Evolving Test Environments to Identify Faults in Swarm Robotics Algorithms

HAO WEI

PhD

University of York

Computer Science

January 2018

Abstract

Swarm robotic systems are often considered to be dependable. However, there is little empirical evidence or theoretical analysis showing that dependability is an inherent property of all swarm robotic systems. Recent literature has identified potential issues with respect to dependability within certain types of swarm robotic control algorithms. However, there is little research on the testing of swarm robotic systems; this provides the motivation for developing a novel testing method for swarm robotic systems. An evolutionary testing method is proposed in this thesis to identify unintended behaviours during the execution of swarm robotic systems autonomously. Three case studies are carried out on flocking control algorithm, foraging algorithm, and task partitioning algorithm. These case studies not only show that the evolutionary testing method has the ability to identify faults in swarm robotic system, but also show that this evolutionary testing method is able to reveal failures in various swarm control algorithms. The experimental results show that the evolutionary testing method can lead to worse swarm performance and reveal more failures than the random testing method within the same number of computing evaluations. Moreover, the case study of flocking control algorithm also shows that the evolutionary testing method covers more failure types than the random testing method. In all three case studies, the dependability of each swarm robotic system has been improved by tackling the faults identified during the testing phase. Consequently, the evolutionary testing method has the potential to be used to help the developers of swarm robotic systems to design and calibrate the swarm control algorithms thereby assuring the dependability of swarm robotic systems.

Contents

List of Tables

List of Figures

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisors, Dr. Rob Alexander and Prof. Jon Timmis, for giving me the opportunity to research in the exciting field of swarm robotics. I am really appreciated for their endless patience, continuous support, helpful guidance, and tireless encouragement given to me throughout the development of this thesis. It has been such an honour and pleasure to be a student of Dr. Rob Alexander and Prof. Jon Timmis.

I would also like to express deepest appreciation to Dr. Fiona Polack for her priceless suggestion during every Thesis Advisory Panel meeting.

I would like to thank my parents who will always be there whenever I need them, not only for their financial assistance, but also for their support and guidance when needed, and for letting me do my own thing when I need to.

Finally, I would like to thank my beloved wife, Guanlan Hu, for knowing me, understanding me, believing in my dreams, and caring about my feelings. And I would like to thank my cats, Rondo and Kiddo, for being amazing companions.

Declaration

I declare that this thesis is a presentation of original work and I am the sole author. This work has not previously been presented for an award at this, or any other, University. All sources are acknowledged as References.

Contributions from this thesis have been published in the following paper:

> Wei, Hao, Jonathan Ian Timmis, and Robert David Alexander. "Evolving Test Environments to Identify Faults in Swarm Robotics Algorithms." IEEE Congress on Evolutionary Computation 2017.
> Based on research described in Chapter 3 and Chapter 4 of this thesis.

Chapter 1

Introduction

Swarm Robotics is the study of how to coordinate large numbers of robots and is inspired by the emergent behaviour observed in nature such as colonies of ants (Bonabeau, Dorigo, and Theraulaz, 1999; Labella, Dorigo, and Deneubourg, 2006), mound-building of termites (Bonabeau, Dorigo, and Theraulaz, 1999; Kube and Zhang, 1993), and information exchange in bacteria (Şahin, 2004; Pugh and Martinoli, 2008). Swarms in nature are well known for their abilities to coordinate their behaviour in order to achieve tasks that are beyond the capabilities of a single individual. Even though there are no centralised coordination mechanisms involved within natural swarms, these swarms have been shown to be robust, flexible and scalable (Camazine, 2003). Consequently, a desired swarm robotic system should be able to work together in a collaborative manner in order to accomplish a task and produce significant results that an individual might not be able to perform on its own.

However, swarm robotic systems are facing various challenges. For example, they are not as robust to multiple failures of units as was first thought (Winfield, Harper, and Nembrini, 2004) and there are significant issues with the reliability and controllability of a swarm in complex tasks (Wei, Timmis, and Alexander, 2017). Issues such as communication failures, obstacles between the line of sight and failing units are examples of such problems (Wei, Timmis, and Alexander, 2017). In addition, swarms might be deployed in challenging environmental conditions such as underwater. The potential issues and challenges mentioned above require the robots and the swarm to adapt in an autonomous and distributed manner.

Due to the collective behaviour of swarm robotic systems, the whole system might function incorrectly if any individuals start to exhibit unintended behaviours. In computer system, a failure occurs if a system does not function correctly and a fault is the potential cause of a failure (Laprie, 1985). Consequently, a failure occurs when any robots in a swarm function abnormally. In order to develop a real world swarm robotic system,

TABLE 1.1: Differences between Swarm Robotics and Other Multi-Robotics

	Swarm Robotics	Other Multi-Robotics
Population Size	Varies in great range	Relatively small
Control	Autonomous and Decentralized	Remote or Centralized
Environment	Unknown	Known or Unkown
Scalability	High	Low
Flexibility	High	Low

certain approaches should be taken to not only ensure that the system exhibits expected behaviours but also ensure that no unintended behaviours occur.

1.1 The Differences between Multi-Robot Systems and Swarm Robotic Systems

Multi-robot systems are collections of two or more robots which works together to achieve tasks (Dudek et al., 1996). Multi-robot systems are introduced to overcome the spatial and information processing limitations in single robot systems. Studies (Parker, 1994; Mataric, Nilsson, and Simsarin, 1995) have shown that it would be more effective and efficient to use a number of simple robots to perform some specific tasks, such as environment exploration and objects transportation, rather than using one very complex robot.

The study of multi-robot systems has been extended to many different sub-areas such as cellular robotics, distributed robotics and collective robotics. Swarm robotics is one sub-area of collective robotics in which swarm intelligence techniques are applied. Table 1.1 shows the major differences between swarm robotics and other multi-robotics. The following three desirable qualities of swarm robotics make swarm robotic systems more beneficent than other multi-robot systems (Şahin, 2004):

- Robustness - The swarm robotic system should support a high level of robustness towards failure of individuals and disturbances in the environment.
- Flexibility - The swarm robotic system should possess the ability of adapting to the changing requirements of the environment.
- Scalability - The coordination mechanisms of the swarm robotic system should be able to operate properly under a wide range of group sizes.

