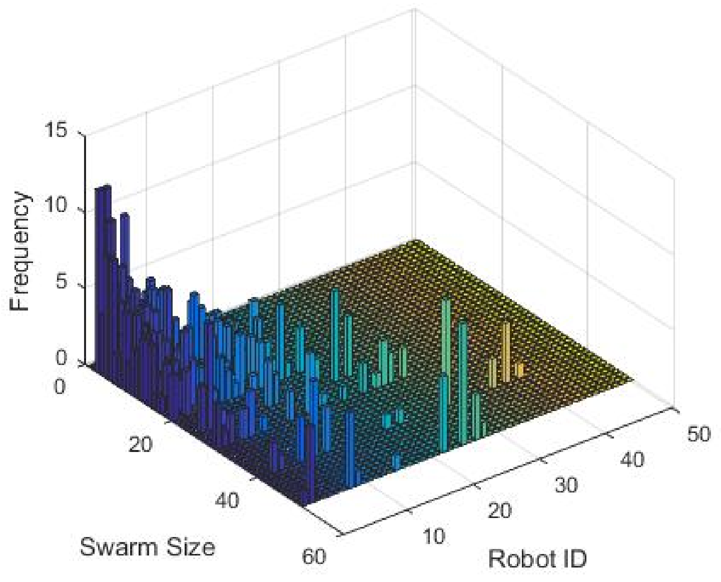


Figure A.56: Frequency of robot ID detecting the target in random initial positions for one robot wide void.

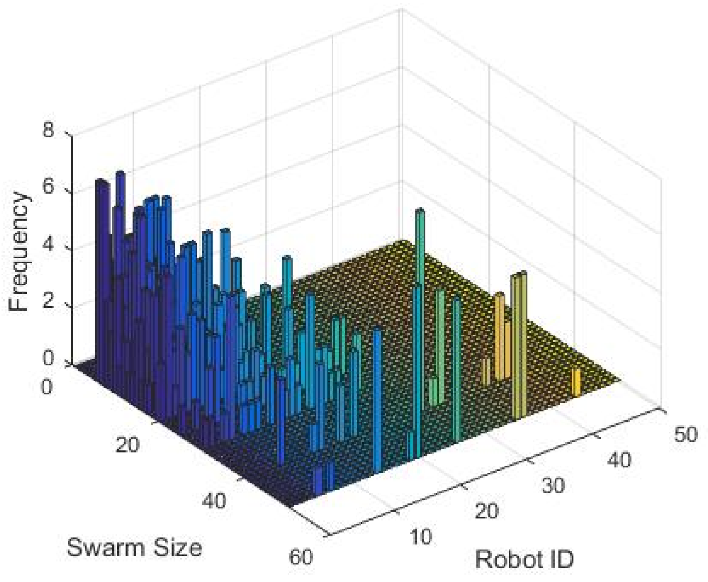


Figure A.57: Frequency of robot ID detecting the target in random initial positions for two robot wide void.

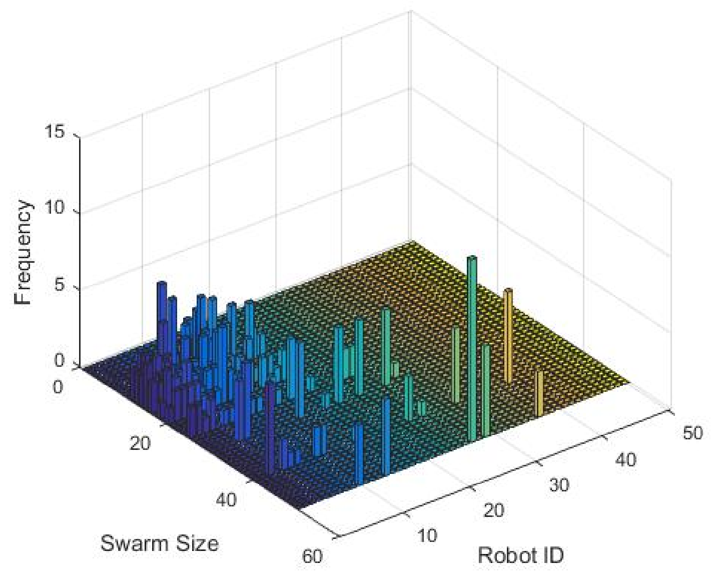


Figure A.58: Frequency of robot ID detecting the target in random initial positions for four robot wide void.

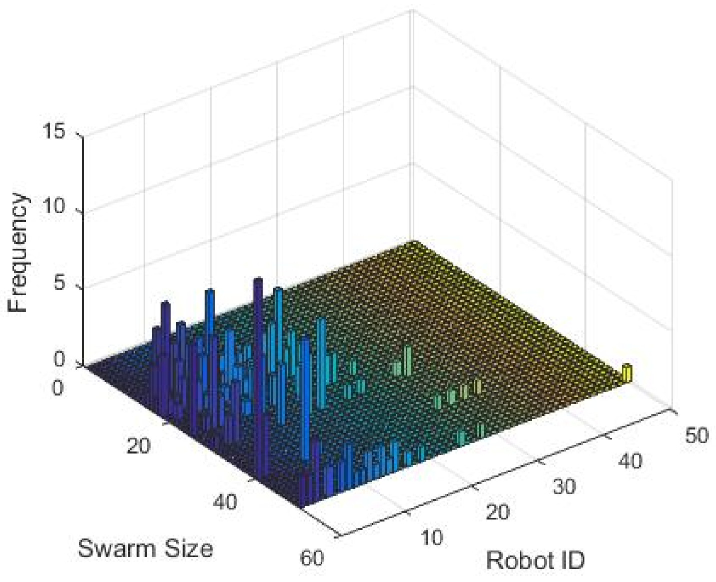


Figure A.59: Frequency of robot ID detecting the target in random initial positions for five robot wide void.

A.4.4 Recorded Time of Simulations for Case D

The following figures show the recorded time of simulations for case D, including T_t, T_s, T_a, T_d and T_o.

A.4.4.1 The Simulation Time (T_t) for Case D

Figure A.60, Figure A.61, Figure A.62 and Figure A.63 show the simulation time for case D, for one, two, four and five robot wide void.

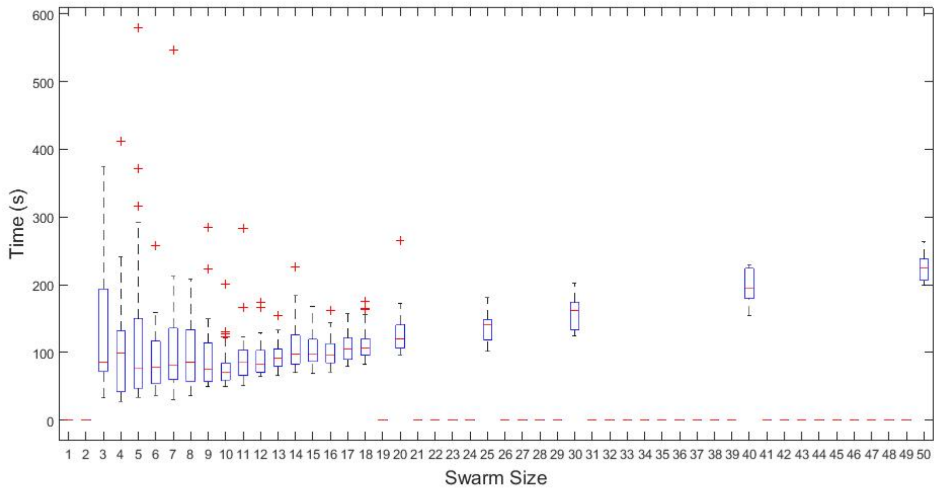


Figure A.60: The simulation time of random initial position for one robot wide void.

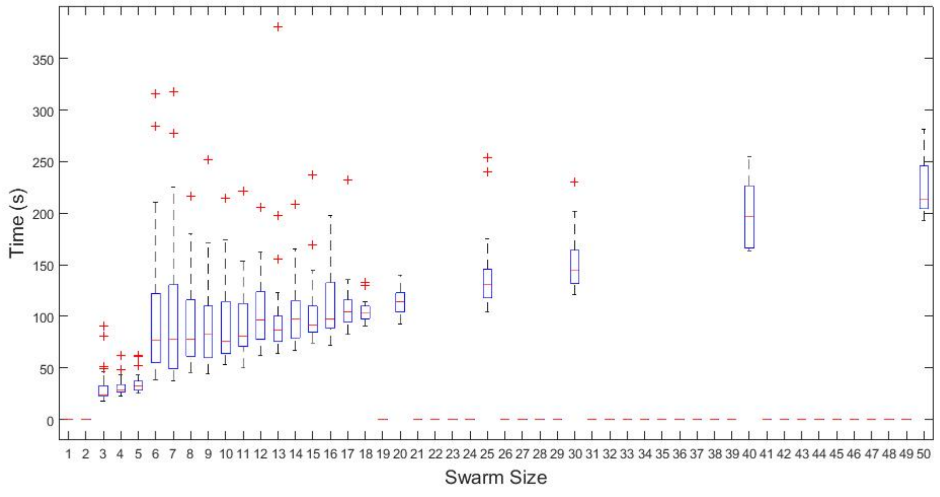


Figure A.61: The simulation time of random initial position for two robot wide void.

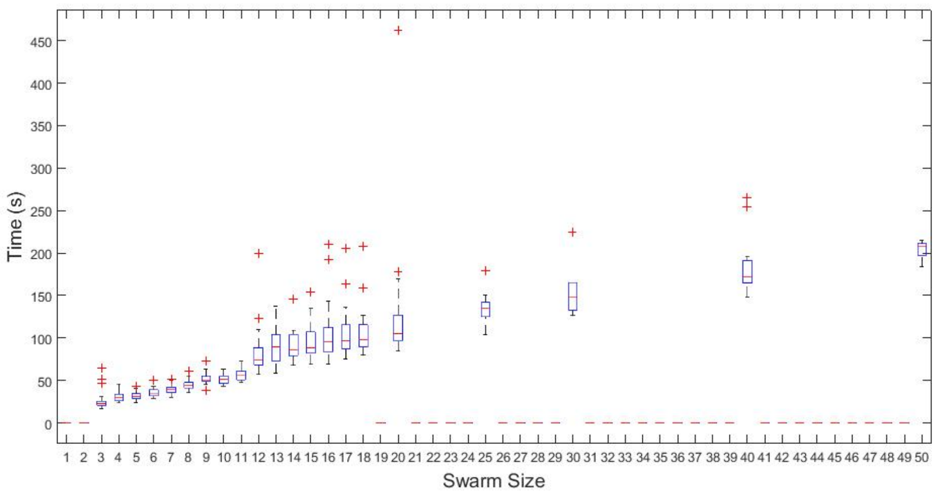


Figure A.62: The simulation time of random initial position for four robot wide void.

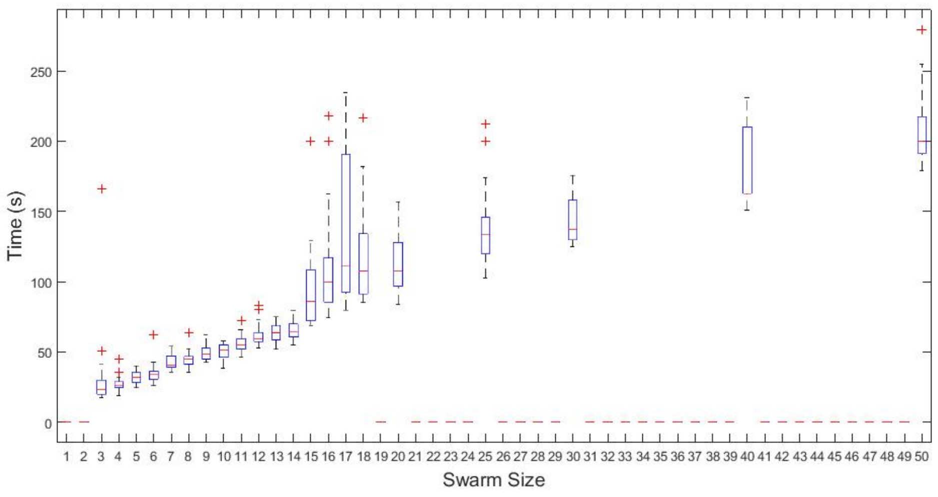


Figure A.63: The simulation time of random initial position for five robot wide void.

A.4.4.2 The Exploration Time at the Initial Zone (T_s) for Case D

Figure A.64, Figure A.65, Figure A.66 and Figure A.67 show the exploration time at the initial zone for case D, for one, two, four and five robot wide void.

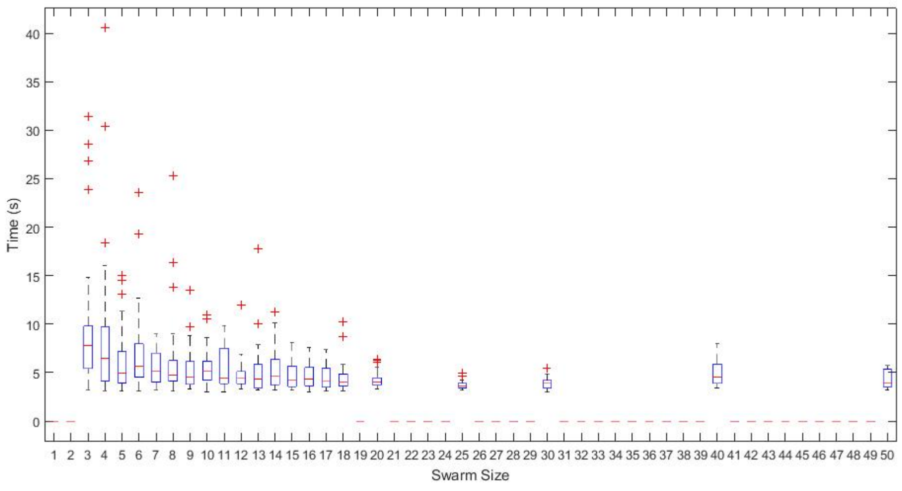


Figure A.64: The exploration time at the initial zone in random initial positions for one robot wide void.

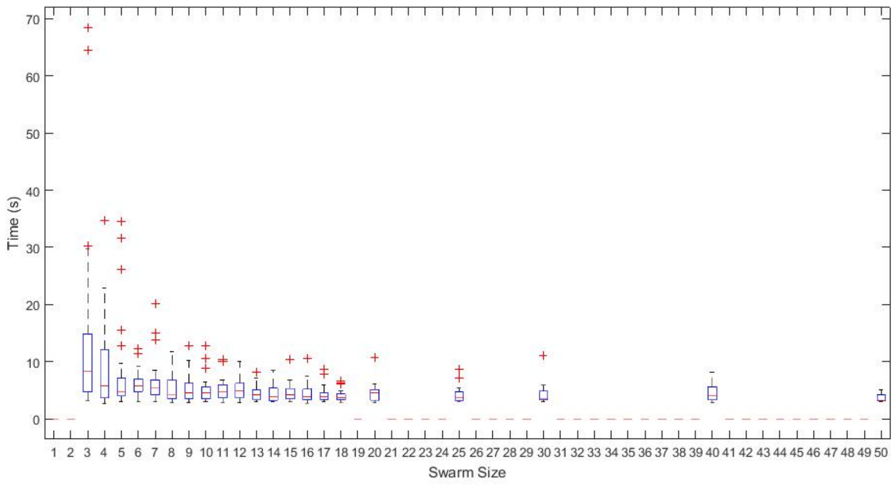


Figure A.65: The exploration time at the initial zone in random initial positions for two robot wide void.

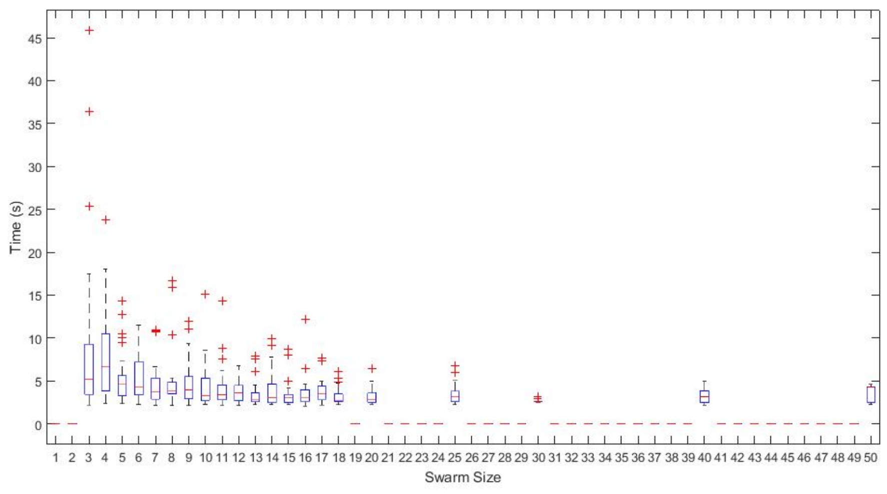


Figure A.66: The exploration time at the initial zone in random initial positions for four robot wide void.

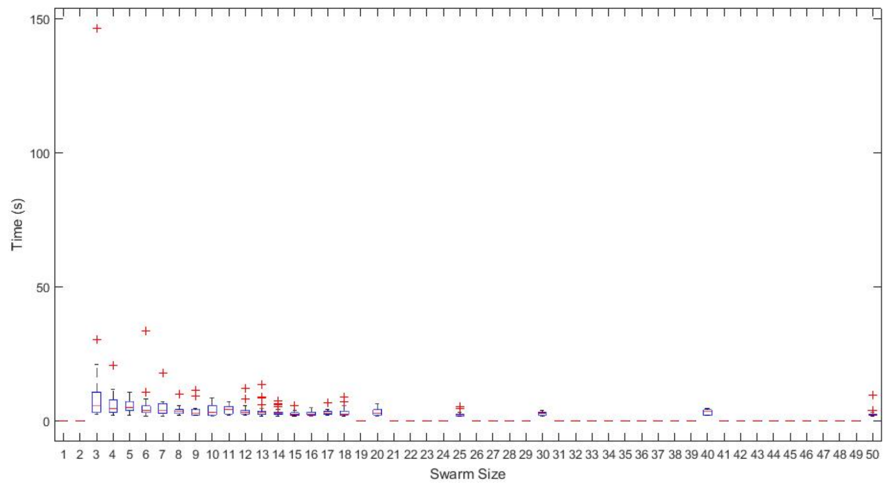


Figure A.67: The exploration time at the initial zone in random initial positions for five robot wide void.

A.4.4.3 The Assembly and Moving Forward Time (T_a) for Case D

Figure A.68, Figure A.69, Figure A.70 and Figure A.71 show the assembly and moving forward time for case D for one, two, four and five robot wide void.

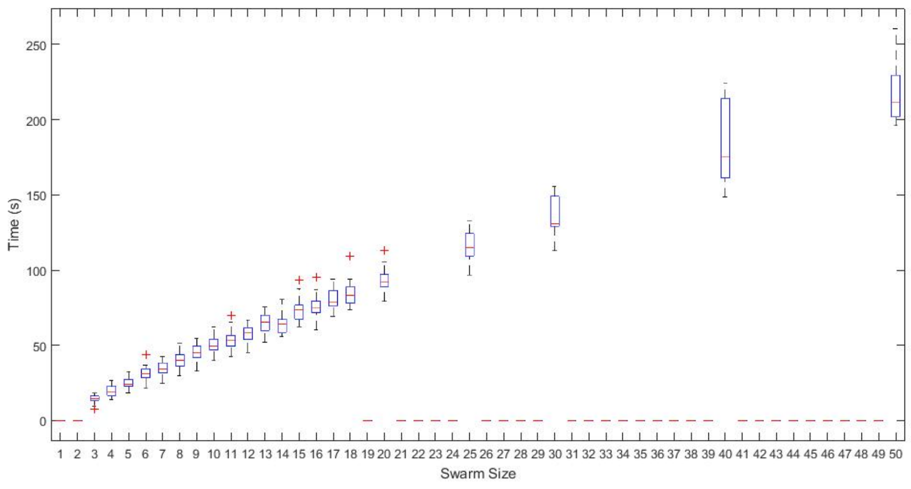


Figure A.68: The assembly and moving forward time in random initial positions for one robot wide void.

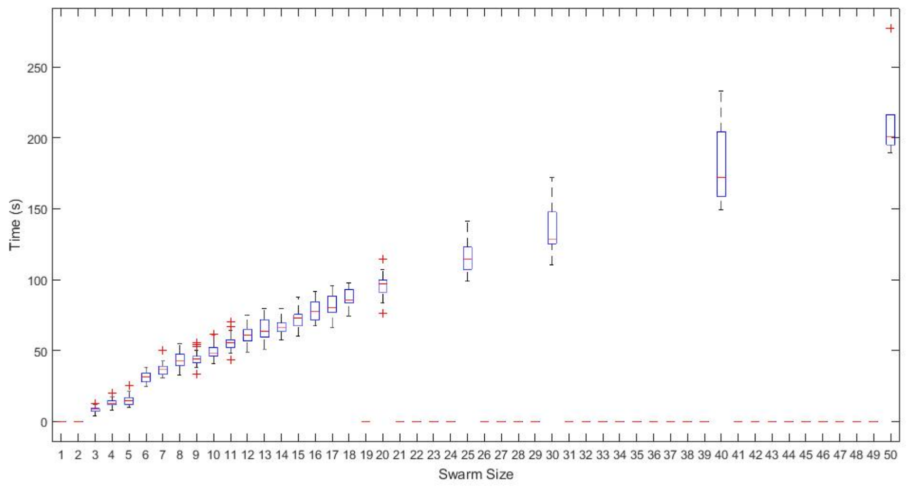


Figure A.69: The assembly and moving forward time in random initial positions for two robot wide void.

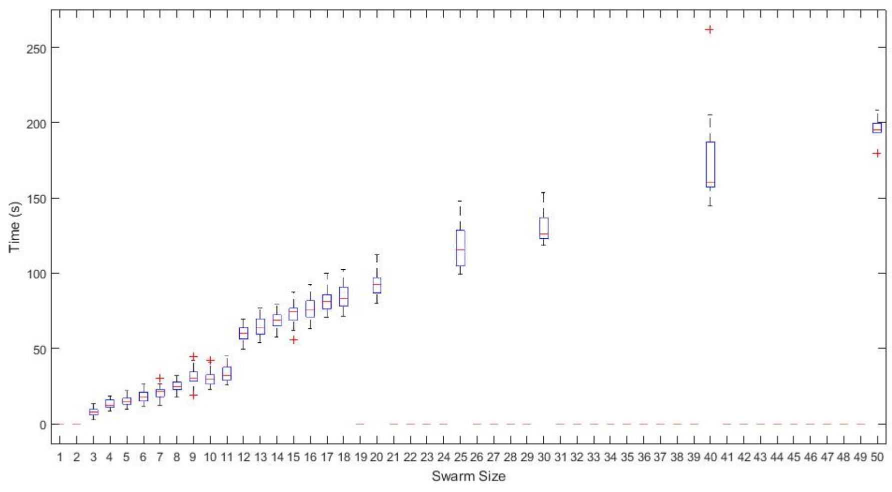


Figure A.70: The assembly and moving forward time in random initial positions for four robot wide void.

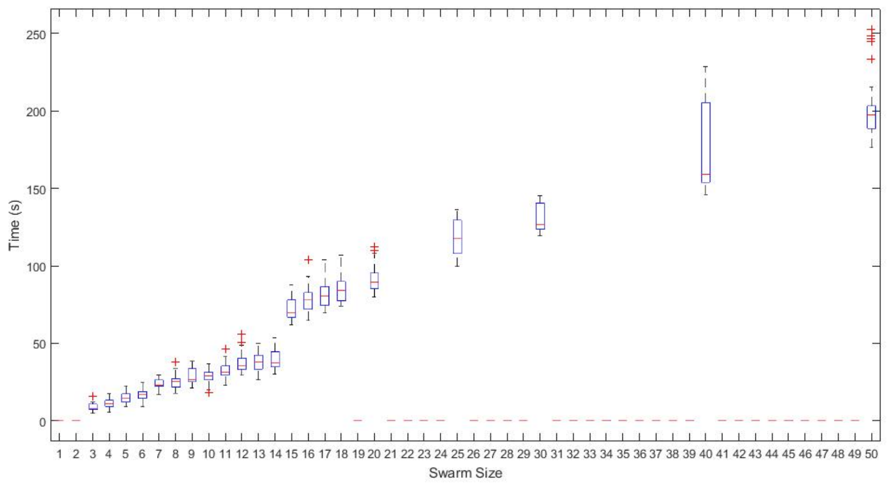


Figure A.71: The assembly and moving forward time in random initial positions for five robot wide void.

A.4.4.4 The Retreat and Disassembly Time (T_d) for Case D

Figure A.72, Figure A.73, Figure A.74 and Figure A.75 show the retreat and disassembly time for case D, for one, two, four and five robot wide void.

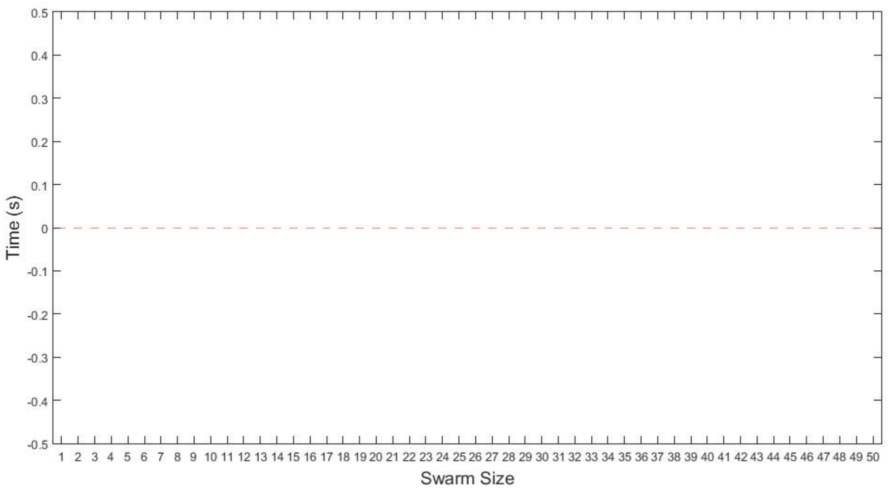


Figure A.72: The retreat and disassembly time for one robot wide void for case D.

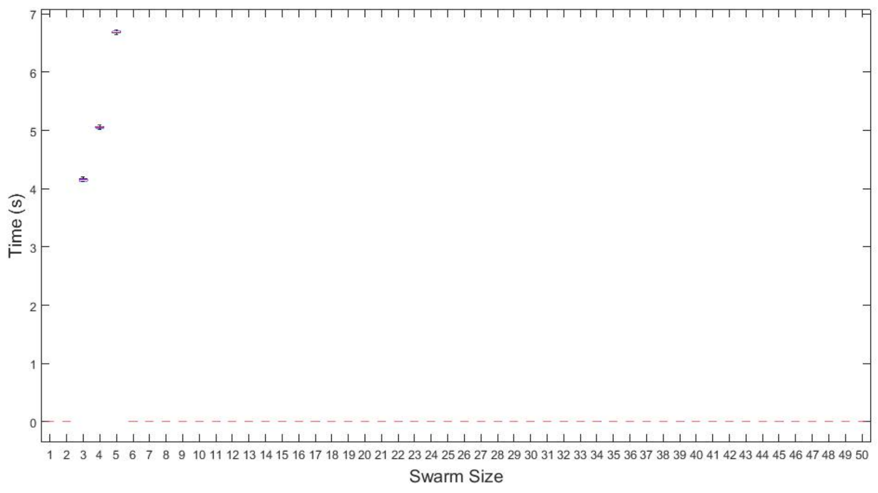


Figure A.73: The retreat and disassembly time for two robot wide void for case D.

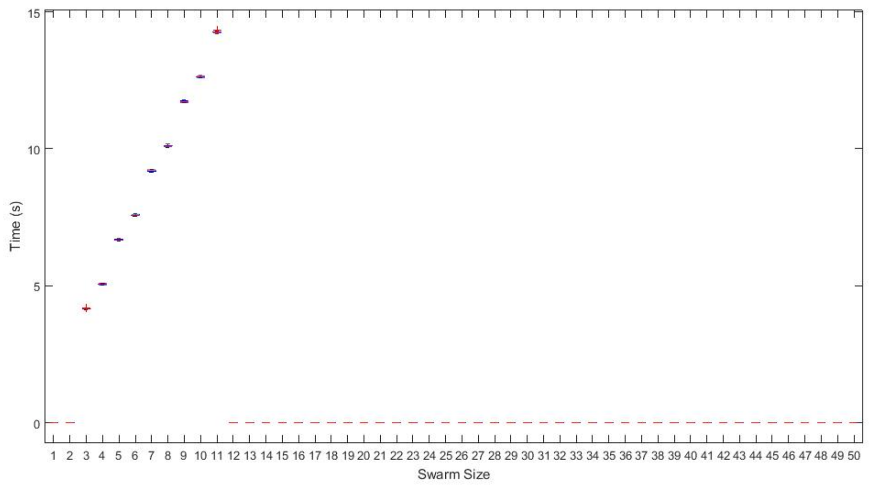


Figure A.74: The retreat and disassembly time for four robot wide void for case D.

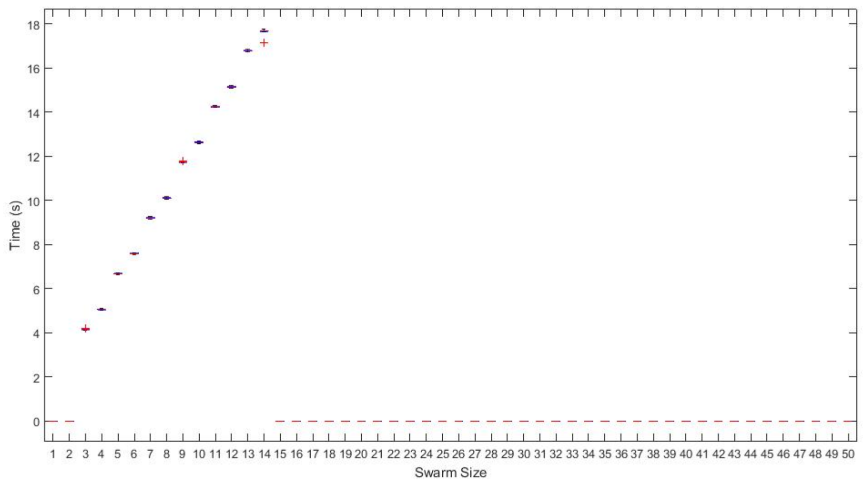


Figure A.75: The retreat and disassembly time for five robot wide void for case D.

A.4.4.5 The Exploration Time at the Target Zone (T_o) for Case D

Figure A.76, Figure A.77, Figure A.78 and Figure A.79 show the exploration time at the target zone for case D, for one, two, four and five robot wide void.

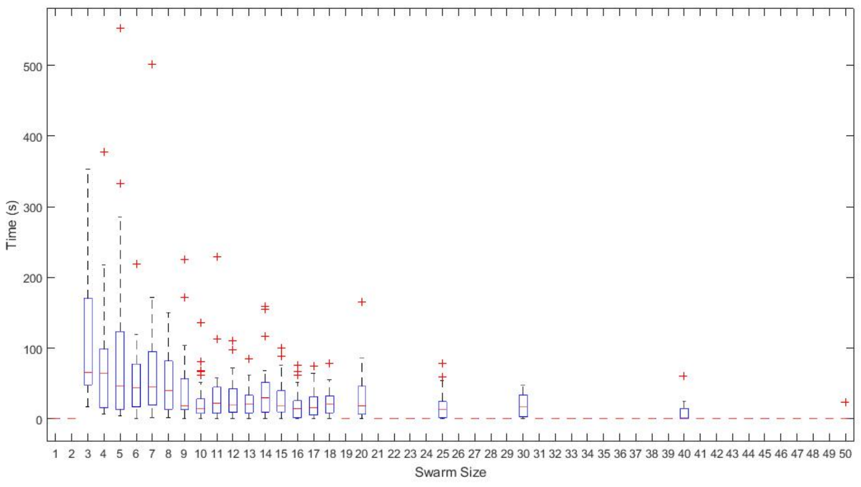


Figure A.76: The exploration time at the target zone for one robot wide void, case D.

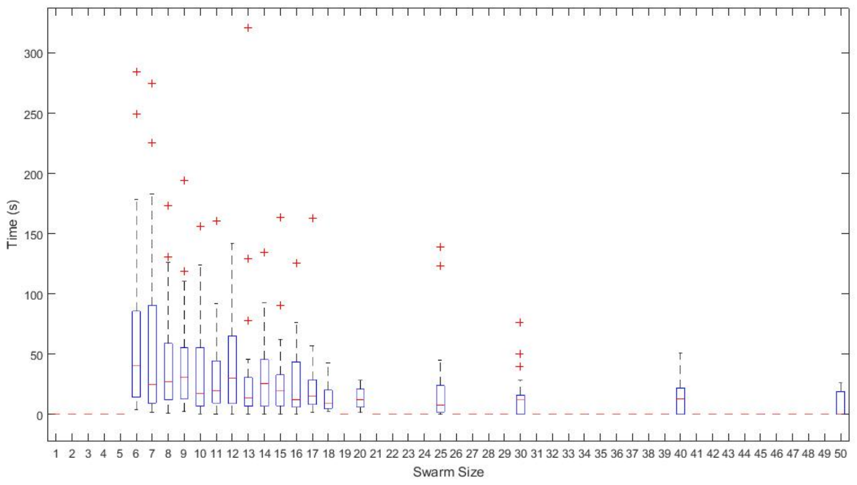


Figure A.77: The exploration time at the target zone for two robot wide void, case D.

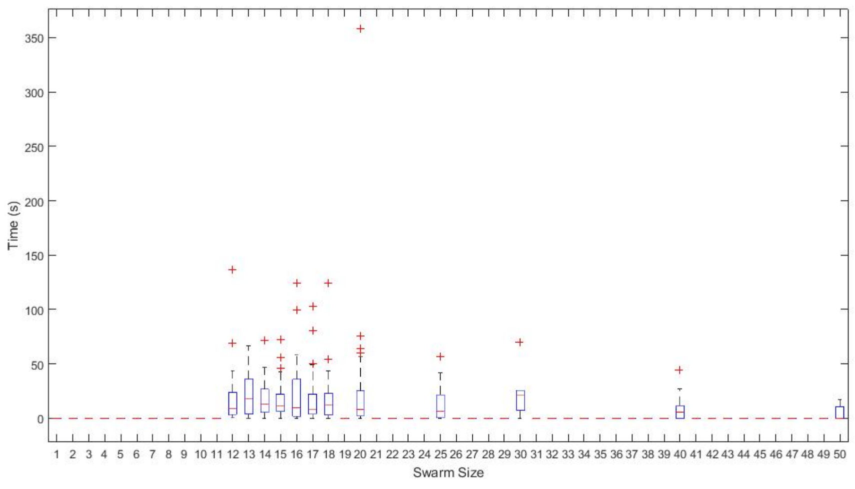


Figure A.78: The exploration time at the target zone for four robot wide void, case D.