1.2 Dependability of Swarm Robotic Systems

Study (Winfield, Harper, and Nembrini, 2006) introduced the notion of a *"dependable swarm"*, which is *"a complex distributed system, designed using the Swarm Intelligence paradigm, which meets standards of analysis, design, and test that would give sufficient confidence that the system could be employed in critical applications."* From the point of view of dependability (Winfield, Harper, and Nembrini, 2004), a dependable swarm robotic system should have the following two properties: *"liveness"* and *"safety"*. Liveness is the property of exhibiting desirable behaviours, which means that the system should always do the right thing. Safety is the property of not exhibiting undesirable behaviours, which means that the system should not do the wrong thing.

1.3 Evolutionary Testing Method

One mechanism used for revealing failures is to observe the behaviour of a running swarm in different environments, which can be treated as test cases. In this thesis, a mechanism represents the instance (concrete occurrence) of a computing method or algorithm. The behaviour of a swarm is the way in which the swarm behaves according to the mechanism, which is implemented according to a certain swarm control algorithm, deployed in the swarm.

Creating test cases for a swarm robotic system is challenging as the inputs for triggering failures are unknown, so a range of diverse test cases, which should be able to cover as many circumstances as possible, are required in order to test the swarm properly. Because of the quantity, diversity and challenging requirement of test cases for swarm robotic systems, traditional methods such as manual test case generation and random test case generation are not adequate.

Due to the large solution space of unexpected behaviours, it would be helpful if the procedure of test cases generation could be automated. Evolutionary testing method is a technique which is able to generate test cases automatically by using optimising search techniques, such as genetic algorithm (McMinn and Holcombe, 2003). It has been widely used to create effective test cases for various forms of testing, such as structural testing (Wegener, Baresel, and Sthamer, 2001), finite state machine testing (Derderian et al., 2006), and autonomous agent testing (Nguyen et al., 2012). In order to find solutions among the large solution space of failures in swarm robotics, a novel testing method is proposed based on evolutionary testing in this thesis.

In computer system, the dependability of a system can be improved using methods such as fault prevention, fault tolerance, fault removal, and fault forecasting (Laprie, 1995) (see section 2.2.1.3). In swarm robotics research, fault detection mechanisms (Bjerknes and Winfield, 2013; Christensen, OGrady, and Dorigo, 2009) are developed to improve the dependability of swarm robotic systems by means of fault tolerance. In this thesis, fault removal method is used to improve the dependability of swarm robotic systems.

1.4 Research Hypothesis

The objective of this thesis is to develop a testing method which is able to identify faults in the control algorithm of a swarm robotic system. So the research hypothesis of this thesis is proposed as follows:

> *It is possible to improve the dependability of a swarm robotic system by involving testing process during its development. The testing method presented in this thesis is more effective in revealing failures during the testing process than the random testing method.*

This testing method presented in this thesis builds severe environments which are difficult for the swarm to operate. It also detects and records the robots which exhibit unexpected behaviour, that is, failure. If a failure occurs during the execution of the swarm, the information recorded can be used to determine which parts (mechanisms) of the swarm control algorithm are responsible for the failures. The mechanism responsible for failures is treated as the fault in the swarm control algorithm. The control algorithm might potentially be improved by replacing the weak mechanism with a better one. As a result, the dependability of this swarm robotic system can be improved.

1.4.1 Research Questions

The main research question is developed as follows in order to address the research hypothesis proposed above:

> *What methodology should be used in order to find out weaknesses of a swarm robotic control algorithm?*

In order to answer the main research question, three sub-questions are proposed below as a guidance in a step-by-step manner:

- *1. How does the evolutionary testing method perform in identifying weaknesses of a swarm robotic control algorithm comparing to the random testing method?* This sub-question guides the research in Chapter 4, 5, and 6.
- *2. Does the evolutionary testing method have the ability of testing different types of swarm robotic control algorithms?* Two swarm robotic control algorithms are treated as different types when the swarm behaviours emerged by them are different. For example, a flocking control algorithm (see section 2.4.1) and a foraging control algorithm (see section 2.4.2) are different types of swarm robotic control algorithms, while static partitioning strategy proposed in (Pini et al., 2014) and dynamic partitioning strategy proposed in (Buchanan, Pomfret, and Timmis, 2016) (see section 2.4.3 for both strategies) are the same type of swarm robotic control algorithm. This sub-question guides the research in Chapter 4, 5, and 6.
- *3. Are the parameters of the evolutionary testing method reusable when testing different types of swarm robotic control algorithms?* This sub-question guides the research in Chapter 5 and 6.

1.5 Thesis Contributions

This section summarizes a number of original research contributions of this thesis as follows:

Evolutionary Testing Method for Swarm Robotic Systems - This thesis introduces the first known method for solving the testing problem in swarm robotics. Evolutionary testing methods in computer systems and autonomous agents are adapted to identify faults in swarm control algorithm. This testing method can be used as the basis of a testing method for specific swarm behaviour, which is work published in (Wei, Timmis, and Alexander, 2017). The experimental infrastructure introduced in this thesis can be used as an assistance tool in developing a swarm robotic system. This contribution is presented in Chapter 3.

Three Case Studies for Evolutionary Testing Method - Three case studies are carried out on flocking control algorithm, foraging algorithm, and task partitioning algorithm. The experimental results not only show that the evolutionary testing method has a better ability to identify faults in swarm robotic systems than the random testing method, but also show that the evolutionary testing method is able to to reveal failures in different types of swarm control algorithm. This contribution is presented in Chapter 4, Chapter 5, and Chapter 6.

Save Human Effort on Parameter Analysis - The experimental results of case studies for foraging algorithm and task partitioning algorithm show that the parameters of the evolutionary testing method are reusable when testing similar (but different types) of swarm control algorithms. In such circumstances, the human effort on parameter analysis can be saved. This contribution is presented in Chapter 5 and Chapter 6.

Fault Removal to Improve The Dependability of Swarm Robotic System - In all three case studies, the dependability of each swarm robotic system has been improved by the means of fault removal. This shows the viability of using the evolutionary testing method to help the developers to assure the dependability of their swarm robotic systems. This contribution is presented in Chapter 4, Chapter 5, and Chapter 6.