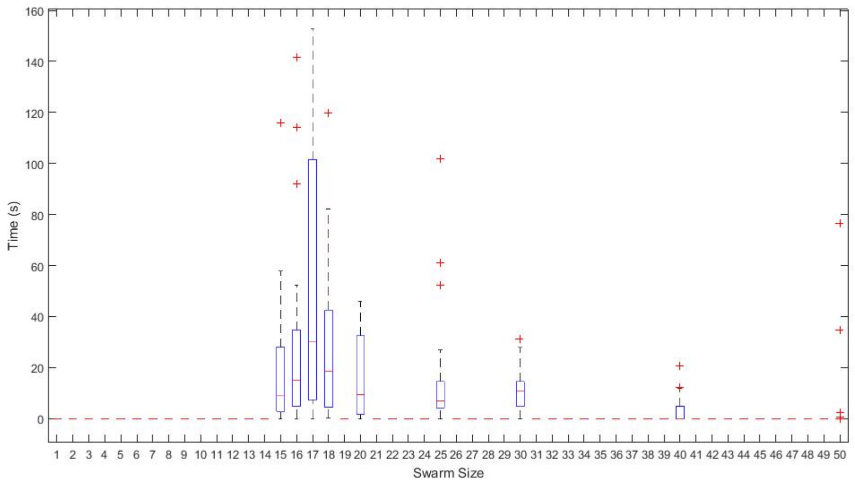


Figure A.79: The exploration time at the target zone for five robot wide void, case D.

Appendix B

Examples of Matlab Scripts for Build Algorithm of Void Crossing

The following Matlab scripts shown in this appendix illustrate the examples of scripts of small and large void crossings. Not all scripts are written down because of repeatation at some point. This appendix is divided into two main sections: shared and specific scripts. Scripts written in shared section are the codes and functions that are common used in all cases such as exploration mode both at initial and target zones, the distance calculation, plotting the robot, and refreshing the data. Scripts written in specific section are the codes for build algorithm for both small and large void crossings.

B.1 Shared Scripts: Exploration Algorithms and Functions

Scripts in this section consist of exploration codes at both initial and target zones, functions to calculate distance and polygon, function to calculate points of the robot body, functions to make the robot's body and wheels, functions to refresh data for robots and their plots, all are applied for all cases and structures. These scripts and functions are the final version, mainly are written in the complex structure. However, the difference to the cases in simple line structure is only the variables and the cliff distance used. For simple line structure cases A and B, the cliff distance used is 6cm, and 5cm for the rest of simple line and complex structures. There are also functions for free robots to explore at the initial and target zones when the structure is being constructed for cases C and D of simple line structure and also for the complex structure.

B.1.1 Script for Exploration at the Initial Zone

Script: Exploration at the Initial Zone

```
while keepLooping == true
    while SearchStart == true
        T = T + dt; tStart = T;

        [robot, border, cliff, dynObstDistance, target] = refreshBotData(robot, border, cliff, map, target, dt, N);
        refreshPlotData(robot, map, target, N);

        for i = 1:N
            robot(i).theta = wrapToPi(robot(i).theta); [robot] = calcPoints(robot, i);

            if robot(i).v > vmax
                robot(i).v = vmax;
            end % of velocity limitation

            if robot(i).dtheta > dthetamax
                robot(i).dtheta = dthetamax;
            elseif robot(i).dtheta < -dthetamax
                robot(i).dtheta = -dthetamax;
            end % of turning angle limitation
            robot(i).v = robot(i).v + robot(i).ddistance/100;

            for r = rand(N,1)
                if r(i) < probl
                    robot(i).dtheta = robot(i).dtheta - 3;
                elseif r(i) < 1-probr
                    robot(i).dtheta = robot(i).dtheta;
                else
                    robot(i).dtheta = robot(i).dtheta + 3;
                end
            end % of random turning angle

            if (border.distance(i).a < 15 || border.distance(i).b < 15 || border.distance(i).c < 15 || ...
                    border.distance(i).d < 15)
                robot(i).v = -vmax/13.89; robot(i).dtheta = -dthetamax; robot(i).v = 5*vmax/13.89;
            end % of avoiding collision with borders

            for j = i:N
                if i == j
                    continue
                end
                if dynObstDistance(i,j) < 15
                    robot(i).v = 0; robot(j).v = 0;
                    robot(i).theta = robot(i).theta - dthetamax/18; robot(j).theta = robot(j).theta - dthetamax/18;
                    robot(i).v = vmax; robot(j).v = vmax;
                end
            end % of avoiding collision with other robots

            if robot(i).x < 10
                robot(i).x = robot(i).x + 15;
            elseif robot(i).y < 10
                robot(i).y = robot(i).y + 15;
            elseif robot(i).y > 990
                robot(i).y = robot(i).y - 15;
            end

            if cliff.distance(i) < 5
                robot(i).coreL = 1; robot(i).start = 0;
                modeStart = modeStart-1; modeCoreL = modeCoreL+1;
                SearchStart = false; BuildCoreL1 = true;
            end
            [robot] = calcPoints(robot, i);
        end
        pause(dt);
        count = count + 1;
        %F = getframe();
        %writeVideo(writerObj, F);
    end % exploration at the initial zone
end
```

B.1.2 Script for Exploration at the Target Zone

Script: Exploration at the Target Zone

```
while keepLooping == true
    while SearchStart == true
        T = T + dt; tOpposite = tOpposite + dt;

        [robot, border, cliff, dynObstDistance, target] = refreshBotData(robot, border, cliff, map, target, dt, N);
        refreshPlotData(robot, map, target, N);

        for i = 1:N
            if robot(i).v > vmax
                robot(i).v = vmax;
            end % of velocity limitation

            if robot(i).dtheta > dthetamax
                robot(i).dtheta = dthetamax;
            elseif robot(i).dtheta < -dthetamax
                robot(i).dtheta = -dthetamax;
            end % of turning angle limitation
            robot(i).theta = wrapToPi(robot(i).theta); robot(i).v = robot(i).v + robot(i).ddistance/100;

            for r = rand(N,1)
                if r(i) < probl
                    robot(i).dtheta = robot(i).dtheta - 3;
                elseif r(i) < 1-probr
                    robot(i).dtheta = robot(i).dtheta;
                else
                    robot(i).dtheta = robot(i).dtheta + 3;
                end
            end % of random turning angle

            if cliff.distance(i) < 5
                robot(i).v = -vmax/13.89; robot(i).dtheta = -dthetamax; robot(i).v = 5*vmax/13.89;
            end

            if (border.distance(i).a < 15 || border.distance(i).b < 15 || border.distance(i).c < 15 || ...
                    border.distance(i).d < 15)
                robot(i).v = -vmax/13.89; robot(i).dtheta = -dthetamax; robot(i).v = 5*vmax/13.89;
            end % of avoiding collision with borders

            for j = i:N
                if i == j
                    continue
                end
                if dynObstDistance(i,j) < 15
                    robot(i).v = 0; robot(j).v = 0;
                    robot(i).theta = robot(i).theta - dthetamax/18; robot(j).theta = robot(j).theta - dthetamax/18;
                    robot(i).v = vmax; robot(j).v = vmax;
                end
            end % of avoiding collision with other robots

            if robot(i).x > 1990
                robot(i).x = robot(i).x - 15;
            elseif robot(i).y < 10
                robot(i).y = robot(i).y + 15;
            elseif robot(i).y > 990
                robot(i).y = robot(i).y - 15;
            end
            [robot] = calcPoints(robot, i);
        end
        pause(dt);
        count = count + 1;
        %F = getframe(); %writeVideo(writerObj, F);

        if any(target.distance(:) < 15)
            keepLooping = false; fprintf('Mission accomplished.')
            break;
            %close(writerObj);
        end
    end
end
```

B.1.3 Script for Exploration at the Initial Zone for Failed Crossing

Script: Exploration at the Initial Zone after Failed Crossing

```
while keepLooping == true
  while (Backward3 == false) && (SearchBack == true)
    T = T + dt; tStartBack = tStartBack + dt;

    [robot, border, cliff, dynObstDistance, target] = refreshBotData(robot, border, cliff, map, target, dt, N);
    refreshPlotData(robot, map, target, N);

    for i = 1:N
      robot(i).theta = wrapToPi(robot(i).theta);
      if robot(i).v > vmax
        robot(i).v = vmax;
      end % of velocity limitation

      if robot(i).dtheta > dthetamax
        robot(i).dtheta = dthetamax;
      elseif robot(i).dtheta < -dthetamax
        robot(i).dtheta = -dthetamax;
      end % of turning angle limitation
      robot(i).v = robot(i).v + robot(i).ddistance/100;

      for r = rand(N,1)
        if r(i) < probl
          robot(i).dtheta = robot(i).dtheta - 3;
        elseif r(i) < 1-probr
          robot(i).dtheta = robot(i).dtheta;
        else
          robot(i).dtheta = robot(i).dtheta + 3;
        end
      end % of random turning angle

      if cliff.distance(i) < 5
        robot(i).v = -vmax/13.89; robot(i).dtheta = -dthetamax; robot(i).v = 5*vmax/13.89;
      end

      if (border.distance(i).a < 15 || border.distance(i).b < 15 || border.distance(i).c < 15 || ...
          border.distance(i).d < 15)
        robot(i).v = -vmax/13.89; robot(i).dtheta = -dthetamax; robot(i).v = 5*vmax/13.89;
      end % of avoiding collision with borders

      for j = i:N
        if i == j
          continue
        end
        if dynObstDistance(i,j) < 15
          robot(i).v = 0; robot(j).v = 0; robot(i).theta = robot(i).theta - dthetamax/18;
          robot(j).theta = robot(j).theta - dthetamax/18; robot(i).v = vmax; robot(j).v = vmax;
        end
      end % of avoiding collision with other robots

      if robot(i).x < 10
        robot(i).x = robot(i).x + 10;
      elseif robot(i).x > 490
        robot(i).y = robot(i).x - 10;
      elseif robot(i).y < 10
        robot(i).y = robot(i).y + 10;
      elseif robot(i).y > 290
        robot(i).y = robot(i).y - 10;
      end
      [robot] = calcPoints(robot, i);
    end
    pause(dt);
    count = count + 1;
    %F = getframe(); %writeVideo(writerObj, F);

    tStartBack > 4.99
      keepLooping = false; fprintf('Mission failed.')
      break; %close(writerObj);
    end
  end % all robots search mode and stop moving after 5s
end
```

B.1.4 Function for Exploration at the Initial Zone for Free Robots

Function: Exploration at the Initial Zone for Free Robots

```
function [robot, r] = searchAssemble(robot, i, vmax, dthetamax, probr, probl, border, cliff, dynObstDistance)
global N
    if robot(i).v > vmax
        robot(i).v = vmax;
    end % of velocity limitation

    if robot(i).dtheta > dthetamax
        robot(i).dtheta = dthetamax;
    elseif robot(i).dtheta < -dthetamax
        robot(i).dtheta = -dthetamax;
    end % of turning angle limitation
    robot(i).theta = wrapToPi(robot(i).theta); robot(i).v = robot(i).v + robot(i).ddistance/100;

    for r = rand(N,1)
        if r(i) < probl
            robot(i).dtheta = robot(i).dtheta - 3;
        elseif r(i) < 1-probr
            robot(i).dtheta = robot(i).dtheta;
        else
            robot(i).dtheta = robot(i).dtheta + 3;
        end
    end % of random turning angle

    if (border.distance(i).a < 15 || border.distance(i).b < 15 || border.distance(i).c < 15 || ...
            border.distance(i).d < 15)
        robot(i).v = -vmax/13.89;
        robot(i).dtheta = -dthetamax;
        robot(i).v = 5*vmax/13.89;
    end % of avoiding collision with borders

    if cliff.distance(i) < 5
        robot(i).v = -vmax/13.89;
        robot(i).dtheta = -dthetamax;
        robot(i).v = 5*vmax/13.89;
    end

    if cliff.distance(i) < 0
        robot(i).x = robot(i).x - 2;
    end

    for j = i:N
        if i == j
continue
        end
        if dynObstDistance(i,j) < 15
            if robot(i).y > 500
                robot(i).x = robot(i).x - 1;
                robot(i).y = robot(i).y - 1;
                robot(i).theta = robot(i).theta + pi;
                robot(j).x = robot(j).x;
                robot(j).y = robot(j).y;
            elseif robot(i).y ¡= 500
                robot(i).x = robot(i).x + 1;
                robot(i).y = robot(i).y + 1;
                robot(i).theta = robot(i).theta + pi;
                robot(j).x = robot(j).x;
                robot(j).y = robot(j).y;
            end
        end
    end % of avoiding collision with other robots

    if robot(i).x < 10
        robot(i).x = robot(i).x + 15;
    elseif robot(i).y < 10
        robot(i).y = robot(i).y + 15;
    elseif robot(i).y > 990
        robot(i).y = robot(i).y - 15;
    end
end
```

B.1.5 Function for Exploration at the Target Zone for Free Robots

Function: Exploration at the Target Zone for Free Robots

```
function [robot, r] = searchDisassemble(robot, i, vmax, dthetamax, probr, probl, border, cliff, dynObstDistance)
global N
    if robot(i).v > vmax
        robot(i).v = vmax;
    end % of velocity limitation

    if robot(i).dtheta > dthetamax
        robot(i).dtheta = dthetamax;
    elseif robot(i).dtheta < -dthetamax
        robot(i).dtheta = -dthetamax;
    end % of turning angle limitation
    robot(i).theta = wrapToPi(robot(i).theta); robot(i).v = robot(i).v + robot(i).ddistance/100;

    for r = rand(N,1)
        if r(i) < probl
            robot(i).dtheta = robot(i).dtheta - 3;
        elseif r(i) < 1-probr
            robot(i).dtheta = robot(i).dtheta;
        else
            robot(i).dtheta = robot(i).dtheta + 3;
        end
    end % of random turning angle

    if (border.distance(i).a < 15 || border.distance(i).b < 15 || border.distance(i).c < 15 || ...
            border.distance(i).d < 15)
        robot(i).v = -vmax/13.89;
        robot(i).dtheta = -dthetamax;
        robot(i).v = 5*vmax/13.89;
    end % of avoiding collision with borders

    if cliff.distance(i) < 5
        robot(i).v = -vmax/13.89;
        robot(i).dtheta = -dthetamax;
        robot(i).v = 5*vmax/13.89;
    end

    if cliff.distance(i) < 0
        robot(i).x = robot(i).x + 2;
    end

    for j = i:N
        if i == j
continue
        end
        if dynObstDistance(i,j) < 15
            if robot(i).y > 500
                robot(i).x = robot(i).x - 1;
                robot(i).y = robot(i).y - 1;
                robot(i).theta = robot(i).theta + pi;
                robot(j).x = robot(j).x;
                robot(j).y = robot(j).y;
            elseif robot(i).y ¡= 500
                robot(i).x = robot(i).x + 1;
                robot(i).y = robot(i).y + 1;
                robot(i).theta = robot(i).theta + pi;
                robot(j).x = robot(j).x;
                robot(j).y = robot(j).y;
            end
        end
    end % of avoiding collision with other robots

    if robot(i).x > 1990
        robot(i).x = robot(i).x - 15;
    elseif robot(i).y < 10
        robot(i).y = robot(i).y + 15;
    elseif robot(i).y > 990
        robot(i).y = robot(i).y - 15;
    end
end
```

B.1.6 Function to Calculate the Distance

Function: Calculate the Distance

```
function l = calcDistance(varargin)
    % calculating the distance between two points
    p1 = varargin1;
    p2 = varargin2;
    l = sqrt(sum((p2-p1).^2));
end
```

B.1.7 Function to Calculate the Polygon

Function: Calculate the Polygon

```
function d = calcDistancePoly(x, y, xv, yv)
    % if (xv, yv) is not closed, close it
    xv = xv(:);
    yv = yv(:);
    Nv = length(xv);
    if ((xv(1) ~= xv(Nv)) || (yv(1) ~=(Nv)))
        xv = [xv; xv(1)];
        yv = [yv; yv(1)];
        %Nv = Nv + 1;
    end

    % linear parameters of segments that connect the vertices
    %Ax + By + C = 0
    A = -diff(yv);
    B = diff(xv);
    C = yv(2:end).*xv(1:end-1) - xv(2:end).*yv(1:end-1);

    % find the projection of point (x,y) on each rib
    AB = 1./(A.^2 + B.^2);
    vv = (A*x+B*y+C);
    xp = x - (A.*AB).*vv;
    yp = y - (B.*AB).*vv;

    % test for the case where a polygon rib is either horizontal or vertical - from Eric Schmitz
    id = find(diff(xv) == 0);
    xp(id) = xv(id);
    clear id
    id = find(diff(yv) == 0);
    yp(id) = yv(id);

    % find all cases where projected point is inside the segment
    idx_x = (((xp>=xv(1:end-1)) & (xp<=xv(2:end))) | ((xp>=xv(2:end)) & (xp<=xv(1:end-1))));
    idx_y = (((yp>=yv(1:end-1)) & (yp<=yv(2:end))) | ((yp>=yv(2:end)) & (yp<=yv(1:end-1))));
    idx = idx_x & idx_y;

    % distance from point (x,y) to the vertices
    dv = sqrt((xv(1:end-1)-x).^2 + (yv(1:end-1)-y).^2);

    if (~any(idx)) % all projections are outside of polygon ribs
        [d,I] = min(dv);
        %x_poly = xv(I);
        %y_poly = yv(I);
    else
        % distance from point (x,y) to the projection on ribs
        dp = sqrt((xp(idx)-x).^2 + (yp(idx)-y).^2);
        [min_dv,I1] = min(dv);
        [min_dp,I2] = min(dp);
        [d,I] = min([min_dv min_dp]);
        if I==2,
            idxs = find(idx);
        end
    end

    if(inpolygon(x, y, xv, yv))
        d = -d;
    end
end
```

B.1.8 Function to Calculate Points of Robot's Body and Wheels

Function: Calculate Points of Robot's Body and Wheels

```
function [robot] = calcPoints(robot, i)
    % transformation matrix
    robot(i).C = [cos(robot(i).theta), -sin(robot(i).theta); sin(robot(i).theta), cos(robot(i).theta)];

    % robot body point positions
    robot(i).body = makeBody(robot(i).size.hl, robot(i).size.hw, robot(i).x, robot(i).y, robot(i).C);

    % wheel positions
    robot(i).wr = robot(i).C * [0; -robot(i).size.wd/2] + [robot(i).x; robot(i).y];
    robot(i).wl = robot(i).C * [0; robot(i).size.wd/2] + [robot(i).x; robot(i).y];

    % centre of mass position (in the middle)
    robot(i).com = (robot(i).wl + robot(i).wr) / 2;

    % robot wheel point positions
    [robot(i).wrpoints, robot(i).wrangle, robot(i).wrradius] = makeWheel(robot(i).wr, robot(i).theta, ...
        robot(i).omega, -1, robot(i).size); % -1 for right wheel
    [robot(i).wlpoints, robot(i).wlangle, robot(i).wlradius] = makeWheel(robot(i).wl, robot(i).theta, ...
        robot(i).omega, +1, robot(i).size); % +1 for left wheel

    % ground sensor
    robot(i).ground = robot(i).C * [robot(i).size.hl-2; 0] + [robot(i).x; robot(i).y];
    robot(i).frontRight = robot(i).C * [robot(i).size.hl; -robot(i).size.wd/2+1.5] + [robot(i).x; robot(i).y];
    robot(i).frontLeft = robot(i).C * [robot(i).size.hl; robot(i).size.wd/2-1.5] + [robot(i).x; robot(i).y];
    robot(i).backRight = robot(i).C * [-robot(i).size.hl; -robot(i).size.wd/2+1.5] + [robot(i).x; robot(i).y];
    robot(i).backLeft = robot(i).C * [-robot(i).size.hl; robot(i).size.wd/2-1.5] + [robot(i).x; robot(i).y];

    % turning radius of the reference point
    robot(i).r = 2*robot(i).size.hl/tan(robot(i).dtheta);
end
```

B.1.9 Function to Make the Robot's Body

Function: Make the Robot's Body

```
function [points] = makeBody(hl, hw, x, y, C)
    % square ––––––––––––––––––––––––––––––––––––––––
    points = zeros(2, 4);
    points(1, :) = [hl, hl, -hl, -hl];
    points(2, :) = [-hw, hw, hw, -hw];

    % octagon ––––––––––––––––––––––––––––––––––––––––
    %points = zeros(2, 8);
    %points(1, :) = [-0.414*hl, 0.414*hl, hl, hl, 0.414*hl, -0.414*hl, -hl, -hl];
    %points(2, :) = [hw, hw, 0.414*hw, -0.414*hw, -hw, -hw, -0.414*hw, 0.414*hw];

    points = C * points;

    points(1, :) = points(1, :) + x;
    points(2, :) = points(2, :) + y;

end
```

B.1.10 Function to Make the Robot's Wheels

Function: Make the Robot's Wheels

```
function [points, angle, rad] = makeWheel(pos, angle, turnangle, which, size)
    wr = size.wr;
    ww = size.ww;
    wd = size.wd;

    if turnangle > 0
        angle = angle + atan2(0, -which * wd/2);
        rad = 0;
    elseif turnangle < 0
        angle = angle - atan2(0, which * wd/2);
        rad = 0;
    else
        rad = Inf;
    end
    C = [cos(angle), -sin(angle); sin(angle), cos(angle)];
    points = [wr, wr, -wr, -wr; -ww/2, ww/2, ww/2, -ww/2];
    points = C*points;
    points(1, :) = points(1, :) + pos(1);
    points(2, :) = points(2, :) + pos(2);
end
```

B.1.11 Function to Refresh Robots' Data

Function: Refresh Robots' Data

```
function [robot, border, cliff, dynObstDistance, target] = refreshBotData(robot, border, cliff, map, target, dt, N)
    dynObstDistance = zeros(1,N);
    for i = 1:N
        % refresh robot data ====================================================
        robot(i).dx = robot(i).v*dt*cos(robot(i).theta);
        robot(i).dy = robot(i).v*dt*sin(robot(i).theta);
        robot(i).ddistance = calcDistance([robot(i).dx, robot(i).dy], [robot(i).x, robot(i).y]);

        robot(i).theta = robot(i).theta + robot(i).dtheta*dt;
        robot(i).omega = robot(i).theta / dt;
        robot(i).x = robot(i).x + robot(i).dx;
        robot(i).y = robot(i).y + robot(i).dy;

        % refresh border data ===================================================
        border.distance(i).a = calcDistance(0, robot(i).y);
        border.distance(i).b = calcDistance(2000, robot(i).x);
        border.distance(i).c = calcDistance(1000, robot(i).y);
        border.distance(i).d = calcDistance(0, robot(i).x);

        % refresh cliff data ====================================================
        cliff.distance(i) = calcDistancePoly(robot(i).x, robot(i).y, map.vertx, map.verty);
        cliff.distanceRight(i) = calcDistancePoly(robot(i).frontRight(1), robot(i).frontRight(2), map.vertx, ...
            map.verty);
        cliff.distanceLeft(i) = calcDistancePoly(robot(i).frontLeft(1), robot(i).frontLeft(2), map.vertx, map.verty);
        cliff.distanceBackRight(i) = calcDistancePoly(robot(i).backRight(1), robot(i).backRight(2), map.vertx, ...
            map.verty);
        cliff.distanceBackLeft(i) = calcDistancePoly(robot(i).backLeft(1), robot(i).backLeft(2), map.vertx, ...
            map.verty);

        % refresh dynamic obstacle distance data ================================
        for j = i:N
            if i == j
                continue
            end
            dynObstDistance(i,j) = calcDistance([robot(i).x, robot(i).y], [robot(j).x, robot(j).y]);
        end

        % refresh target distance data ==========================================
        target.distance(i) = calcDistance([target.x, target.y], [robot(i).x, robot(i).y]);
    end
end
```

B.1.12 Function to Refresh Robots' Plot

Function: Refresh Robots' Plot

```
function [] = refreshPlotData(robot, map, target, N)
    for i = 1:N
        set(map.robot(i).body, 'XData', robot(i).body(1, :), 'YData', robot(i).body(2, :));
        set(map.robot(i).wr, 'XData', robot(i).wrpoints(1, :), 'YData', robot(i).wrpoints(2, :));
        set(map.robot(i).wl, 'XData', robot(i).wlpoints(1, :), 'YData', robot(i).wlpoints(2, :));
        set(map.robot(i).ground, 'XData', robot(i).ground(1), 'YData', robot(i).ground(2));
        set(map.robot(i).frontRight, 'XData', robot(i).frontRight(1), 'YData', robot(i).frontRight(2));
        set(map.robot(i).frontLeft, 'XData', robot(i).frontLeft(1), 'YData', robot(i).frontLeft(2));
        set(map.robot(i).backRight, 'XData', robot(i).backRight(1), 'YData', robot(i).backRight(2));
        set(map.robot(i).backLeft, 'XData', robot(i).backLeft(1), 'YData', robot(i).backLeft(2));
    end

    set(map.target, 'XData', target.x, 'YData', target.y);
    set(map.cliff, 'XData', map.vertx, 'YData', map.verty);
end
```

B.2 Specific Scripts: Build Algorithms

B.2.1 Small Void Crossing

The Matlab scripts for small void crossing consists of four cases, but only examples of three cases are written down as the different of case C and D is stated in the initiation of initial robot positions. Scripts shown in this section are the application for simple line structure algorithms and all of them are in the main loop **keepLooping**.

B.2.1.1 Case A

Script: Set the Seed Robot Perpendicular to the Void

```
while (Free_Start == false) && (AsDis_0 == true) %1st robot set perpendicular position
    T = T + dt; tAsDis = tAsDis + dt;
    [robot, border, cliff, dynObstDistance, obstacle, target] = refreshBotData(robot, border, cliff, map, ...
        obstacle, target, dt, N);
    for i = 1:N
        cliff.distance(i) = calculateDistancePolygon(robot(i).x, robot(i).y, map.vertx, map.verty);
        cliff.distanceRight(i) = calculateDistancePolygon(robot(i).frontRight(1), robot(i).frontRight(2), ...
            map.vertx, map.verty);
        cliff.distanceLeft(i) = calculateDistancePolygon(robot(i).frontLeft(1), robot(i).frontLeft(2), ...
            map.vertx, map.verty);
    end
    refreshPlotData(robot, map, target, obstacle, N);
    for i = 1:N
        if robot(i).assemble_number == 1
            x1 = i;
            s = numel(x1);
            p1 = 0; p2 = 0; p3 = 0;
            g1 = 0;
            o1 = 0; o2 = 0; o3 = 0;
            ns = s;
            np = p1 + p2 + p3;
            ng = g1;
            no = o1 + o2 + o3;
            if cliff.distanceRight(x1) == cliff.distanceLeft(x1)
                robot(x1).v = 0;
                robot(x1).dtheta = 0;
            elseif cliff.distanceRight(x1) > cliff.distanceLeft(x1)
                robot(x1).v = -vmax;
                robot(x1).dtheta = dthetamax/9;
                robot(x1).v = 0;
            else
                robot(x1).v = -vmax;
                robot(x1).dtheta = -dthetamax/9;
                robot(x1).v = 0;
            end
            xStart = robot(x1).x;
            yStart = robot(x1).y;
        else
            robot(i).v = 0;
            robot(i).dtheta = 0;
        end
        [robot] = calculatePoints(robot, i);
        if robot(i).assemble_number == 1
            if (robot(i).theta < 0.062) && (robot(i).theta > -0.062)
                AsDis_0 = false;
                AsDis_1 = true;
            end
        end
    end
    pause(dt);
    count = count + 1;
    F = getframe();
    writeVideo(writerObj, F);
end
```

Script: Sorting Free Robot

```
while (AsDis_0 == false) && (AsDis_1 == true) %sorting 2nd bot
    T = T + dt; tAsDis = tAsDis + dt;
    for i = 1:N
        if robot(i).assemble_number ~= -1
            assembleDistance(i) = 0;
        else
            assembleDistance(i) = calculateDistance([robot(i).x, robot(i).y], [robot(x1).x, robot(x1).y]);
            sorting = sort(assembleDistance(:));
        end
    end
    shortest = sorting(2);
    for i = 1:N
        if assembleDistance(i) == shortest
            robot(i).assemble_number = 2;
        end
        if robot(i).assemble_number == 2
            x2 = i;
            s = numel(x1);
            p1 = numel(x2); p2 = 0; p3 = 0;
            g1 = 0;
            o1 = 0; o2 = 0; o3 = 0;
            ns = s;
            np = p1 + p2 + p3;
            ng = g1;
            no = o1 + o2 + o3;
            AsDis_1 = false;
            AsDis_2 = true;
        end
    end
    pause(dt);
    count = count + 1;
    F = getframe();
    writeVideo(writerObj, F);
end
```

Script: Calculate the Angle of Recruiter and Recruitee

```
while (AsDis_1 == false) && (AsDis_2 == true) %calculate angle x1-x2
    T = T + dt; tAsDis = tAsDis + dt;
    [robot, border, cliff, dynObstDistance, obstacle, target] = refreshBotData(robot, border, cliff, map, ...
        obstacle, target, dt, N);
    refreshPlotData(robot, map, target, obstacle, N);
    for i = 1:N
        if robot(i).assemble_number == -1
            robot(i).v = 0;
            robot(i).dtheta = 0;
        end
        robot(x1).v = 0;
        robot(x1).dtheta = 0;
        robot(x2).v = 0;
        phi = atan2((robot(x2).y - robot(x1).y),(robot(x2).x - (robot(x1).x - 12)));
        if (phi > 0) && (phi < pi)
            dphi = phi - pi;
        elseif (phi < 0) && (phi > -pi)
            dphi = phi + pi;
        end
        [robot] = calculatePoints(robot, i);
    end
    pause(dt);
    count = count + 1;
    F = getframe();
    writeVideo(writerObj, F);
    AsDis_2 = false;
    AsDis_3 = true;
end
```

Script: Recruitee Moves Closer to Recruiter

```
while (AsDis_2 == false) && (AsDis_3 == true) %x2 gets close to x1
    T = T + dt; tAsDis = tAsDis + dt;
    [robot, border, cliff, dynObstDistance, obstacle, target] = refreshBotData(robot, border, cliff, map, ...
        obstacle, target, dt, N);
    refreshPlotData(robot, map, target, obstacle, N);
    for i = 1:N
        if robot(i).assemble_number == -1
            robot(i).v = 0;
            robot(i).dtheta = 0;
        end
        robot(x1).v = 0;
        robot(x1).dtheta = 0;
        robot(x2).theta = dphi;
        robot(x2).v = vmax;
        botsDistance = calculateDistance([robot(x2).x, robot(x2).y], [robot(x1).x, robot(x1).y]);
        [robot] = calculatePoints(robot, i);
    end
    pause(dt);
    count = count + 1;
    F = getframe();
    writeVideo(writerObj, F);
    if botsDistance < 12
        AsDis_3 = false;
        AsDis_4 = true;
    end
end
```

Script: Recruitee and Recruiter Coupled

```
while (AsDis_3 == false) && (AsDis_4 == true) %x2 sticks to x1
    T = T + dt; tAsDis = tAsDis + dt;
    [robot, border, cliff, dynObstDistance, obstacle, target] = refreshBotData(robot, border, cliff, map, ...
        obstacle, target, dt, N);
    refreshPlotData(robot, map, target, obstacle, N);
    for i = 1:N
        if robot(i).assemble_number == -1
            robot(i).v = 0;
            robot(i).dtheta = 0;
        end
        robot(x1).v = 0; robot(x1).dtheta = 0;
        if (robot(x2).theta > 0) && (robot(x2).theta < pi)
            robot(x2).v = 0;
            robot(x2).dtheta = -dthetamax/9;
        elseif (robot(x2).theta < 0) && (robot(x2).theta > -pi)
            robot(x2).v = 0;
            robot(x2).dtheta = dthetamax/9;
        end
        [robot] = calculatePoints(robot, i);     end
    pause(dt);
    count = count + 1;
    F = getframe();
    writeVideo(writerObj, F);
     if (robot(x2).theta < 0.062) && (robot(x2).theta > -0.062)
        robot(x2).frontRight = robot(x1).backRight;
        robot(x2).frontLeft = robot(x1).backLeft;
        robot(x2).y = robot(x1).y;
        robot(x2).dtheta = 0;
        robot(x2).v = 0;
        if np == ns
            AsDis_4 = false;
            AsDis_5 = true;
        end
    end
end
```

Script: The Structure Moves Forward

```
while (AsDis_8 == false) && (AsDis_9 == true) %x1-x2-x3 move towards the gap; x1 in the gap
  T = T + dt; tAsDis = tAsDis + dt;
  [robot, border, cliff, dynObstDistance, obstacle, target] = refreshBotData(robot, border, cliff, map, ...
    obstacle, target, dt, N);
  refreshPlotData(robot, map, target, obstacle, N);
  for i = 1:N
     if robot(i).assemble_number == -1
      robot(i).v = 0;
      robot(i).dtheta = 0;
    end
    if robot(i).assemble_number ~= -1
      robot(i).dtheta = 0;
      robot(i).v = vmax;
      robot(x2).frontRight = robot(x1).backRight;
      robot(x2).frontLeft = robot(x1).backLeft;
      robot(x2).y = robot(x1).y;
      robot(x3).frontRight = robot(x2).backRight;
      robot(x3).frontLeft = robot(x2).backLeft;
      robot(x3).y = robot(x2).y;
    end
    if robot(i).assemble_number == 2
      if (cliff.distanceRight(i) < 0) || (cliff.distanceLeft(i) < 0)
        s = 0;
        p1 = numel(x2); p2 = numel(x3); p3 = 0;
        g1 = numel(x1);
        o1 = 0; o2 = 0; o3 = 0;
        ns = s;
        np = p1 + p2 + p3;
        ng = g1;
        no = o1 + o2 + o3;
        AsDis_9 = false;
        AsDis_10 = true;
      end
    end
    [robot] = calculatePoints(robot, i);
  end
  pause(dt);
  count = count + 1;
  F = getframe();
  writeVideo(writerObj, F);
end
```

Script: The Front Robot Disassembles

```
while (AsDis_24 == false) && (AsDis_25 == true) %x1 disassemble
    T = T + dt; tAsDis = tAsDis + dt;
    disassembleDistance(1) = calculateDistance([robot(x1).x, robot(x1).y], [robot(x2).x, robot(x2).y]);
    [robot, border, cliff, dynObstDistance, obstacle, target] = refreshBotData(robot, border, cliff, map, ...
        obstacle, target, dt, N);
    refreshPlotData(robot, map, target, obstacle, N);
    for i = 1:N
        if robot(i).assemble_number == 1
            robot(i).v = vmax;
            robot(i).dtheta = dthetamax/6;
            if (border.distance(i).a < 15 || border.distance(i).b < 15 || border.distance(i).c < 15 || ...
                border.distance(i).d < 15)
                robot(i).v = 0;
                robot(i).theta = robot(i).theta - pi/2;
                robot(i).v = vmax;
            end
        else
            robot(i).dtheta = 0;
            robot(i).v = 0;
        end
        if robot(i).assemble_number == 1
            if disassembleDistance(1) > 90
                AsDis_25 = false;
                AsDis_26 = true;
            end
        end
        [robot] = calculatePoints(robot, i);
    end
    pause(dt);
    count = count + 1;
    F = getframe();
    writeVideo(writerObj, F);
end
```