Metrics for Measuring The Performance of Flocking Behaviour - A new set of metrics is developed to measure the performance of flocking behaviour. A fitness function is then defined according to these metrics to calculate the performance of a flocking swarm in the experiments. This contribution is presented in Chapter 4.

Failure Classification - Failure Classification for flocking behaviour is proposed in this thesis to identify the diversity of the failures discovered during the execution of the swarm. The experimental results show that the evolutionary testing method covers more failure types than the random testing method when testing flocking control algorithm. This contribution is presented in Chapter 4.

1.6 Thesis Structure

The rest of this thesis is structured as follows:

Chapter 2 - Background and Related Work

This chapter is divided into four sections. The first section provides an introduction to fundamentals and current state of swarm robotics. Then the design problem and the methodologies used to solve it are respectively discussed. The concepts of dependability in the context of computer systems are introduced at the beginning of the second section. A discussion of the dependability of swarm robotic systems is then presented. At the end of second section, fault detection techniques used to ensure the dependability of swarm robotic systems are discussed. In the third section, the processes of analysis and testing of the design problem for swarm robotics are discussed. In the last section, the background and related work of the swarm behaviours which are tested in this thesis are introduced.

Chapter 3 - Evolutionary Testing Method for Swarm Robotic Systems

The testing problems in current swarm robotics research are specified at the beginning of this chapter. The evolutionary testing method is proposed in order to solve the testing problems. Following this, the simulator used throughout this thesis, the external assistance tools developed for the simulator, and test case generator developed in this thesis are introduced at the end of this chapter.

Chapter 4 - Testing Method for Flocking Behaviour

This chapter carries out a case study for testing flocking behaviour. Metrics for measuring the performance of the flocking behaviour, failure classification used to categorize failures, chromosome, and fitness function are all defined. The flocking behaviour is then implemented in the simulator. Parameter analysis and experimental results are then discussed. At the end of this chapter, fault removal approach is applied to flocking control algorithm and the corresponding experimental results are discussed.

Chapter 5 - Testing Method for Foraging Behaviour

This chapter carries out a case study for testing ant foraging behaviour. It begins with the decisions taken with regard to the metrics, chromosome, fitness function, and control algorithm of ant foraging behaviour. Following this, the parameter analysis and experimental results are discussed. Then fault removal approach is applied to ant foraging behaviour and finally the experimental results are discussed.

Chapter 6 - Testing Method for Task Partitioning Behaviour

A case study for testing task partitioning behaviour is carried out in this chapter. First, the metrics for task partitioning algorithm are introduced and the control algorithm for task partitioning algorithm is developed in the simulator. Experiments are then carried out using the testing method built for ant foraging behaviour. The experimental results are analysed and fault removal approach is applied to the task partitioning algorithm. The performance of the improved task partitioning algorithm is discussed at the end of this chapter.

Chapter 7 - Evaluation and Conclusion

This chapter evaluates the research hypothesis proposed in section 1.4 and summarises the contributions and limitations of this thesis. Potential future research directions are also suggested.

Chapter 2

Background and Related Work

A brief introduction to fundamentals and current state of swarm robotics is provided at the beginning of this chapter. Then the design problem of the swarm robotics and the methodologies used to solve it are respectively discussed. The second section of this chapter introduces the concepts of dependability in the context of computer systems. Following this, a discussion of the dependability of swarm robotic system is presented. Fault detection technique used to ensure dependability in swarm robotic systems will be discussed at the end of second section. In the third section, the processes of analysis and testing of the design problem for swarm robotics are discussed. Finally, the background and related work of the swarm behaviours which are tested in this thesis are introduced.

2.1 Swarm Robotics

2.1.1 What is Swarm Robotics?

Erol Sahin (Şahin, 2004) states that:

> *"Swarm robotics is the study of how a large number of relatively simple physically embodied agents can be designed such that a desired collective behaviour emerges from the local interactions among the agents and between the agents and the environment."*

Swarm robotics research is inspired by the emergent behaviour observed in nature, such as colonies of ants (Bonabeau, Dorigo, and Theraulaz, 1999; Labella, Dorigo, and Deneubourg, 2006), mound-building of termites (Bonabeau, Dorigo, and Theraulaz, 1999; Kube and Zhang, 1993), and information exchange in bacteria (Şahin, 2004; Pugh and Martinoli, 2008). Natural swarms are well known for their ability to coordinate their behaviour in order to achieve tasks that are beyond the capabilities of a single individual, and such collective behaviours have aroused a great deal of attention and interest of biologists (Pasteels and Deneubourg, 1987;

Brian, 2012) in 1980s. Later studies (Bonabeau et al., 1997) have shown that there are no centralized coordination mechanisms involved within natural swarms. Hence, unlike the traditional multi-robotic systems which use centralised coordination mechanisms for controlling the behaviour of the systems, swarm robotics adopts a decentralised coordination mechanism in order to generate desired collective behaviour for the system (Şahin and Winfield, 2008). The desired swarm robotic system should be able to work together by collaborating to achieve goals that an individual might not be able to reach on its own.

In addition to the definition of swarm robotics provided above (taken from (Şahin, 2004)), Erol Sahin proposed a set of criteria for distinguishing swarm robotic research from traditional multi-robot research (Şahin, 2004):

- Autonomous Robots - The individuals involved in the swarm robotic system should physically interact with the environment and be autonomous.
- Large Number of Robots - Not only large number of robots should be involved in a swarm robotic system (greater than 10 is acceptable), but the swarm robotics should also aim for scalability.
- Few Homogenous Groups of Robots - The swarm robotic system should consist of relatively few homogenous groups of individuals. The individuals in each group should not be assigned different roles.
- Relatively Incapable or Inefficient Robots - The robots involved should be relatively incapable of or inefficient in performing the task on their own. "Incapable" means that the individuals are unable to carry out the task by themselves and the cooperation of a swarm is required in order to achieve the goal. "Inefficient" means that the deployment of multi-robots should improve the performance/robustness of handling the task.
- Robots with local sensing and communication capabilities: The robots involved should use local communications only and have limited sensing abilities. Global information is prohibited to ensure that the coordination between the robots is distributed.