Script: The Structure Moves Backward

```
while (AsDis_42c == false) && (MoveBack_0 == true)
    T = T + dt; tMoveBack = tMoveBack + dt;
    [robot, border, cliff, dynObstDistance, obstacle, target] = refreshBotData(robot, border, cliff, map, ...
        obstacle, target, dt, N);
    refreshPlotData(robot, map, target, obstacle, N);
    for i = 1:N
        robot(i).v = -vmax;
        robot(i).dtheta = 0;
        robot(x2).frontRight = robot(x1).backRight;
        robot(x2).frontLeft = robot(x1).backLeft;
        robot(x2).y = robot(x1).y;
        robot(x3).frontRight = robot(x2).backRight;
        robot(x3).frontLeft = robot(x2).backLeft;
        robot(x3).y = robot(x2).y;
        robot(x4).frontRight = robot(x3).backRight;
        robot(x4).frontLeft = robot(x3).backLeft;
        robot(x4).y = robot(x3).y;
        robot(x5).frontRight = robot(x4).backRight;
        robot(x5).frontLeft = robot(x4).backLeft;
        robot(x5).y = robot(x4).y;
        robot(x6).frontRight = robot(x5).backRight;
        robot(x6).frontLeft = robot(x5).backLeft;
        robot(x6).y = robot(x5).y;
        robot(x7).frontRight = robot(x6).backRight;
        robot(x7).frontLeft = robot(x6).backLeft;
        robot(x7).y = robot(x6).y;
        robot(x8).frontRight = robot(x7).backRight;
        robot(x8).frontLeft = robot(x7).backLeft;
        robot(x8).y = robot(x7).y;
        robot(x9).frontRight = robot(x8).backRight;
        robot(x9).frontLeft = robot(x8).backLeft;
        robot(x9).y = robot(x8).y;
        robot(x10).frontRight = robot(x9).backRight;
        robot(x10).frontLeft = robot(x9).backLeft;
        robot(x10).y = robot(x9).y;
        if robot(i).assemble_number == 6
            if (cliff.distanceRight(i) > 0) || (cliff.distanceLeft(i) > 0)
                MoveBack_0 = false;
                MoveBack_1 = true;
            end
        end
        [robot] = calculatePoints(robot, i);
    end
    pause(dt);
    count = count + 1;
    F = getframe();
    writeVideo(writerObj, F);
end %x1 to x10 move backward 3
```

Script: The Rear Robot Disassembles

```
while (MoveBack_0 == false) && (MoveBack_1 == true)
    T = T + dt; tMoveBack = tMoveBack + dt;
     disassembleDistance(10) = calculateDistance([robot(x10).x, robot(x10).y], [robot(x9).x, robot(x9).y]);
    [robot, border, cliff, dynObstDistance, obstacle, target] = refreshBotData(robot, border, cliff, map, ...
        obstacle, target, dt, N);
    refreshPlotData(robot, map, target, obstacle, N);
    for i = 1:N
        if robot(i).assemble_number == 10
            robot(i).theta = dthetamax;
            robot(i).v = -vmax;
            if (border.distance(i).a < 15 || border.distance(i).b < 15 || border.distance(i).c < 15 || ...
                    border.distance(i).d < 15 || disassembleDistance(10) > 90)
                MoveBack_1 = false;
                MoveBack_2 = true;
            end
        else
            robot(i).dtheta = 0;
            robot(i).v = 0;
        end
        [robot] = calculatePoints(robot, i);
    end
    pause(dt);
    count = count + 1;
    F = getframe();
    writeVideo(writerObj, F);
end %x10 disassemble 3
```

B.2.1.2 Case B

Script: Set the Seed Robot Perpendicular to the Void

```
while (Search_Start == false) && (AsDis_01 == true)
    T = T + dt; tAsDis = tAsDis + dt;
    [robot, border, cliff, dynObstDistance, obstacle, target] = refreshBotData_c(robot, border, cliff, map, ...
        obstacle, target, dt, N);
    for i = 1:N
        cliff.distance(i) = calculateDistancePolygon(robot(i).x, robot(i).y, map.vertx, map.verty);
        cliff.distanceRight(i) = calculateDistancePolygon(robot(i).frontRight(1), robot(i).frontRight(2), ...
            map.vertx, map.verty);
        cliff.distanceLeft(i) = calculateDistancePolygon(robot(i).frontLeft(1), robot(i).frontLeft(2), map.vertx, ...
            map.verty);
    end
    refreshPlotData(robot, map, target, obstacle, N);
    for i = 1:N
        robot(i).theta = wrapToPi(robot(i).theta);
        if robot(i).assemble_number == 1
            if cliff.distanceRight(i) == cliff.distanceLeft(i)
                robot(i).v = 0;
                robot(i).dtheta = 0;
            elseif cliff.distanceRight(i) > cliff.distanceLeft(i)
                robot(i).v = -vmax;
                robot(i).dtheta = dthetamax;
                robot(i).v = 0;
            else
                robot(i).v = -vmax;
                robot(i).dtheta = -dthetamax;
                robot(i).v = 0;
            end
            seed_robot = i;
            x1 = i;
            xStart = robot(i).x;
            yStart = robot(i).y;
        elseif robot(i).assemble_number == -1
                robot(i).v = 0;
                robot(i).dtheta = 0;
        end
        [robot] = calculatePoints(robot, i);
        if robot(i).assemble_number == 1
            if (robot(i).theta < 0.062) && (robot(i).theta > -0.062)
                AsDis_01 = false;
                AsDis_02 = true;
            end
        end
    end
    pause(dt);
    count = count + 1;
    F = getframe();
    writeVideo(writerObj, F);
end
```

Script: Sorting Free Robot

```
while (AsDis_01 == false) && (AsDis_02 == true)
    T = T + dt; tAsDis = tAsDis + dt;
    for i = 1:N
        if robot(i).assemble_number ~= -1
            assembleDistance(i) = NaN;
        else
            assembleDistance(i) = calculateDistance([robot(i).x, robot(i).y], [robot(x1).x, robot(x1).y]);
        end
        sorting = sort(assembleDistance(:));
    end
    for i = 1:N
        if assembleDistance(i) == shortest
            robot(i).assemble_number = 2;
            x2 = i;
            phi = atan2((robot(x2).y - robot(x1).y),(robot(x2).x - (robot(x1).x - 12)));
            if (phi > 0) && (phi < pi)
                dphi = phi - pi;
            elseif (phi < 0) && (phi > -pi)
                dphi = phi + pi;
            end
            clear sorting;
            modeFreeStart = modeFreeStart-1;
            modeStructure = modeStructure+1;
            modeFreeOpposite = modeFreeOpposite+0;
            modeFreeBack = modeFreeBack+0;
            AsDis_02 = false;
            AsDis_03 = true;
        end
    end
    pause(dt);
    count = count + 1;
    F = getframe();
    writeVideo(writerObj, F);
end % sorting free robots, calls 2nd robot
```

Script: Calculate the Angle of Recruiter and Recruitee

```
while (AsDis_02 == false) && (AsDis_03 == true)
    T = T + dt; tAsDis = tAsDis + dt;
    [robot, border, cliff, dynObstDistance, obstacle, target] = refreshBotData_c(robot, border, cliff, map, ...
        obstacle, target, dt, N);
    refreshPlotData(robot, map, target, obstacle, N);
    for i = 1:N
        robot(i).theta = wrapToPi(robot(i).theta);
        if robot(i).assemble_number == -1
            robot(i).v = 0;
            robot(i).dtheta = 0;
        end
        robot(x1).v = 0;
        robot(x1).dtheta = 0;
        robot(x2).v = 0;
        robot(x2).dtheta = dthetamax;
        [robot] = calculatePoints(robot, i);
    end
    pause(dt);
    count = count + 1;
    F = getframe();
    writeVideo(writerObj, F);
    if (robot(x2).theta < dphi+0.062) && (robot(x2).theta > dphi-0.062)
        clear shortest;
        AsDis_03 = false;
        AsDis_04 = true;
    end
end % set 2nd robot's angle respect to 1st robot
```

Script: Recruitee Moves Closer to Recruiter

```
while (AsDis_03 == false) && (AsDis_04 == true)
    T = T + dt; tAsDis = tAsDis + dt;
    [robot, border, cliff, dynObstDistance, obstacle, target] = refreshBotData_c(robot, border, cliff, map, ...
        obstacle, target, dt, N);
    refreshPlotData(robot, map, target, obstacle, N);
    for i = 1:N
        robot(i).theta = wrapToPi(robot(i).theta);
        if robot(i).assemble_number == -1
            robot(i).v = 0;
            robot(i).dtheta = 0;
        end
        robot(x1).v = 0;
        robot(x1).dtheta = 0;
        robot(x2).theta = dphi;
        robot(x2).v = vmax;
        botsDistance = calculateDistance([robot(x2).x, robot(x2).y], [robot(x1).x, robot(x1).y]);
        [robot] = calculatePoints(robot, i);
    end
    pause(dt);
    count = count + 1;
    F = getframe();
    writeVideo(writerObj, F);
    if (botsDistance < 12.5) && (botsDistance > 11.5)
        clear shortest;
        AsDis_04 = false;
        AsDis_05 = true;
    end
end % 2nd robot gets close to 1st robot
```

Script: Recruitee and Recruiter Coupled

```
while (AsDis_04 == false) && (AsDis_05 == true)
    T = T + dt; tAsDis = tAsDis + dt;
    [robot, border, cliff, dynObstDistance, obstacle, target] = refreshBotData_c(robot, border, cliff, map, ...
        obstacle, target, dt, N);
    refreshPlotData(robot, map, target, obstacle, N);
    for i = 1:N
        robot(i).theta = wrapToPi(robot(i).theta);
        if robot(i).assemble_number == -1
            robot(i).v = 0;
            robot(i).dtheta = 0;
        end
        robot(x1).v = 0;
        robot(x1).dtheta = 0;
        if (robot(x2).theta > 0) && (robot(x2).theta < pi)
            robot(x2).v = 0;
            robot(x2).dtheta = -dthetamax;
        elseif (robot(x2).theta < 0) && (robot(x2).theta > -pi)
            robot(x2).v = 0;
            robot(x2).dtheta = dthetamax;
        end
        [robot] = calculatePoints(robot, i);
    end
    pause(dt);
    count = count + 1;
    F = getframe();
    writeVideo(writerObj, F);
    if (robot(x2).theta < 0.062) && (robot(x2).theta > -0.062)
        robot(x2).x = robot(x1).x - 12;
        robot(x2).y = robot(x1).y;
        robot(x2).dtheta = 0;
        robot(x2).v = 0;
        [robot] = calculatePoints(robot, i);
        AsDis_05 = false;
        AsDis_06 = true;
    end
end % 2nd robot sticks to 1st robot
```

Script: The Structure Moves Forward

```
while (AsDis_09 == false) && (AsDis_10 == true)
    T = T + dt; tAsDis = tAsDis + dt;
    [robot, border, cliff, dynObstDistance, obstacle, target] = refreshBotData_c(robot, border, cliff, map, ...
        obstacle, target, dt, N);
    refreshPlotData(robot, map, target, obstacle, N);
    for i = 1:N
        if robot(i).assemble_number == -1
            robot(i).v = 0;
            robot(i).dtheta = 0;
        end
        if robot(i).assemble_number ~= -1
            robot(i).dtheta = 0;
            robot(i).v = vmax;
            robot(x2).x = robot(x1).x - 12;
            robot(x2).y = robot(x1).y;
            robot(x3).x = robot(x2).x - 12;
            robot(x3).y = robot(x2).y;
        end
        if robot(i).assemble_number == 2
            if (cliff.distanceRight(i) < 0) && (cliff.distanceLeft(i) < 0)
                modeFreeStart = modeFreeStart-0;
                modeStructure = modeStructure+0;
                modeFreeOpposite = modeFreeOpposite+0;
                modeFreeBack = modeFreeBack+0;
                AsDis_10 = false;
                AsDis_11 = true;
            end
        end
        [robot] = calculatePoints(robot, i);
    end
    pause(dt);
    count = count + 1;
    F = getframe();
    writeVideo(writerObj, F);
end % 1st-2nd-3rd robots move towards the gap; 1st robot is over the gap
```

Script: The Front Robot Disassembles

```
while (AsDis_72a == false) && (AsDis_73a == true)
    T = T + dt; tAsDis = tAsDis + dt;
    [robot, border, cliff, dynObstDistance, obstacle, target] = refreshBotData_c(robot, border, cliff, map, ...
        obstacle, target, dt, N);
    refreshPlotData(robot, map, target, obstacle, N);
    for i = 1:N
        if robot(i).assemble_number == -2
            [robot, r] = searchDisassemble(robot, i, vmax, dthetamax, p_r, p_l, border, cliff, obstacle, ...
                dynObstDistance);
        elseif robot(i).assemble_number == 1
            robot(i).v = vmax; robot(i).dtheta = 0;
        else
            robot(i).v = 0; robot(i).dtheta = 0;
        end
        [robot] = calculatePoints(robot, i);
    end
    if (disassembleDistance > 45)
        robot(x1).assemble_number = -2; clear x1; x1 = x2;
        robot(x1).assemble_number = 1; x2 = x3;
        robot(x2).assemble_number = 2; x3 = x4;
        robot(x3).assemble_number = 3; x4 = x5;
        robot(x4).assemble_number = 4; x5 = x6;
        robot(x5).assemble_number = 5; x6 = x7;
        robot(x6).assemble_number = 6; x7 = x8;
        robot(x7).assemble_number = 7; x8 = x9;
        robot(x8).assemble_number = 8; x9 = x10;
        robot(x9).assemble_number = 9; x10 = x11;
        robot(x10).assemble_number = 10; x11 = x12;
        robot(x11).assemble_number = 11; x12 = x13;
        robot(x12).assemble_number = 12; x13 = x14;
        robot(x13).assemble_number = 13; x14 = x15;
        robot(x14).assemble_number = 14; clear x15;
        modeFreeStart = modeFreeStart-0;
        modeStructure = modeStructure-1;
        modeFreeOpposite = modeFreeOpposite+1;
        modeFreeBack = modeFreeBack+0;
        AsDis_73a = false;
        AsDis_74a = true;
    end
    pause(dt);
    count = count + 1;
    F = getframe();
    writeVideo(writerObj, F);
end % 1st robot disassembles then turns to search mode and assemble number becomes -2; 2nd to 15th
robots become 1st to 14th robots
```

Script: The Structure Moves Backward

```
while ((AsDis_38 == false) && (MoBack_04 == true)) || ((BaDis_09 == false) && (MoBack_04 == true))
    T = T + dt; tAsDis = tAsDis + dt;
    [robot, border, cliff, dynObstDistance, obstacle, target] = refreshBotData_c(robot, border, cliff, map, ...
        obstacle, target, dt, N);
    refreshPlotData(robot, map, target, obstacle, N);
    for i = 1:N
        if robot(i).assemble_number == -3
        [robot, r] = searchDisassemble(robot, i, vmax, dthetamax, p_r, p_l. border, cliff, obstacle, ...
            dynObstDistance);
        elseif robot(i).assemble_number ~= -3
            robot(i).dtheta = 0;
            robot(i).v = -vmax;
            robot(x2).x = robot(x1).x - 12; robot(x2).y = robot(x1).y;
            robot(x3).x = robot(x2).x - 12; robot(x3).y = robot(x2).y;
            robot(x4).x = robot(x3).x - 12; robot(x4).y = robot(x3).y;
            robot(x5).x = robot(x4).x - 12; robot(x5).y = robot(x4).y;
            robot(x6).x = robot(x5).x - 12; robot(x6).y = robot(x5).y;
            robot(x7).x = robot(x6).x - 12; robot(x7).y = robot(x6).y;
            robot(x8).x = robot(x7).x - 12; robot(x8).y = robot(x7).y;
            robot(x9).x = robot(x8).x - 12; robot(x9).y = robot(x8).y;
        end
        if robot(i).assemble_number == 4
            if (cliff.distanceRight(i) > 0) && (cliff.distanceLeft(i) > 0)
                modeFreeStart = modeFreeStart-0;
                modeStructure = modeStructure+0;
                modeFreeOpposite = modeFreeOpposite+0;
                modeFreeBack = modeFreeBack+0;
                MoBack_04 = false;
                BaDis_08 = true;
            end
        end
        [robot] = calculatePoints(robot, i);
    end
    pause(dt);
    count = count + 1;
    F = getframe();
    writeVideo(writerObj, F);
end % 1st-2nd-3rd-4th-5th-6th-7th-8th-9th robots move backward; 1st-2nd-3rd robot are over the gap
```

Script: The Rear Robot Disassembles

```
while (MoBack_04 == false) && (BaDis_08 == true)
    T = T + dt; tBaDis = tBaDis + dt;
    disassembleDistance = calculateDistance([robot(x9).x, robot(x9).y], [robot(x8).x, robot(x8).y]);
    [robot, border, cliff, dynObstDistance, obstacle, target] = refreshBotData_c(robot, border, cliff, map, ...
        obstacle, target, dt, N);
    refreshPlotData(robot, map, target, obstacle, N);
    for i = 1:N
        if robot(i).assemble_number == 9
            robot(i).v = 0; robot(i).dtheta = dthetamax;
            if (robot(i).theta < pi+0.062) && (robot(i).theta > pi-0.062)
                robot(i).v = vmax; robot(i).dtheta = 0;
            end
        else
            robot(i).dtheta = 0; robot(i).v = 0;
        end
        [robot] = calculatePoints(robot, i);
    end
    if (disassembleDistance > 45)
        robot(x9).assemble_number = -3;
        clear x9;
        modeFreeStart = modeFreeStart-0;
        modeStructure = modeStructure-1;
        modeFreeOpposite = modeFreeOpposite+0;
        modeFreeBack = modeFreeBack+1;
        BaDis_08 = false;
        BaDis_07 = true;
    end
    pause(dt);
    count = count + 1;
    F = getframe();
    writeVideo(writerObj, F);
end % 9th robot disassembles backward
```

B.2.1.3 Cases C and D

Script: Set the Seed Robot Perpendicular to the Void

```
while (SearchStart == false) && (Forward1 == true)
    T = T + dt; tForward = tForward + dt;
    [robot, border, cliff, dynObstDistance, target] = refreshBotDataComb_2robot, border, cliff, map, target, dt, N;
    for i = 1:N
        cliff.distance(i) = calcDistancePolyComb_2(robot(i).x, robot(i).y, map.vertx, map.verty);
        cliff.distanceRight(i) = calcDistancePolyComb_2(robot(i).frontRight(1), robot(i).frontRight(2), ...
            map.vertx, map.verty);
        cliff.distanceLeft(i) = calcDistancePolyComb_2(robot(i).frontLeft(1), robot(i).frontLeft(2), map.vertx, ...
            map.verty);
    end
    refreshPlotDataComb_2(robot, map, target, N);
    for i = 1:N
        robot(i).theta = wrapToPi(robot(i).theta);
        if robot(i).structure == 1
            if cliff.distanceRight(i) == cliff.distanceLeft(i)
                robot(i).v = 0; robot(i).dtheta = 0;
            elseif cliff.distanceRight(i) > cliff.distanceLeft(i)
                robot(i).v = -vmax; robot(i).dtheta = dthetamax; robot(i).v = 0;
            else
                robot(i).v = -vmax; robot(i).dtheta = -dthetamax; robot(i).v = 0;
            end
            seedRobot = i; x1 = i; xStart = robot(i).x; yStart = robot(i).y;
        elseif robot(i).structure == -1
            [robot, r] = searchAssembleComb_2(robot, i, vmax, dthetamax, probr, probl, border, cliff, ...
                dynObstDistance);
        end
    [robot] = calcPointsComb_2(robot, i);
    end
    pause(dt);
    count = count + 1;
    %F = getframe();
    %writeVideo(writerObj, F);
end % 1st robot sets orthogonal
```

Script: Sorting Free Robot

```
while (Forward1 == false) && (Sorting1 == true)
    for i = 1:N
        if robot(i).structure ~= -1
            assembleDistance(i) = NaN;
        else
            assembleDistance(i) = calcDistanceComb_2([robot(i).x, robot(i).y], [robot(x1).x, robot(x1).y]);
        end
        sorting = sort(assembleDistance(:));
    end
    shortest = sorting(1);
    for i = 1:N
        if assembleDistance(i) == shortest
            robot(i).structure = 2;
            x2 = i;
            phi = atan2((robot(x2).y - robot(x1).y),(robot(x2).x - (robot(x1).x - 10)));
            if (phi > 0) && (phi < pi)
                dphi = phi - pi;
            elseif (phi < 0) && (phi > -pi)
                dphi = phi + pi;
            end
            clear sorting;
            modeStart = modeStart-1;
            modeStructure = modeStructure+1;
            modeOpposite = modeOpposite+0;
            modeBack = modeBack+0;
            Sorting1 = false;
            Forward2 = true;
        end
    end
    pause(dt);
    count = count + 1;
    %F = getframe();
    %writeVideo(writerObj, F);
end % sorting free robots, calls 2nd robot
```

Script: Calculate the Angle of Recruiter and Recruitee

```
while (Sorting1 == false) && (Forward2 == true)
    T = T + dt; tForward = tForward + dt;
    [robot, border, cliff, dynObstDistance, target] = refreshBotDataComb_2robot, border, cliff, map, target, dt, N;
    refreshPlotDataComb_2(robot, map, target, N);
    for i = 1:N
        robot(i).theta = wrapToPi(robot(i).theta);
        if robot(i).structure == 2
            robot(i).v = 0; robot(i).dtheta = 5*dthetamax;
        elseif robot(i).structure == -1
            [robot, r] = serachAssembleComb_2(robot, i, vmax, dthetamax, probr, probl, border, cliff, ...
                dynObstDistance);
        else
            robot(i).v = 0; robot(i).dtheta = 0;
        end
        [robot] = calcPointsComb_2(robot, i);
    end
    pause(dt);
    count = count + 1;
    %F = getframe();
    %writeVideo(writerObj, F);
    if (robot(x2).theta < dphi+0.062) && (robot(x2).theta > dphi-0.062)
        clear shortest;
        Forward2 = false;
        Forward3 = true;
    end
end % set 2nd robot's angle respect to 1st robot
```

Script: Recruitee Moves Closer to Recruiter

```
while (Forward2 == false) && (Forward3 == true)
    T = T + dt; tForward = tForward + dt;
    [robot, border, cliff, dynObstDistance, target] = refreshBotDataComb_2robot, border, cliff, map, target, dt, N;
    refreshPlotDataComb_2(robot, map, target, N);
    for i = 1:N
        robot(i).theta = wrapToPi(robot(i).theta);
        if robot(i).structure == 2
            robot(i).theta = dphi; robot(i).v = 3*vmax;
        elseif robot(i).structure == -1
            [robot, r] = searchAssembleComb_2(robot, i, vmax, dthetamax, probr, probl, border, cliff, ...
                dynObstDistance);
        else
            robot(i).v = 0; robot(i).dtheta = 0;
        end
        botsDistance = calcDistanceComb_2([robot(x2).x, robot(x2).y], [robot(x1).x, robot(x1).y]);
        [robot] = calcPointsComb_2(robot, i);
    end
    pause(dt);
    count = count + 1;
    %F = getframe();
    %writeVideo(writerObj, F);
    if (botsDistance < 10.5) && (botsDistance > 9.5)
        Forward3 = false;
        Forward4 = true;
    end
end % 2nd robot gets close to 1st robot
```

Script: Recruitee and Recruiter Coupled

```
while (Forward3 == false) && (Forward4 == true)
    T = T + dt; tForward = tForward + dt;
    [robot, border, cliff, dynObstDistance, target] = refreshBotDataComb_2robot, border, cliff, map, target, dt, N;
    refreshPlotDataComb_2(robot, map, target, N);
    for i = 1:N
        robot(i).theta = wrapToPi(robot(i).theta);          if robot(i).structure == 2
            if (robot(i).theta > 0) && (robot(i).theta < pi)
                robot(i).v = 0; robot(i).dtheta = -5*dthetamax;
            elseif (robot(i).theta < 0) && (robot(i).theta > -pi)
                robot(i).v = 0; robot(i).dtheta = 5*dthetamax;
            end
        elseif robot(i).structure == -1
            [robot, r] = searchAssembleComb_2(robot, i vmax, dthetamax, probr, probl, border, cliff, ...
                dynObstDostance);
        else
            robot(i).v = 0; robot(i).dtheta = 0;
        end
        [robot] = calcPointsComb_2(robot, i);
    end
    pause(dt);
    count = count + 1;
    %F = getframe();
    %writeVideo(writerObj, F);
    if (robot(x2).theta < 0.062) && (robot(x2).theta > -0.062)
        robot(x2).x = robot(x1).x - 10;
        robot(x2).y = robot(x1).y;
        robot(x2).dtheta = 0;
        robot(x2).v = 0;
        [robot] = calcPointsComb_2(robot, i);
        Forward4 = false;
        Sorting2 = true;
    end
end % 2nd robot sticks to 1st robot
```

Script: Option to Moves Forward or Recruit Free Robot

```
while ((Option0 == false) && (Option1a == true)) || ((Forward13 == false) && (Option1a == true))
    T = T + dt; tForward = tForward + dt;
    [robot, border, cliff, dynObstDistance, target] = refreshBotDataComb_2robot, border, cliff, map, target, dt, N;
    refreshPlotDataComb_2(robot, map, target, N);
    for i = 1:N
        if robot(i).structure == -2
            [robot, r] = searchDisassembleComb_2(robot, i, vmax, dthetamax, probr, probl, border, cliff, ...
                dynObstDistance);
        elseif robot(i).structure == -1
            [robot, r] = searchAssembleComb_2(robot, i vmax, dthetamax, probr, probl, border, cliff, ...
                dynObstDistance);
        elseif (robot(i).structure ~= -1) && (robot(i).structure ~= -2)
            robot(i).dtheta = 0; robot(i).v = vmax;
            robot(x2).x = robot(x1).x - 10; robot(x2).y = robot(x1).y;
            robot(x3).x = robot(x2).x - 10; robot(x3).y = robot(x2).y;
        end
        if robot(i).structure == 3
            if (cliff.distanceRight(i) < 0) && (cliff.distanceLeft(i) < 0)
                modeStart = modeStart-0;
                modeStructure = modeStructure+0;
                modeOpposite = modeOpposite+0;
                modeBack = modeBack+0;
                if modeStart == 0
                    Option1a = false;
                    Forward9a = true;
                elseif modeStart > 0
                    Option1a = false;
                    Sorting3a = true;
                end
            end
        end
        [robot] = calcPointsComb_2(robot, i);
    end
    pause(dt);
    count = count + 1;
    %F = getframe();
    %writeVideo(writerObj, F);
end % option to move forward or recruit; the structure moves forward and 2nd robot is over the gap
if (Option1a == false) && (Forward9a == true)
    buildForward1a_2;
elseif (Option1a == false) && (Sorting3a == true)
    buildForward1b_2;
end
```

Script: The Structure Moves Forward

```
while ((Option1a == false) && (Forward9a == true)) || ((Forward23a == false) && (Forward9a == true))
    T = T + dt; tForward = tForward + dt;
    [robot, border, cliff, dynObstDistance, target] = refreshBotDataComb_2robot, border, cliff, map, target, dt, N;
    refreshPlotDataComb_2(robot, map, target, N);
    for i = 1:N
        if robot(i).structure == -2
            [robot, r] = searcjDisassembleComb_2(robot, i, vmax, dthetamax, probr, probl, border, cliff, ...
dynObstDistance);
        elseif robot(i).structure ~= -2
            robot(i).dtheta = 0; robot(i).v = vmax;
            robot(x2).x = robot(x1).x - 10; robot(x2).y = robot(x1).y;
            robot(x3).x = robot(x2).x - 10; robot(x3).y = robot(x2).y;
        end
        if robot(i).structure == 1
            if (cliff.distanceBackRight(i) > 20) && (cliff.distanceBackLeft(i) > 20)
                modeStart = modeStart-0;
                modeStructure = modeStructure+0;
                modeOpposite = modeOpposite+0;
                modeBack = modeBack+0;
                Forward9a = false;
                Forward10a = true;
            end
        end
        [robot] =calPointsComb_2(robot, i);
    end
    pause(dt);
    count = count + 1;
    %F = getframe();
    %writeVideo(writerObj, F);
end % 1st-2nd-3rd robots move towards the gap until they reach the opposite
```

Script: The Front Robot Disassembles

```
while (Forward9a == false) && (Forward10a == true)
    T = T + dt; tForward = tForward + dt;
    disassembleDistance = calcDistanceComb_2([robot(x1).x, robot(x1).y], [robot(x2).x, robot(x2).y]);
    [robot, border, cliff, dynObstDistance, target] = refreshBotDataComb_2robot, border, cliff, map, target, dt, N;
    refreshPlotDataComb_2(robot, map, target, N);
    for i = 1:N
        if robot(i).structure == -2
            [robot, r] = searchDisassembleComb_2(robot, i, vmax, dthetamax, probr, probl, border, cliff, ...
                dybObstDostance);
        elseif robot(i).structure == 1
            robot(i).v = vmax; robot(i).dtheta = 0;
        else
            robot(i).dtheta = 0; robot(i).v = 0;
        end
        [robot] = calcPointsComb_2(robot, i);
    end
    if (disassembleDistance > 30)
        robot(x1).structure = -2;
        clear x1; x1 = x2;
        robot(x1).structure = 1;
        x2 = x3;
        robot(x2).structure = 2;
        x3 = 0;
        modeStart = modeStart-0;
        modeStructure = modeStructure-1;
        modeOpposite = modeOpposite+1;
        modeBack = modeBack+0;
        Forward10a = false;
        Forward11a = true;
    end
    pause(dt);
    count = count + 1;
    %F = getframe();
    %writeVideo(writerObj, F);
end % 1st robot disassembles then turns to search mode and assemble number becomes -2; 2nd robot becomes 1st
robot and 3rd robot becomes 2nd robot
```

Script: The Structure Moves Backward

```
while ((Option1b == false) && (Backward1 == true)) || ((Backward4 == false) && (Backward1 == true))
    T = T + dt; tForward = tForward + dt;
    [robot, border, cliff, dynObstDistance, target] = refreshBotDataComb_2robot, border, cliff, map, target, dt, N;
    refreshPlotDataComb_2(robot, map, target, N);
    for i = 1:N
        if robot(i).structure == -3
            [robot, r] = searchDisassembleComb_2(robot, i, vmax, dthetamax, probr, probl, broder, cliff, ...
                dynObstDistance);
        elseif robot(i).structure ~= -3
            robot(i).dtheta = 0; robot(i).v = -vmax;
            robot(x2).x = robot(x1).x - 10; robot(x2).y = robot(x1).y;
            robot(x3).x = robot(x2).x - 10; robot(x3).y = robot(x2).y;
        end
        if robot(i).structure == 1
            if (cliff.distanceRight(i) > 0) && (cliff.distanceLeft(i) > 0)
                modeStart = modeStart-0;
                modeStructure = modeStructure+0;
                modeOpposite = modeOpposite+0;
                modeBack = modeBack+0;
                Backward1 = false;
                Backward2 = true;
            end
        end
        [robot] = calcPointsComb_2(robot, i);
    end
    pause(dt);
    count = count + 1;
    %F = getframe();
    %writeVideo(writerObj, F);
end % 1st-2nd-3rd robots move backward; the structure at the initial zone
```

Script: The Rear Robot Disassembles

```
while (Backward1 == false) && (Backward2 == true)
    T = T + dt; tForward = tForward + dt;
    disassembleDistance = calcDistanceComb_2([robot(x1).x, robot(x1).y], [robot(x2).x, robot(x2).y]);
    [robot, border, cliff, dynObstDistance, target] = refreshBotDataComb_2robot, border, cliff, map, target, dt, N;
    refreshPlotDataComb_2(robot, map, target, N);
    for i = 1:N
        if robot(i).structure == -3
            [robot, r] = searchDisassembleComb_2(robot, i, vmax, dthetamax, probr, probl, border, cliff, ...
                dynObstDistance);
        elseif robot(i).structure == 3
            robot(i).v = 0;
                robot(i).dtheta = 5*dthetamax;
            if (robot(i).theta < pi+0.062) && (robot(i).theta > pi-0.062)
                robot(i).v = 3*vmax; robot(i).dtheta = 0;
            end
        else
            robot(i).dtheta = 0; robot(i).v = 0;
        end
        [robot] = calcPoints_2(robot, i);
    end
    if (disassembleDistance > 30)
        robot(x3).structure = -3;
        clear x3;
        modeStart = modeStart-0;
        modeStructure = modeStructure-1;
        modeOpposite = modeOpposite+0;
        modeBack = modeBack+1;
        Backward2 = false;
        Backward3 = true;
    end
    pause(dt);
    count = count + 1;
    %F = getframe();
    %writeVideo(writerObj, F);
end % 3rd robot disassembles backward
```

B.2.2 Large Void Crossing

Scripts for the first line of the structure for large void crossing is the same as scripts for cases C and D in simple line structure for small void crossing. Therefore, scripts written in this section are examples of how to form the second line and how front robots disassemble at the target zone.

Script: Sorting Free Robot to Join the Second Line

```
while (Option5 == false) && (SortCoreR1 == true)
    for i = 1:N
        if robot(i).coreL ~= 0
            assembleDistance(i) = NaN;
        elseif robot(i).coreR ~= 0
            assembleDistance(i) = NaN;
        else
            assembleDistance(i) = calcDistance([robot(i).x, robot(i).y], [robot(cL11).x, robot(cL11).y]);
        end
        sorting = sort(assembleDistance(:));
    end
    shortest = sorting(1);
    for i = 1:N
        if assembleDistance(i) == shortest
            robot(i).coreR = 1;
            robot(i).start = 0;
            cR1 = i;
            phi = atan2((robot(cR1).y - (robot(cL11).y - 10)),(robot(cR1).x - (robot(cL11).x - 10)));
            if (phi > 0) && (phi < pi)
                dphi = phi - pi;
            elseif (phi < 0) && (phi > -pi)
                dphi = phi + pi;
            end
            clear sorting;
            modeStart = modeStart-1;
            modeCoreR = modeCoreR+1;
            SortCoreR1 = false;
            BuildCoreR1a = true;
        end
    end
    pause(dt);
    count = count + 1;
    %F = getframe();
    %writeVideo(writerObj, F);
end % sorting R1
```

Script: Calculate the Angle of Recruiter (Left Side Line) and Recruite (Right Side Line)

```
while (SortCoreR1 == false) && (BuildCoreR1a == true)
    T = T + dt; tAsm = tAsm + dt; tConstruct = tConstruct + dt;
    [robot, border, cliff, dynObstDistance, target] = refreshBotData(robot, border, cliff, map, target, dt, N);
    refreshPlotData(robot, map, target, N);
    for i = 1:N
        robot(i).theta = wrapToPi(robot(i).theta);
        if robot(i).coreR == 1
            robot(cR1).v = 0; robot(cR1).dtheta = 5*dthetamax;
        elseif robot(i).start == 1
            [robot, r] = searchAssemble(robot, i, vmax, dthetamax, probr, probl, border, cliff, dynObstDistance);
        else
            robot(i).v = 0; robot(i).dtheta = 0;
        end
        [robot] = calcPoints(robot, i);
    end
    pause(dt);
    count = count + 1;
    %F = getframe();
    %writeVideo(writerObj, F);
    if (robot(cR1).theta < dphi+0.062) && (robot(cR1).theta > dphi-0.062)
        clear shortest;
        BuildCoreR1a = false;
        BuildCoreR1b = true;
    end
end % setting R1's angle to L11
```