In complement to the *fourth* criterion above is that Erol Sahin in (Sharkey, 2007) proposes a simple separation of swarm robotics into two sub-areas, Scalable Swarm Robotics and Minimalist Swarm Robotics. Both Scalable Swarm Robotics and Minimalist Swarm Robotics enforce constraints on local communication and decentralised control, but Minimalist Swarm Robotics emphasizes the use of simple robots while Scalable Swarm Robotics does not concern about the simplicity of individuals.

2.1.2 Motivations for Swarm Robotics Research

Even though there are no centralised coordination mechanisms involved within natural swarms, these swarms are shown to be flexible, scalable and robust (Camazine, 2003). These three qualities are desirable for swarm robotic systems and so form motivations for the swarm robotics approach (Şahin, 2004):

- Robustness - The swarm robotic system should support a high level of robustness towards failure of individuals and disturbances in the environment.
- Flexibility - The swarm robotic system should possess the ability of adapting to the changing requirements of the environment.
- Scalability - The coordination mechanisms of the swarm robotic system should be able to operate properly under a wide range of group sizes.

2.1.3 Examples of Inspiration to Swarm Intelligence

Swarm robotics research resulted from the application of swarm intelligence concepts and so a good way of understanding swarm robotics is to take a close look at its root, that is, Swarm Intelligence. The term swarm intelligence, first introduced by Beni (Beni and Wang, 1993) in 1989, "is the collective behaviour of decentralized, self-organized systems, natural or artificial." The following are some examples of swarm intelligence algorithms inspired by varying biological systems.

2.1.3.1 Boids (Flocks, Herds, and Schools)

Flocks of birds, herds of land animals, and schools of fish are classic examples of emergent collective behaviour in nature. Reynolds (Reynolds, 1987) presented an approach (Boids) to simulate flocks of birds. The agents in the simulation follow a set of rules such as separation, alignment, cohesion, obstacle avoidance and goal seeking. This approach assumes that the form of a flock is the result of the interaction between the behaviours of the individual birds, so in order to simulate a flock, the behaviour of each bird is simulated. The birds are independent of each other in the simulation with each bird acting according to its local perception of the environment and the rules of flocking. Even though Reynold's approach is proposed for producing aggregate motion in computer animation (Reynolds, 1987), it has also been used as a swarm intelligence algorithm.

2.1.3.2 Insect Colonies

The colonies of insect are also sources of inspiration for swarm intelligence algorithm. To solve hard combinatorial optimisation problems, e.g. the traveling salesman problem (Dorigo and Gambardella, 1997), Ant Colony Optimization (ACO) (Dorigo, Di Caro, and Gambardella, 1999) is introduced. ACO (Dorigo, Birattari, and Stutzle, 2006) takes inspiration from the foraging behaviour of some ant species. These ants use pheromones as a communication medium. In fact, they lay down pheromones on the ground in order to lead other members in the colony to the resource. Consequently, each simulated 'ant' records its position and the quality of its solution so that better solutions can be selected by comparing the qualities of all the solutions. Another well-known example of taking inspiration from social insects is the Artificial Bee Colony algorithm (ABC). ABC is an optimization algorithm motivated by the intelligence foraging behaviour of honey bees (Karaboga, 2005) and it can be used for clustering analysis which identifies homogeneous groups of objects according to the values of their attributes. Moreover, the Bees Algorithm (Pham and Castellani, 2009) is another optimization algorithm motivated by the foraging behaviour of honey bees, and a combination of global explorative search and local exploitative search is used in the algorithm.

2.1.3.3 Particle Swarm Optimization

Particle Swarm Optimization (PSO) (Eberhart and Kennedy, 1995) is a population based optimisation methodology which is inspired by social behaviour of bird flocking. PSO searches for the optimum solution of a problem by repeatedly updating its candidate solutions according to their fitness values. PSO shares a few similarities with genetic algorithm, such as, randomly generated initial population, using fitness function for calculating fitness values for each solution, and reproducing the population with regard to fitness values. Even PSO does not have genetic operators such as crossover and mutation, each candidate has its own internal velocity and stores the best fitness value it has achieved thus far. These features allow PSO to have faster convergence to the best solution compared with genetic algorithm. The Darwinian Particle Swarm Optimization (DPSO) (Couceiro et al., 2012a), which uses natural selection to improve the ability of escaping from local-optimal solution, is proposed as an extension of PSO. Robotic Darwinian PSO (RDPSO) (Couceiro et al., 2012b) extends the DPSO to multi-robot applications to allow dynamic partitioning for the whole population of robots. The experimental results in (Couceiro et al., 2013b) show that RDPSO has the ability to improve the scalability of applications by decreasing the amount of required information exchange among robots. Study (Couceiro et al., 2013a) carries out experiments to benchmark five state-of-the-art algorithms for cooperative exploration tasks. Both simulated and physical experimental results show that the RDPSO algorithm converges to the optimal solution faster and more accurately than other algorithms without significantly increasing the computational demand, memory and communication complexity. A swarm exploration strategy, named Darwinian Robotics Strategy, is proposed in (Sànchez, Vargas, and Couceiro, 2018) based on RDPSO. Darwinian Robotics Strategy is applied to simulated 3D underwater environments. The experimental results show that the Darwinian Robotics Strategy has the ability to increase the exploration speeds and improve the robustness of the swarm when compared to single remotely operated vehicles, which are controlled by a human operator.

2.1.4 Potential Applications

A long-term goal is to use swarm robotics to solve real-world problems. There follows a number of potential application areas where swarm robotics would be applicable:

- Nuclear, biological, and chemical (NBC) attack detection and reconnaissance (Akyildiz et al., 2002; Gu et al., 2006) - In chemical and biological warfare, swarm robotic system can be deployed in the friendly area and used as a chemical or biological warning system. The swarm robotic system can also be used for detailed reconnaissance once an NBC (mass destruction) attack is detected.
- Battlefield surveillance (Vincent and Rubin, 2004) – The battlefield can be rapidly covered using a swarm robotic system, e.g. UAVs (unmanned aerial vehicles) in order to watch for the activities of the opposing forces.
- Space exploration (Hinchey, Sterritt, and Rouff, 2007) – Swarm robotic system can be used for exploring regions that human beings or larger

robotic systems cannot reach. The swarm robotic system can also provide backups and ensure survival in outer space.