Script: Recruitee Moves Closer to Recruiter

```
while (BuildCoreR1a == false) && (BuildCoreR1b == true)
    T = T + dt; tAsm = tAsm + dt; tConstruct = tConstruct + dt;
    [robot, border, cliff, dynObstDistance, target] = refreshBotData(robot, border, cliff, map, target, dt, N);
    refreshPlotData(robot, map, target, N);
    for i = 1:N
        robot(i).theta = wrapToPi(robot(i).theta);
        if robot(i).coreR == 1
            robot(cR1).theta = dphi; robot(cR1).v = 3*vmax;
        elseif robot(i).start == 1
            [robot, r] = searchAssemble(robot, i, vmax, dthetamax, probr, probl, border, cliff, dynObstDistance);
        else
            robot(i).v = 0; robot(i).dtheta = 0;
        end
        botsDistance = calcDistance([robot(cR1).x, robot(cR1).y], [robot(cL11).x, robot(cL11).y]);
        [robot] = calcPoints(robot, i);
    end
    pause(dt);
    count = count + 1;
    %F = getframe();
    %writeVideo(writerObj, F);
    if (botsDistance < 14.65) && (botsDistance > 13.35)
        BuildCoreR1b = false;
        BuildCoreR1c = true;
    end
end % R1 gets close to L11
```

Script: Recruitee Touchs the Corner of the Recruiter's Rear

```
while (BuildCoreR1b == false) && (BuildCoreR1c == true)
    T = T + dt; tAsm = tAsm + dt; tConstruct = tConstruct + dt;
    [robot, border, cliff, dynObstDistance, target] = refreshBotData(robot, border, cliff, map, target, dt, N);
    refreshPlotData(robot, map, target, N);
    for i = 1:N
        robot(i).theta = wrapToPi(robot(i).theta);
        if robot(i).coreR == 1
            if (robot(cR1).theta > 0) && (robot(cR1).theta < pi)
        robot(cR1).v = 0; robot(cR1).dtheta = -5*dthetamax;
            elseif (robot(cR1).theta < 0) && (robot(cR1).theta > -pi)
        robot(cR1).v = 0; robot(cR1).dtheta = 5*dthetamax;
            end
        elseif robot(i).start == 1
            [robot, r] = searchAssemble(robot, i, vmax, dthetamax, probr, probl, border, cliff, dynObstDistance);
        else
            robot(i).v = 0; robot(i).dtheta = 0;
        end
        [robot] = calcPoints(robot, i);
    end
    pause(dt);
    count = count + 1;
    %F = getframe();
    %writeVideo(writerObj, F);
    if (robot(cR1).theta < 0.062) && (robot(cR1).theta > -0.062)
        robot(cR1).x = robot(cL11).x - 10;
        robot(cR1).y = robot(cL11).y - 10;
        robot(cR1).dtheta = 0;
        robot(cR1).v = 0;
        [robot] = calcPoints(robot, i);
        BuildCoreR1c = false;
        ForwardR1 = true;
    end
end % R1's front corner touches L11's back corner
```

Script: Recruitee and Recruiter Stand Side-by-Side

```
while (BuildCoreR1c == false) && (ForwardR1 == true)
    T = T + dt; tAsm = tAsm + dt; tConstruct = tConstruct + dt;
    [robot, border, cliff, dynObstDistance, target] = refreshBotData(robot, border, cliff, map, target, dt, N);
    refreshPlotData(robot, map, target, N);
    for i = 1:N
        if robot(i).start == 1
            [robot, r] = searchAssemble(robot, i, vmax, dthetamax, probr, probl, border, cliff, dynObstDistance);
        elseif robot(i).coreR ~= 0
            robot(i).dtheta = 0; robot(i).v = vmax; robot(cR1).y = robot(cL11).y - 10;
        else
            robot(i).v = 0; robot(i).dtheta = 0;
        end
        if robot(i).coreR == 1
            if robot(cR1).x > robot(cL11).x
                modeStart = modeStart-0;
                modeCoreR = modeCoreR+0;
                ForwardR1 = false;
                SortCoreR2 = true;
            end
        end
        [robot] = calcPoints(robot, i);
    end
    pause(dt);
    count = count + 1;
    %F = getframe();
    %writeVideo(writerObj, F);
end % set R1 side-by-side with L11
```

Script: Front Robot of the Right Side Line Disassembles

```
while (Option6o6 == false) && (Construct6a1 == true)
    T = T + dt; tDisassembly = tDisassembly + dt; tConstruct = tConstruct + dt;
    [robot, border, cliff, dynObstDistance, target] = refreshBotData(robot, border, cliff, map, target, dt, N);
    refreshPlotData(robot, map, target, N);
    for i = 1:N
        if robot(i).opposite == 1
            [robot, r] = searchAssemble(robot, i, vmax, dthetamax, probr, probl, border, cliff, dynObstDistance);
        elseif robot(i).opposite ~= 1
            robot(i).dtheta = 0; robot(i).v = vmax;
            robot(cR1).x = robot(cL1).x; robot(cR1).y = robot(cL1).y - 10;
            robot(cL2).x = robot(cL1).x - 10; robot(cL2).y = robot(cL1).y;
            robot(cR2).x = robot(cR1).x - 10; robot(cR2).y = robot(cR1).y;
            robot(cL3).x = robot(cL2).x - 10; robot(cL3).y = robot(cL2).y;
            robot(cR3).x = robot(cR2).x - 10; robot(cR3).y = robot(cR2).y;
            robot(cL4).x = robot(cL3).x - 10; robot(cL4).y = robot(cL3).y;
            robot(cR4).x = robot(cR3).x - 10; robot(cR4).y = robot(cR3).y;
            robot(cL5).x = robot(cL4).x - 10; robot(cL5).y = robot(cL4).y;
            robot(cR5).x = robot(cR4).x - 10; robot(cR5).y = robot(cR4).y;
            robot(cL6).x = robot(cL5).x - 10; robot(cL6).y = robot(cL5).y;
            robot(cR6).x = robot(cR5).x - 10; robot(cR6).y = robot(cR5).y;
            robot(cL7).x = robot(cL6).x - 10; robot(cL7).y = robot(cL6).y;
            robot(cR7).x = robot(cR6).x - 10; robot(cR7).y = robot(cR6).y;
            robot(cL8).x = robot(cL7).x - 10; robot(cL8).y = robot(cL7).y;
            robot(cR8).x = robot(cR7).x - 10; robot(cR8).y = robot(cR7).y;
            robot(cL9).x = robot(cL8).x - 10; robot(cL9).y = robot(cL8).y;
            robot(cR9).x = robot(cR8).x - 10; robot(cR9).y = robot(cR8).y;
            robot(cL10).x = robot(cL9).x - 10; robot(cL10).y = robot(cL9).y;
            robot(cR10).x = robot(cR9).x - 10; robot(cR10).y = robot(cR9).y;
            robot(cL11).x = robot(cL10).x - 10; robot(cL11).y = robot(cL10).y;
            robot(cR11).x = robot(cR10).x - 10; robot(cR11).y = robot(cR10).y;
            robot(cL12).x = robot(cL11).x - 10; robot(cL12).y = robot(cL11).y;
            robot(cR12).x = robot(cR11).x - 10; robot(cR12).y = robot(cR11).y;
            robot(cL13).x = robot(cL12).x - 10; robot(cL13).y = robot(cL12).y;
            robot(cR13).x = robot(cR12).x - 10; robot(cR13).y = robot(cR12).y;
            robot(cL14).x = robot(cL13).x - 10; robot(cL14).y = robot(cL13).y;
            robot(cR14).x = robot(cR13).x - 10; robot(cR14).y = robot(cR13).y;
            robot(cL15).x = robot(cL14).x - 10; robot(cL15).y = robot(cL14).y;
            robot(cR15).x = robot(cR14).x - 10; robot(cR15).y = robot(cR14).y;
            robot(cL16).x = robot(cL15).x - 10; robot(cL16).y = robot(cL15).y;
            robot(cR16).x = robot(cR15).x - 10; robot(cR16).y = robot(cR15).y;
            robot(cL17).x = robot(cL16).x - 10; robot(cL17).y = robot(cL16).y;
            robot(cR17).x = robot(cR16).x - 10; robot(cR17).y = robot(cR16).y;
            robot(cL18).x = robot(cL17).x - 10; robot(cL18).y = robot(cL17).y;
            robot(cR18).x = robot(cR17).x - 10; robot(cR18).y = robot(cR17).y;
        end
        if robot(i).coreL == 10
            if (cliff.distanceBackRight(i) > 0) && (cliff.distanceBackLeft(i) > 0)
                modeStart = modeStart-0;
                modeCoreL = modeCoreL+0;
                modeCoreR = modeCoreR+0;
                Construct6a1 = false;
                Construct6a2 = true;
            end
        end
        [robot] = calcPoints(robot, i);
    end
    pause(dt);
    count = count + 1;
    %F = getframe();
    %writeVideo(writerObj, F);
end % cL1-cL18 and cR1-cR18 move forward; cL11-cL16 and cR11-cR16 are over the gap
```

Script: Front Robot of the Right Side Line Disassembles

```
while (Construct6a1 == false) && (Construct6a2 == true)
    T = T + dt; tDisassembly = tDisassembly + dt; tConstruct = tConstruct + dt;
    disassembleDistance = calcDistance([robot(cR1).x, robot(cR1).y], [robot(cR2).x, robot(cR2).y]);
    [robot, border, cliff, dynObstDistance, target] = refreshBotData(robot, border, cliff, map, target, dt, N);
    refreshPlotData(robot, map, target, N);
    for i = 1:N
        if robot(i).opposite == 1
            [robot, r] = searchAssemble(robot, i, vmax, dthetamax, probr, probl, border, cliff, dynObstDistance);
        elseif robot(i).coreR == 1
            robot(i).v = vmax; robot(i).dtheta = 0;
        else
            robot(i).dtheta = 0; robot(i).v = 0;
        end
        [robot] = calcPoints(robot, i);
    end
    if (disassembleDistance > 40)
        robot(cR1).opposite = 1;
        robot(cR1).coreR = 0;
        clear cR1;
        cR1 = cR2; robot(cR1).coreR = 1;
        cR2 = cR3; robot(cR2).coreR = 2;
        cR3 = cR4; robot(cR3).coreR = 3;
        cR4 = cR5; robot(cR4).coreR = 4;
        cR5 = cR6; robot(cR5).coreR = 5;
        cR6 = cR7; robot(cR6).coreR = 6;
        cR7 = cR8; robot(cR7).coreR = 7;
        cR8 = cR9; robot(cR8).coreR = 8;
        cR9 = cR10; robot(cR9).coreR = 9;
        cR10 = cR11; robot(cR10).coreR = 10;
        cR11 = cR12; robot(cR11).coreR = 11;
        cR12 = cR13; robot(cR12).coreR = 12;
        cR13 = cR14; robot(cR13).coreR = 13;
        cR14 = cR15; robot(cR14).coreR = 14;
        cR15 = cR16; robot(cR15).coreR = 15;
        cR16 = cR17; robot(cR16).coreR = 16;
        cR17 = cR18; robot(cR17).coreR = 17;
        cR18 = 0;
        modeStart = modeStart-0;
        modeCoreL = modeCoreL-0;
        modeCoreR = modeCoreR-1;
        modeOpposite = modeOpposite+1;
        Construct6a2 = false;
        Construct6a3 = true;
    end
    pause(dt);
    count = count + 1;
    %F = getframe();
    %writeVideo(writerObj, F);
end % cR1 disassembles; cR2-cR18 become cR1-cR17
```

Script: Script: Front Robot of the Left Side Line Disassembles

```
while (Construct6a2 == false) && (Construct6a3 == true)
    T = T + dt; tDisassembly = tDisassembly + dt; tConstruct = tConstruct + dt;
    disassembleDistance = calcDistance([robot(cL1).x, robot(cL1).y], [robot(cL2).x, robot(cL2).y]);
    [robot, border, cliff, dynObstDistance, target] = refreshBotData(robot, border, cliff, map, target, dt, N);
    refreshPlotData(robot, map, target, N);
    for i = 1:N
        if robot(i).opposite == 1
            [robot, r] = searchAssemble(robot, i, vmax, dthetamax, probr, probl, border, cliff, dynObstDistance)
        elseif robot(i).coreL == 1
            robot(i).v = vmax; robot(i).dtheta = 0;
        else
            robot(i).dtheta = 0; robot(i).v = 0;
        end
        [robot] = calcPoints(robot, i);
    end
    if (disassembleDistance > 30)
        robot(cL1).opposite = 1;
        robot(cL1).coreL = 0;
        clear cL1;
        cL1 = cL2; robot(cL1).coreL = 1;
        cL2 = cL3; robot(cL2).coreL = 2;
        cL3 = cL4; robot(cL3).coreL = 3;
        cL4 = cL5; robot(cL4).coreL = 4;
        cL5 = cL6; robot(cL5).coreL = 5;
        cL6 = cL7; robot(cL6).coreL = 6;
        cL7 = cL8; robot(cL7).coreL = 7;
        cL8 = cL9; robot(cL8).coreL = 8;
        cL9 = cL10; robot(cL9).coreL = 9;
        cL10 = cL11; robot(cL10).coreL = 10;
        cL11 = cL12; robot(cL11).coreL = 11;
        cL12 = cL13; robot(cL12).coreL = 12;
        cL13 = cL14; robot(cL13).coreL = 13;
        cL14 = cL15; robot(cL14).coreL = 14;
        cL15 = cL16; robot(cL15).coreL = 15;
        cL16 = cL17; robot(cL16).coreL = 16;
        cL17 = cL18; robot(cL17).coreL = 17;
        cL18 = 0;
        modeStart = modeStart-0;
        modeCoreL = modeCoreL-1;
        modeCoreR = modeCoreR-0;
        modeOpposite = modeOpposite+1;
        Construct6a3 = false;
        Construct6a4 = true;
    end
    pause(dt);
    count = count + 1;
    %F = getframe();
    %writeVideo(writerObj, F);
end % cL1 disassembles; cL2-cL17 become cL1-cL17
```

References

Alonso, J. R. (2015). Building Bridges. Available from: http://mappingignorance.org/2015/12/16/. Accessed 14 April 2016.

Ampatzis, C., Tuci, E., Trianni, V., Christensen, A. L., and Dorigo, M. (2009). Evolving Self-Assembly in Autonomous Homogeneous Robots: Experiments with Two Physical Robots. *Artificial Life*, 15:465–484.

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

ANYKODE (2014). Simulate your Robotics Applications. Available from: http://www.anykode.com/index.php. Accessed 15 August 2014.

Ashley-Rollman, M. P., Pillai, P., and Goodstein, M. L. (2011). Simulating Multi-Million-Robot Ensembles. In *2011 IEEE International Conference on Robotics and Automation*, pages 1006–1013, Shanghai, China. IEEE.

Avril, A., Purcell, J., and Chapuisat, M. (2016). Ant workers exhibit specialization and memory during raft formation. *The Science of Nature*, 103(36):1–6.

Aziz, F., Bandaranayake, R., Cole, A. C., and Harwood, R. (2015). Octbot: A Low Cost Platform for Swarm Robotics Research. Technical report, Electronics and Computer Science University of Southampton, Southampton, UK.

Barca, J. C. and Sekercioglu, Y. A. (2013). Swarm robotics reviewed. *Robotica*, 31:345–359.

Bayindir, L. and Şahin, E. (2007). A Review of Studies in Swarm Robotics. *Turkish Journal of Electrical Engineering*, 15(2):115–147.

Beni, G. (2005). From Swarm Intelligence to Swarm Robotics. In Şahin, E. and Spears, W. M., editors, *Swarm robotics workshop: state-of-the-art survey*, pages 1–9. Springer, New York.

Bhaumik, A. (2012). Open Source Robotics: Multi-Robot Simulators. *Linux For You Magazine*, pages 48–50.

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

Bonani, M., Baaboura, T., Retornaz, P., Vaussard, F., Magnenat, S., Burnier, D., Longchamp, V., and Mondada, F. (2010a). Marxbot. Available from: `http://mobots.epfl.ch/`. Accessed 23 July 2015.

Bonani, M., Longchamp, V., Magnenat, S., Retornaz, P., Burnier, D., Roulet, G., Vaussard, F., Bleuler, H., and Mondada, F. (2010b). The MarXbot, a Miniature Mobile Robot Opening new Perspectives for the Collective-robotic Research. In *The IEEE/RSJ International Conference on Intelligent Robots and Systems*, Taipei, Taiwan.

Bongard, J. (2014). *Why Morphology Matters*, chapter 6, pages 125–152. The MIT Press, Cambridge, MA, USA.

Brambilla, M., Ferrante, E., Birattari, M., and Dorigo, M. (2013). Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7:1–41.

Brambilla, M., Pinciroli, C., Birattari, M., and Dorigo, M. (2012). Property-driven design for swarm robotics. In *11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, pages 139–146, Valencia, Spain. International Foundation for Autonomous Agents and Multiagent Systems.

Brennan, L. (2012). That's teamwork: Ants turn themselves into a bridge after their route home is blocked. Available from: `http://www.dailymail.co.uk/news/article-2119826/Indian-ants-turn-bridge-route-home-blocked.html`, Accessed 18 July 2014.

Bruni, R., Corradini, A., Gadducci, F., Lafuente, A. L., and Vandin, A. (2015). Modelling and analyzing adaptive self-assembly strategies with Maude. *Science of Computer Programming*, 99:75–94.

Carpin, S., Lewis, M., Wang, J., Balakirsky, S., and Scrapper, C. (2007). USARSim: a robot simulator for research and education. In *2007 IEEE International Conference on Robotics and Automation*, pages 1400–1405, Roma, Italy. IEEE.

Castano, A., Chokkalingam, R., and Will, P. (2000). Autonomous and Self-Sufficient CONRO Modules for Reconfigurable Robots. In *Distributed Autonomous Robotic Systems 4*, pages 155–164, Tokyo, Japan.

Chechkin, A., Metzler, R., Klafter, J., and Gonchar, V. (2008). Introduction to the Theory of Lévy Flights. In Klages, R., Radons, G., and Sokolov, I., editors, *Anomalous Transport: Foundations and Applications*, pages 129–162. Wiley, Weinheim.