- Pollution detection (Cortes, Martinez, and Bullo, 2005; Cortés and Bullo, 2005) – Swarm robotic system can be used for detecting and identifying pollution.
- Search and rescue (Martinez, Cortes, and Bullo, 2007; Penders et al., 2011) – Swarm robotic system can be used in search and rescue of injured human beings in a disaster.

2.1.5 The Design Problem of Swarm Robotics

Swarm robotics can be defined as *"a novel approach to the coordination of large numbers of robots" (Şahin, 2004).* In order to design a swarm robotic system, it is essential to define the method of designing individual level behaviours which leads to the desired collective behaviour. In Camazine's book (Camazine, 2003), swarm robotic systems are shown to be complex systems. In Trianni's book (Trianni, 2008), Trianni refers to the problem of designing individual level behaviour as the design problem. As the dynamics of complex system are difficult to predict (Abbott, 2006), the design problem is non-trivial. As swarm robotics research is still at a very early stage, there are still no standard ways of solving the design problem. Due to the lack of analysis and testing methods, Brambilla declared that: *"The intuition of the human designer is still the main ingredient in the development of swarm robotics systems."* (Brambilla et al., 2013) Design methods are used for solving the design problem, and in swarm robotics the existing behaviour design methods can be categorized into two categories (Brambilla et al., 2013): behaviour-based design and automatic design. These two kinds of design methods will be discussed respectively in the next two sections.

2.1.5.1 Behaviour-based design methods

In behaviour-based design method, the behaviour of each individual robot is designed iteratively until the desired collective behaviour is obtained. The behaviour of each individual robot is a combination of the behaviour provided by its control algorithm and the disturbance in the environment. Because of the uncertainty existing in the environment, the behaviour of an individual robot is difficult to predict which makes it more difficult to provide a group of individual behaviours in order to predict the collective behaviour of the whole swarm system. Consequently, designing a swarm robotic system using behaviour-based design method is a trial-and-error process (Brambilla et al., 2013).

Even though the swarm robotic system designed by behaviour-based design methods is restricted by many constraints and is time-consuming, there are still many successful examples. Several examples of swarm robotic system designed using behaviour-based design methods are presented next:

- Soysal et al. (Soysal and Sahin, 2005) presented a systematic analysis of the probabilistic aggregation strategy design method in swarm robotic system. An aggregation behaviour in swarm robotic system is a combination of four basic behaviours: obstacle avoidance, approach, repel, and wait. The author uses a three-state finite state machine with two probabilistic transitions to combine the approach, repel and wait behaviours.
- Nouyan et al. (Nouyan and Dorigo, 2006) presented a swarm intelligence control mechanism for distributed robot path formation. The mechanism is able to form a path between two distant locations situated at a distance beyond the sensing ability of any single robot using chain-based robot path formation.
- Brambilla et al. (Brambilla et al., 2012) proposed a top-down design method for designing swarm robotic systems. This method is called property-driven design and uses a set of properties to define a desired system. When designing a swarm robotic system, a model of the system which meets all the desired properties is first produced, then the whole system is developed by following all the requirements defined for the desired system. By following this approach, the property-driven design assures that the final swarm robotic system satisfies all the required properties.

In conclusion, most behaviour-based design is a bottom-up process (Crespi, Galstyan, and Lerman, 2008) except the recently proposed property-driven design method (Brambilla et al., 2012). Developers design, test and modify the behaviours of the individuals until the desired collective behaviour is obtained. This makes behaviour-based design time-consuming, and the quality of the swarm robotic system depends on the expertise of the developers.

2.1.5.2 Automatic design methods

Automatic design methods can be used to generate desired collective behaviour automatically without the explicit intervention of the developer, and the use of automatic design methods may also reduce the effort of the development (Brambilla et al., 2013) and improve the quality of the swarm robotic systems (Brambilla et al., 2013; Brambilla et al., 2012). Brambilla in (Brambilla et al., 2013) divided existing automatic design methods into two categories: reinforcement learning and evolutionary robotics.

Reinforcement learning In reinforcement learning, the agent receives feedback for its actions, and learns its behaviour through trial-and-error interaction with the environment (Kaelbling, Littman, and Moore, 1996). There are some studies (Matarić, 1997; Panait and Luke, 2005) about reinforcement learning in multi-robot system, but all the design methods introduced have limited scope, such as non-dynamic environment. In case of a single robot, the agent normally receives reward at individual level. However, in the case of multi-robot systems, the challenging problem is how to separate the global reward into individual rewards. Another challenging problem is that each individual robot is not only disturbed by the changes in the environment but also by the actions performed by other robots.

Maja (Matarić, 1997) presented a formulation of reinforcement learning which enables a group of four mobile robots to learn a foraging task in a noisy and dynamic environment, but as the size of the swarm grows (larger than 4), the method fails. There are no other methods which perform better in reinforcement learning for swarm robotic systems than Maja's.

In conclusion, it might not be suitable to treat swarm robotics problem as a reinforcement learning problem. Indeed, there are no evidences showing that reinforcement learning design methods can be suitable to be applied to swarm robotic systems. There are also no evidences showing that reinforcement learning is suitable for testing swarm robotic systems. Moreover, due to the enormous size of the searching space when testing swarm robotic systems, reinforcement learning might also not be suitable to generate test cases for swarm robotic systems.

Evolutionary robotics Evolutionary robotics was first introduced by Cliff et al. (Cliff, Husbands, and Harvey, 1993) in 1993. It is an automatic design method which applies evolutionary computation techniques for developing autonomous robotic systems. The evolutionary computation technique (Nolfi et al., 2016) was inspired by the evolution by natural selection in biology and is typically used to search for near-optimal solutions because of their abilities to *"escape from local optimum and due to previous successes when applied to similar problems"* (Nolfi et al., 2016). The procedure of Evolutionary robotics design method (Nolfi et al., 2016) can be described as follows:

- 1. Generate initial population - A population of individual behaviours is generated randomly following certain specific rules;
- 2. Execution - Each individual behaviour is executed a number of times. For each execution, the performance of each behaviour is individually recorded;
- 3. Calculate fitness value - The fitness value for each individual behaviour is measured according to the performance of the individual. Fitness values are calculated using fitness function, which uses a single figure of merit to summarise the goodness of the individual. If no improvements are observed after a number of generations, the procedure stops;
- 4. Reproduction - Genetic operators, such as crossover and mutation, are applied to a selection of individual behaviours with highest fitness value.