Claytronics (2014). Claytronics. Available from: `http://www.cs.cmu.edu/~./claytronics/software/index.html`, Accessed 13 August 2014.

Correll, N. (2011). Swarm Intelligence and Self-Assembly course announcement. Available from: http://correll.cs.colorado.edu/?p=651. Accessed 24 July 2014.

Şahin, E. (2005). Swarm Robotics: From Sources of Inspiration. In Şahin, E. and Spears, W. M., editors, *Swarm robotics workshop: state-of-the-art survey*, pages 1–9. Springer, New York.

Cucu, L., Rubenstein, M., and Nagpal, R. (2015). Towards Self-assembled Structures with Mobile Climbing Robots. In *2015 IEEE International Conference on Robotics and Automation*, pages 1955–1961, Seattle, Washington. IEEE.

Cyberbotics (2014). Webots 7: fast prototyping and simulation of mobile robots. Available from: http://www.cyberbotics.com/. Accessed 11 August 2014.

delta3d (2014). delta3d. Available from: http://delta3d.org/. Accessed 12 August 2014.

Dorigo, M., Floreano, D., Gambardella, L. M., Mondada, F., Nolfi, S., Baaboura, T., Birattari, M., Bonani, M., Brambilla, M., Brutschy, A., Burnier, D., Campo, A., Christensen, A. L., Decugniere, A., Di Caro, G., Ducatelle, F., Ferrante, E., Forster, A., Guzzi, J., Longchamp, V., Magnenat, S., Gonzales, J. M., Mathews, N., de Oca, M. M., O'Grady, R., Pinciroli, C., G.Pini, Retornaz, P., Roberts, J., Sperati, V., Stirling, T., Stranieri, A., Stutzle, T., Trianni, V., Tuci, E., Turgut, A. E., and Vaussard, F. (2013). Swarmanoid: A Novel Concept for the Study of Heterogeneous Robotic Swarms. *IEEE Robotics & Automation Magazine*, pages 60–71.

EXPO21XX (2014). Ecagents: Embodied and communicating agents. Available from: http://www.expo21xx.com/automation21xx/15423_st3_university/default.htm. Accessed 23 July 2015.

EZPhysics (2014). EZPhysics: Robotic Simulation Made Easy. Available from: http://ezphysics.org/joomla/. Accessed 18 August 2014.

Gaines, J. (2013). Adventures Among Ants by Mark W. Moffett. Available from: http://www.librarypoint.org/adventures_among_ants_moffett. Accessed 24 July 2014.

Gere, J. M. and Goodno, B. J. (2009). *Mechanics of Materials*. Cengage Learning, Stamford, USA.

Gilpin, K. and Rus, D. (2010). Modular Robot Systems: From Self-Assembly to Self-Disassembly. *IEEE Robotics & Automation Magazine*, pages 38–55.

Gro*β*, R. and Dorigo, M. (2008). Self-Assembly at the Macroscopic Scale. In *Proceedings of the IEEE*, volume 96, pages 1490–1508.

Guizzo, E. (2012). Smart pebble robots will let you duplicate objects on the fly. Available from: http://spectrum.ieee.org/automaton/robotics/robotics-hardware/smart-pebble-robots-duplicate-objects. Accessed 23 July 2015.

Jasmine (2005). Jasmine: swarm robot platform. Available from: http://www.swarmrobot.org/index.html#. Accessed 14 August 2014.

Jorgensen, M., Ostergaard, E., and Lund, H. (2004). Modular ATRON: Modules for a self-reconfigurable robot. In *Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2068–2073, Sendai, Japan.

Kernbach, S., Meister, E., Schlachter, F., Jebens, K., Szymanski, M., Liedke, J., Laneri, D., Winkler, L., Schmickl, T., Thenius, R., Corradi, P., and Ricotti, L. (2008). Symbiotic Robot Organisms: REPLICATOR and SYMBRION Projects. In *PerMIS 2008*, pages 62–69, Gaithersburg, MD, USA.

Kernbach, S., Meister, E., Schlachter, F., and Kernbach, O. (2010). Adaptation and Self-adaptation of Developmental Multi-Robot System. *International Journal on Advances in Intelligence Systems*, 3(1 & 2):121–140.

Kernbach, S., Meister, E., Scholz, O., Humza, R., Liedke, J., Ricotti, L., Jemai, J., Havlik, J., and Liu, W. (2009). Evolutionary Robotics: The Next-Generation-Platform for On-line and On-board Artificial Evolution. In *2009 IEEE Congress on Evolutionary Computation*, pages 1079–1086, Trondheim, Norway.

Kernbach, S., Schlachter, F., Humza, R., Liedke, J., Popesku, S., Russo, S., Ranzani, T., Manfredi, L., Stefanini, C., Matthias, R., Schwarzer, C., Girault, B., Alschbach, P., Meister, E., and Scholz, O. (2011). Heterogeneity for Increasing Performance and Reliability of Self-Reconfigurable Multi-Robot Organisms. In *IROS11 workshop on Reconfigurable Modular Robotics*, San Francisco, CA, USA.

Klein, J. (2008). breve: a 3d Simulation Environment for Multi-Agent Simulations and Artificial Life. Available from: http://www.spiderland.org/breve. Accessed 14 August 2014.

Kornienko, S., Kornienko, O., Nagarathinam, A., and Levi, P. (2007). From real robot swarm to evolutionary multi-robot organism. In *2007 IEEE Congress on Evolutionary Computation*, pages 1483–1490, Singapore.

Kutzer, M. D. M., Moses, M. S., Brown, C. Y., Scheidt, D. H., Chirikjian, G. S., and Ahmad, M. (2010). Design of a New Independently-Mobile Reconfigurable Modular Robot. In *2010 IEEE Int. Conf. on Robotics and Automation*, pages 2758–2764, Anchorage, Alaska, USA.

Kuyucu, T., Tanev, I., and Shimohara, K. (2013). Hormone-Inspired Behaviour Switching for the Control of Collective Robotics Organisms. *Robotics*, 2:165–184.

Levi, P., Meister, E., and Schlachter, F. (2014). Reconfigurable swarm robots produce self-assembling and self-repairing organisms. *Robotics and Autonomous Systems*, 62:1371–1376.

Li, H., Wang, T., and Wei, H. (2016). Response Strategy to Environmental Cues for Modular Robots with Self-Assembly from Swarm to Articulated Robots. *Journal of Intelligent Robot Systems*, 81:359–376.

Li, H., Wei, H., Xiao, J., and Wang, T. (2015). Co-evolution framework of swarm self-assembly robots. *Neurocomputing*, 148:112–121.

Liedke, J. and Worn, H. (2011). CoBoLD - A Bonding Mechanism for Modular Self-Reconfigurable Mobile Robots. In *Proceedings of the 2011 IEEE International Conference on Robotics and Biomimetics*, pages 2025–2030, Phuket, Thailand.

Liu, W. and Winfield, A. F. (2012). *Distributed autonomous morphogenesis in a self-assembling robotic system*, chapter 3, pages 89–113. Springer, Berlin.

LpzRobots (2013). Lpzrobots. Available from: `http://robot.informatik.uni-leipzig.de/wiki/doku.php?id=start`, Accessed 18 August 2014.

Martinoli, A. (2001). Book Review: Collective Complexity out of Individual Simplicity. *Artificial Life*, 7:315–319.

Mathews, N., Christensen, A. L., O'Grady, R., and Dorigo, M. (2010). Cooperation in a Heterogeneous Robot Swarm through Spatially Targeted Communication. In *7th International Conference on Swarm Intelligence*, pages 400–407, Brussels, Belgium. Springer.

Mathews, N., Christensen, A. L., O'Grady, R., and Dorigo, M. (2012). Spatially Targeted Communication and Self-Assembly. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura, Algarve, Portugal.

Mathews, N., Christensen, A. L., O'Grady, R., Retornaz, P., Bonani, M., Mondada, F., and Dorigo, M. (2011). Enhanced Directional Self-Assembly Based on Active Recruitment and Guidance. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Francisco, CA, USA.

Mathews, N., Valentini, G., Christensen, A. L., O'Grady, R., Brutschy, A., and Dorigo, M. (2015). Spatially targeted communication in decentralized multirobot systems. *Autonomous Robots*, 38:439–457.

Meister, E. and Gutenkunst, A. (2012). Self-Adaptive Framework for Modular and Self-Reconfigurable Robotic Systems. In *ADAPTIVE 2012: The Fourth International Conference on Adaptive and Self-Adaptive Systems and Applications*, pages 30–37, Nice, France.

Meister, E., Nosov, E., and Levi, P. (2013). Automatic Onboard and Online Modelling of Modular and Self-Reconfigurable Robots. In *6th IEEE Conference on Robotics, Automation and Mechatronics (RAM)*, pages 91–96, Manila, Philippines.

Microsoft (2012). Microsoft Robotics Developer Studio. Available from: http://www.microsoft.com/en-gb/download/details.aspx?id=29081. Accessed 15 August 2014.

MilanHurban (2013). Ackermann steering geometry model and basic control in matlab. Available from: https://github.com/MilanHurban/ackermann. Accessed 10 February 2014.

MIT (2013). Self-assembling robots are here. Available from: https://www.eecs.mit.edu/news-events/media/self-assembling-robots-are-here. Accessed 23 July 2015.

Mohan, Y. and Ponnambalam, S. G. (2009). An Extensive Review in Swarm Robotics. *IEEE*, pages 140–145.

Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J.-C., Floreano, D., and Martinoli, A. (2009). The e-puck, a Robot Designed for Education in Engineering. In *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, pages 59–65, CasteloBranco, Portugal. IPCB.

Mondada, F., Pettinaro, G. C., Guignard, A., Kwee, I. W., Floreano, D., Deneuborg, J., Nolfi, S., Gambardella, L. M., and Dorigo, M. (2004). Swarm-bot: A New Distribute Robotic Concept. *Autonomous Robot*, 17:193–221.

Murray, L., Timmis, J., and Tyrrell, A. (2013). Modular self-assembling and self-reconfiguring e-pucks. *Swarm Intelligence*, 7:83–113.

Navarro, I. and Matia, F. (2013). An Introduction to Swarm Robotics. *ISRN Robotics*, 2013:1–10.

O'Grady, R., Christensen, A. L., and Dorigo, M. (2009). SWARMORPH: Multirobot Morphogenesis Using Directional Self-Assembly. *IEEE Transactions on Robotics*, 25(3):738–743.

O'Grady, R., Christensen, A. L., and Dorigo, M. (2012). *SWARMORPH: Morphogenesis with Self-Assembling Robots*, chapter 2, pages 27–60. Springer Berlin Heidelberg, Berlin.

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

Open-Source-Robotics-Foundation (2014). Gazebo: Robot simulation made easy. Available from: http://gazebosim.org/. Accessed 12 August 2014.

OpenRobots (2014). MORSE, the Modular OpenRobots Simulation Engine. Available from: http://www.openrobots.org/morse/doc/stable/morse.html. Accessed 18 August 2014.

Patil, M., Abukhalil, T., and Sobh, T. (2013). Hardware Architecture Review of Swarm Robotics System: Self-Reconfigurability, Self-Reassembly, and Self-Replication. *ISRN Robotics*, 2013:1–11.

Peeters, C. and Greef, S. D. (2015). Predation on large millipedes and self-assembling chains in *Leptogenys* ants from Cambodia. *Insectes Sociaux*, 62:471–477.

Penders, J. (2007). Robot Swarming Application. In *Liber Amicorum ter gelegenheid van de 60e verjaardag van Prof.dr. H. Jaap van den Herik, J. Donkers, Ed. Maastricht, Netherlands: Maastricht ICT Competence Center*, pages 227–234, Maastricht, Netherlands.

Pfeifer, R. and Bongard, J. (2007). *How the Body Shapes the Way We Think: A New View of Intelligence*. The MIT Press, Cambridge, Massachusetts.

Pfeifer, R., Lungarella, M., and Iida, F. (2007). Self-Organization, Embodiment, and Biologically Inspired Robotics. *Science*, pages 1088–1093.

Pinciroli, C. (2014). ARGoS: Large-scale robot simulations. Available from: http://bohr.ulb.ac.be/~pincy/argos/index.php. Accessed 13 August 2014.

Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Caro, G. D., Ducatelle, F., Stirling, T., Gutierrez, A., Gambardella, L. M., and Dorigo, M. (2011). ARGoS: a Modular, Multi Engine Simulator for Heterogeneous Swarm Robotics. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5027–5034, San Francisco, CA, USA. IEEE.

Player (2010). Player: The Player Project. Available from: http://playerstage.sourceforge.net/index.php?src=index. Accessed 12 August 2014.

Ramchurn, S. D., Wu, F., Jiang, W., Fischer, J. E., Reece, S., Roberts, S., Greenhalgh, C., Rodden, T., and Jennings, N. R. (2015). Human-agent collaboration for disaster response. *Autonomous Agents and Multi-Agent Systems: Special Issue on Human-Agent Interaction (to appear)*.

Robotics, C. (2014). v-rep: virtual robot experimentation platform. Available from: http://www.coppeliarobotics.com/. Accessed 14 August 2014.

Romanishin, J. W., Gilpin, K., and Rus, D. (2013). M-Blocks: Momentum-driven, Magnetic Modular Robots. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4288–4295, Tokyo, Japan. IEEE.

Rubenstein, M., Cornejo, A., and Nagpal, R. (2014). Programmable self-assembly in a thousand-robot swarm. *Science*, 345:795–799.

Saska, M., Vonasek, V., Kulich, M., Fiser, D., Krajnik, T., and Preucil, L. (2011). Bringing reality to evolution of modular robots: bio-inspired techniques for building a simulation environment in the SYMBRION project. In *IROS 2011 Workshop Reconfigurable Modular Robotics: Challenges of Mechatronic and Bio-Chemo-Hybrid Systems*, San Francisco, CA, USA.

Sendova-Franks, A. B. and Franks, N. R. (1998). Self-assembly, self-organization and division of labour. *Philosphical Transactions of the Royal Society of London B*, pages 1395–1405.

SeSAm (2012). Sesam. Available from: http://130.243.124.21/mediawiki/index.php/Main_Page. Accessed 14 August 2014.

Statheropoulos, M., Agapiou, A., Pallis, G. C., Mikedi, K., Karma, S., Vamvakari, J., Dandoulaki, M., Andritsos, F., and Thomas, C. L. P. (2015). Factors that affect rescue time in urban search and rescue (USAR) operations. *Nat. Hazards*, 75:57–69.

Swarm-Robot-Project (2014). Swarm Robot Project Simulator. Available from: http://hades.mech.northwestern.edu/index.php/Swarm_Robot_Project. Accessed 13 August 2014.

Syafitri, N., Crowder, R. M., and Chappell, P. H. (2017). Swarm Robots Crossing Small Voids: Practical Considerations. In *2017 IEEE International Conference on Systems, Man, and Cybernetics*, Banff, Canada. IEEE.

Syafitri, N., Crowder, R. M., Chappell, P. H., and Mazlan, J. D. (2015). A Self-Assembly Strategy for Swarm Robots. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems*, page 4750, Hamburg, Germany. IEEE.

Trianni, V. and Dorigo, M. (2006). Self organisation and communication in groups of simulated and physical robots. *Biological Cybernetics*, 95(3):213–231.

Trianni, V. and Nolfi, S. (2009). Self-Organizing Sync in a Robotic Swarm: A Dynamical System View. *IEEE Transactions on Evolutionary Computation*, 13(4):722–741.

Trianni, V. and Nolfi, S. (2010). *Evolving collective control, cooperation and distributed cognition*, chapter 6, pages 127–165. Pan Stanford Publishing, Singapore.

Trianni, V., Tuci, E., Ampatzis, C., and Dorigo, M. (2014). *Evolutionary Swarm Robotics: a theoretical and methodological itinerary from individual neuro-controllers to collective behaviours*, chapter 7, pages 153–178. The MIT Press, Cambridge, MA, USA.

Unsal, C. and Khosla, P. K. (2001). A Multi-Layered Planner for Self-Reconfiguration of a Uniform Group of I-Cube Modules. In *2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 598–605, Maui, HI. IEEE.

Unsal, C., Kiliccote, H., and Khosla, P. K. (2001). A Modular Self-Reconfigurable Bipartite Robotic System: Implementation and Motion Planning. *Autonomous Robots*, 10:23–40.

USARSim (2013). Usarsim. Available from: http://sourceforge.net/projects/usarsim/. Accessed 8 September 2014.

Voland, G. (1999). *Engineering by Design*. Addison-Wesley Longman, Inc., Reading, MA.

Vonásek, V., Saska, M., Winkler, L., and Přeučil, L. (2015). High-level motion planning for CPG-driven modular robots. *Robotics and Autonomous Systems*, 68:116–128.

Wei, H., Chen, Y., Tan, J., and Wang, T. (2010). Sambot: A Self-Assembly Modular Robot System. *IEEE/ASME Transactions on Mechatronics*, 16(4):745–757.

Wei, H., Chen, Y., Tan, J., and Wang, T. (2011). Sambot: A Self-Assembly Modular Robot System. *IEEE/ASME Transactions on Mechatronics*, 16(4):745–757.

Winfield, A. F. T. (2012). *Robotics: A Very Short Introduction*. Oxford University Press, Oxford, UK.

Winkler, L. and Worn, H. (2009). *Symbricator3D – A Distributed Simulation Environment for Modular Robots*, pages 1266–1277. Springer Berlin Heidelberg, Berlin.

Wolfe, K. C., Moses, M. S., Kutzer, M. D. M., and Chirikjian, G. S. (2012). M^3Express: A Low-Cost Independently-Mobile Reconfigurable Modular Robot. In *2012 IEEE International Conference on Robotics and Automation*, pages 2704–2710, Minnesota, USA. IEEE.

Yim, M., Roufas, K., Duff, D., Zhang, Y., Eldershaw, C., and Homans, S. (2003). Modular Reconfigurable Robots in Space Applications. *Autonomous Robots*, 14:225–237.