Nolfi et al. (Nolfi et al., 2016) showed that the evolution occurs on both real robots and simulated robots. Even though the evolutionary process is feasible on real robots (Meyer, Husbands, and Harvey, 1998), the time needed for evaluating real robots is much longer than that for simulated robots. One way to avoid the time problem is to evolve robots in the simulation and then implement the robots with highest fitness value on real robots. However, there is a "reality gap" (Jakobi, Husbands, and Harvey, 1995) for evolutionary robotics between the simulation and real world. This reality gap might cause problems when implementing simulated robots on physical robots. Varela et al. (Varela and Bourgine, 1992) claimed the following problems occur:

- *"Without regular validation on real robots there is a great danger that much effort will go into solving problems that simply do not come up in the real world with physical robots;"*
- *"There is a real danger that programs which work well on simulated robots will completely fail on real robots because of the differences in real world sensing and actuation - it is very hard to simulate the actual dynamics of the real worlds."*

Nolfi et al. (Nolfi et al., 2016) described two techniques which can be used to reduce the “reality gap”. One is called Methods for Accurately Modelling Robot-Environment Interactions, which attempts to model the interactions between the robots and the environment as accurately as possible, while the second is called Minimal simulations, which attempts to model only the characteristics that are relevant for forming the required behaviour. Jakobi (Jakobi, Husbands, and Harvey, 1995) concluded that it is impossible to develop an exact copy of the robot and its environment in simulation, which denies the feasibility of Methods for accurately modelling robot-environment interactions. Jakobi (Jakobi, 1997) also concluded that it is possible to transfer evolved simulations to reality by using minimal simulations. This allows the reality gap to be reduced without the need for an exact copy of the robot and its environment.

Evolutionary swarm robotics Trianni in (Trianni, 2008) argues that *“Evolutionary robotics represents an effective solution to the design problem because it eliminates the arbitrary decompositions at both the level of finding the mechanisms that lead to the emergent global behaviour, and the level of implementing those mechanisms into a controller for the robots.”* Therefore, Trianni proposed the Evolutionary Swarm Robotics approach, which applies Evolutionary Robotics techniques into swarm robotics to solve the design problem.

Unlike the behaviour-based design method in which the system is evaluated at the individual level, the evolutionary swarm robotic system is evaluated at the global level. The initial population of evolution consists of different individual behaviours. Each behaviour is assigned a fitness value based on its ability of producing the desired collective behaviour. During the evolution, well-performed individual behaviours are kept in the population and used to reproduce offspring while badly performed individual behaviours are discarded. Hence, in evolutionary swarm robotics, *“no arbitrary choice is performed by the designer, but the process is left free to choose and test any possible solution that can produce the desired global behaviour.”* (Trianni, 2008)

Dorigo et al. in (Dorigo et al., 2004) obtained self-organising behaviour for a swarm robotic system in different environmental situations by synthesizing the individual behaviours using artificial evolution in the simulation. Analysis also showed that the evolved swarm robotic system scales well with different sizes of systems. Moreover, Dorigo et al. (Dorigo et al., 2006) showed that the different collective behaviours, such as coordinated motion and self-assembly, of evolved swarm robotic system can still be observed when transferred from the simulation to the physical swarm robotic systems.

A coherent theoretical and methodological approach is represented in (Vargas et al., 2014) to achieve the synthesis of self-organizing behaviours for a swarm robotic system. In (Vargas et al., 2014), a series of experiments are performed for different types of swarm robotic control algorithms. All swarm controllers are evolved in the simulation. When the desired results have been obtained in the simulation, the evolved controllers are then implemented on physical robots to test the viability of the obtained controllers. The experiment results in (Vargas et al., 2014) show that the evolved swarm robotic system has the potential to perform as good in the real world as it does in the simulation.

Embodied evolutionary robotics In order to reduce the influence of the reality gap in evolutionary robotics, Watson et al. (Watson, Ficici, and Pollack, 2002) introduced Embodied Evolution as a new methodology for evolutionary robotics. In Embodied Evolution, a fully distributed evolutionary algorithm is embodied into a population of physical robots and these robots are able to perform the reproduction while doing a task in the physical environment. As an upgraded version of evolutionary robotics, one major advantage of embodied evolutionary robotics is that the developers do not need to be concerned about transferring the evolved results from simulated robots to real robots. Because the whole evolution takes place in the physical world, the developers do not need to simulate the model of the physical environment. Watson (Watson, Ficici, and Pollack, 2002) also mentioned that Embodied Evolution reduces the evaluation time for the evolution as the evolutionary algorithm is entirely decentralized across the swarm. Watson et al. (Watson, Ficici, and Pollack, 2002) suggested several circumstances that are suitable for Embodied Evolution:

- *Where a simulator for the task domain is impossible, unavailable, or insufficiently accurate.*
- *Where a centralized, globally coordinated adaptive algorithm is not implementable or is unavailable, or where coordination of parallelized embodied trials is difficult.*
- *Where we are interested in evolving interactive or collective behaviours.*
- *Where the interaction between task behaviours and reproductive behaviours is of interest.*
- *Where the agents must learn "in the field".*

Watson et al. (Watson, Ficici, and Pollack, 2002) showed that a swarm of physical robots is able to produce the desired beacon taxis behaviour by using Embodied Evolution. In swarm robotics, the behaviour of a swarm flocking together towards a light source is called beacon taxis. However, the robots in the swarm do not cooperate with each other to find the light source but rather only depend on themselves. Therefore, the beacon taxis behaviour in the experiment is not a collective behaviour.

O'Dowd et al. (O'Dowd, Winfield, and Studley, 2010) presented a methodology named Accelerated Distributed Evolutionary Algorithm (ADEA), which provides life time behavioural adaptability to each of the robots in the swarm rapidly and continuously. The approach taken in Accelerated Distributed Evolution is similar to Watson's Embodied Evolution (Watson, Ficici, and Pollack, 2002) except that the evolutionary evaluation in ADEA is executed using an on-board embedded simulator in each robot, which allows each robot to simulate the interaction between the other robots and itself. Hence, ADEA can be treated as a combination of a distributed embodied evolutionary algorithm and an embedded simulation.

By applying ADEA to the swarm robotic system, O'Dowd et al. (O'Dowd, Winfield, and Studley, 2010) managed to evolve distributed obstacle avoidance behaviour on varying sizes of swarms. However, ADEA is of limited use as the evolved behaviour in the experiment is very simple. Furthermore, another weakness of ADEA is that there is no feedback from the robots to the embedded simulator which causes inconsistency between the actual performance of the robots in the real world and the simulated performance.

O'Dowd et al. in (O'Dowd, Winfield, and Studley, 2011) categorize the "reality gap" into three correspondences between simulation model and reality:

- **Robot to Robot:** *"robot-robot correspondence refers to physical robotic aspects, such as differences in morphology."*
- **Robot to Environment:** *"robot-environment correspondence refers to differences in the dynamic interactions between a robot and the environment, both sensory and through actuation."*
- **Environment to Environment:** *"environment-environment correspondence relates the representation of salient features of the environment."*

O'Dowd et al. (O'Dowd, Winfield, and Studley, 2011) also present a distributed co-evolutionary method in order to fill up the vacancy of the feedback from the robots to the embedded simulator in ADEA. However, this upgraded methodology only validates the environment to environment correspondences of the embedded simulator. In (O'Dowd, Winfield, and Studley, 2011), O'Dowd et al. managed to emerge the foraging behaviour by evolving the swarm controllers in the experiment, but the chosen behaviour is not self-organising as there are no interdependencies between operating robots.

Experiments on embodied evolutionary swarm robotic system show that the swarm is able to evolve its behaviour online using embodied evolution. However, there is no cooperation between the robots in the swarm, none of the evolved behaviour belongs to self-organising behaviour. Due to the "reality gap" between the simulation and the real world, there are very few successful experiments showing that the evolutionary swarm robotic system can perform as well in the real world as they do in the simulation. Fortunately, the work in (Vargas et al., 2014) shows the viability of implemented the swarm controllers evolved in the simulation on physical robots. So in conclusion, even current evolutionary swarm robotics is not a formal and precise method to solve the design problem of swarm robotics, it still has the potential to solve the design problem of swarm robotics in the future.

2.1.6 Summary

This section first reviewed fundamentals and current state of swarm robotics. The methodologies used to solve the design problems of swarm robotic systems are discussed. The existing methodologies are categorized into two categories: behaviour-based design and automatic design. Current literature in behaviour-based design method showed that these kinds of methods use trial-and-error approaches and are never shown to be effective to solve the design problem. Automatic designed methods are shown to be less dependent on the expertise of the developers than behaviour-based design methods. Even there exists *"reality gap"* between the simulation and the real world, experimental results in (Vargas et al., 2014) showing that automatic design method has the potential to be as effective in the real world as it is in the simulation.

In conclusion, there are still no formal methods to solve the design problem in swarm robotics. Winfield et al. in (Winfield, Harper, and Nembrini, 2006) introduced a notion named "dependable swarm" in which the design problem might be solved with the assist of formal analysis and testing method. The dependability and analysis and testing methods of swarm robotics will be discussed in the following two sections respectively.

2.2 Dependability of Swarm Robotics

This section first introduces the concepts of dependability in the context of computer systems. Following this, a discussion of the dependability of swarm robotic system is presented. Finally, fault detection techniques used in swarm robotic systems will be discussed at the end of this section.

2.2.1 Dependability Concepts

Even though many people have given a definition to the term *Dependability*, Laprie's definition of Dependability is currently the most used. Laprie (Laprie, 1993) defined *Dependability* as *"The trustworthiness of a computer system such that reliance can justifiably be placed on the service it delivers."* The term *"service"* in this definition means the actions performed by the system which can be perceived by the environment. The environment consists of both the physical environment which contains the system, and another system (human or physical) which interacts with the system. Laprie (Laprie, 1995) defined a set of notions which is relevant to dependability and categorized them into three classes: the threats to dependability, the attributes of dependability, and the means for dependability. The categorization of notions for dependability are shown in Figure 2.1 and will be respectively discussed in the next three sections.

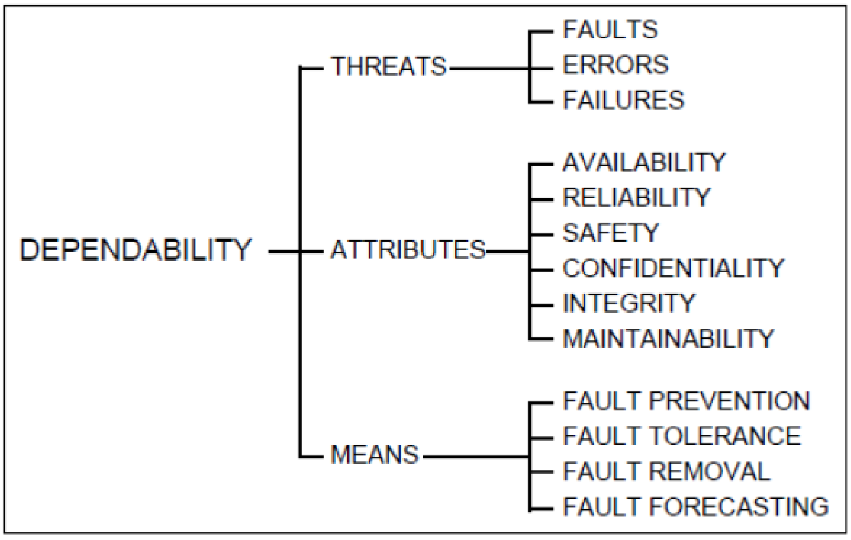


FIGURE 2.1: Laprie's dependability tree (Laprie, 1995)

2.2.1.1 Threats: faults, errors, and failures

Fault, error, and failure are always used to describe undesirable states which appear in a computer system. However, different authors may use these words differently which might mislead other researchers or readers. Consequently, in order to find uniformity for these words, Laprie (Laprie, 1985) made a distinction between fault, error, and failure:

- A system failure appears when the service delivered does not implement the desired system function, e.g. cash machine gives customers extra money.
- An error is the potential cause of a system failure, e.g. a mistake made during the development of a system.
- A fault is the potential cause of an error, e.g. a mistake in an algorithm.

Laprie in (Laprie, 1985) used several examples to illustrate faults, such as, *a programmer's mistake, a short-circuit occurring in an integrated circuit, an electromagnetic perturbation of sufficient energy*, and so on. Laprie separated the fault classes in the examples offered in (Laprie, 1985) into two groups:

- **Physical faults:** adverse physical phenomena:
  - **Internal:** *threshold changes, short-circuits, open-circuits;*
  - **External:** *environmental perturbations, such as temperature and vibration;*
- **Human-made faults:** imperfections which may be:
  - **Design faults:** *omissions either made a) during initial system design or during subsequent modifications, or b) during the establishment of operating or maintenance procedures;*
  - **Interaction faults:** *inadvertent or deliberate violations of operating or maintenance procedures.*

An error can either be latent or effective. A latent error is an error that is present but not detected, and an effective error is an error which has been detected. Laprie (Laprie, 1985) also stated the properties that decide errors:

- *A latent error becomes effective once it is active;*
- *An error may cycle between its latent and effective states;*
- *An effective error may, and in general does, propagate from one component to another; by propagating, an error creates other (new) errors.*

Finally, the categorization for failure is straightforward:

- **Physical failure:** system failure which is caused by physical fault;
- **Human-made failure:** system failure which is caused by human-made fault.

2.2.1.2 The Attributes of Dependability

Laprie (Laprie, 1995) classified dependability of computer system into six attributes, which are availability, reliability, safety, confidentiality, integrity and maintainability. In later study (Avizienis, Laprie, and Randell, 2001), Avizienis and Laprie introduced the attribute of security which can be viewed as a combination of confidentiality and integrity. They also suggested that secondary attributes can be defined for a computer system such as robustness, accountability, authenticity, and non-reputability. Only the five main attributes of dependability are discussed in this thesis, which are, availability, reliability, safety, security (confidentially and integrity), and maintainability.

Availability is defined as *"readiness for correct service"* (Laprie, 1995), which can be described as the ability of a computer system to deliver the correct service. In addition, Heddaya et al. (Heddaya and Helal, 1997) suggest that the attribute of availability should not only consider the readiness for correct service, but should also consider the accessibility of the system for correct service. For example, if all of the network resource built for a multi-service system is just sufficient for delivering one service to users, all services will not be available at the same time. Consequently, availability is the ability of a computer system to provide the correct service at a given instant of time.

Reliability is defined as *"continuity of correct service"* (Laprie, 1995), which is the ability of a computer system to deliver correct service continuously. Unlike availability which is instantaneous, reliability focuses on continuous operation over a period of time. In order for the system to operate continuously, recovery techniques are used for detecting the failures and restoring operation after failures occur.

Note that in Laprie's early work (Laprie, 1985), the dependability of a computer system has only two attributes: availability and reliability. This is because these are the only two attributes which exist in all computer systems. Therefore, the next three attributes may not appear in all systems.

Safety is defined as *"absence of catastrophic consequences on the user(s) and the environment"* (Laprie, 1995), which is the ability of a computer system to deliver the services without causing immediate and direct harm to its users and the environment. The attribute of safety always appears in a safety critical system, which is *"one whose failure could be sufficient to cause us harm."* (Burns, McDermid, and Dobson, 1992)

Security is defined as *"the absence of unauthorized access to, or handling of, system state"* (Laprie, 1995), which is the ability of a computer system to deliver the services without unauthorized access. The attribute of security always appears in a security critical system, which is *"one whose failure could not be sufficient to cause us harm, but could increase the number of possibilities, or likelihood of existing possibilities, for others to intentionally cause us harm."* (Burns, McDermid, and Dobson, 1992)

Maintainability is defined as *"ability to undergo repairs and modification"*, which is the ability of a computer system to be restored to operation state after a failure or modification. There are two kinds of maintenance techniques: preventive maintenance which is used for postponing failures and corrective maintenance which is used for correcting failures. Furthermore, a system with high levels of maintainability will increase the availability, reliability, safety, and security of the system.

2.2.1.3 The Means for Dependability

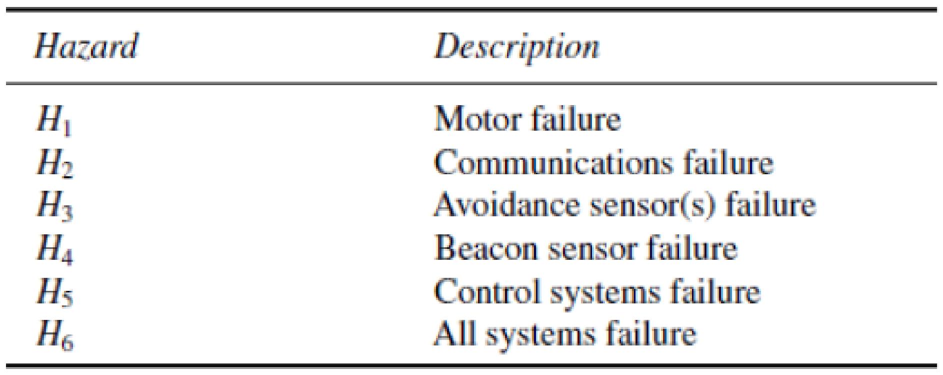
The dependability of a computing system is affected if faults exist in any of the five attributes discussed in section 2.2.1.2. In order to achieve a dependable computing system, various techniques are developed and have been classified into four groups (Laprie, 1995):

- **Fault prevention:** *how to prevent fault occurrence or introduction;*
- **Fault tolerance:** *how to guarantee a service to maintain the system's functions in the presence of faults;*
- **Fault removal:** *how to reduce the presence (number, seriousness) of faults;*
- **Fault forecasting:** *how to estimate the number, the future incidence, and the consequences of faults.*

2.2.2 Dependability of Current Swarm Robotics

In section 1.2, the dependability of swarm robotic systems and its attributes, which are liveness and safety, are introduced. Note that the attributes of dependability named by Winfield (Winfield, Harper, and Nembrini,

TABLE 2.1: INTERNAL HAZARDS FOR A SINGLE ROBOT (Winfield and Nembrini, 2006)



Hazard	*Description*
H_1	Motor failure
H_2	Communications failure
H_3	Avoidance sensor(s) failure
H_4	Beacon sensor failure
H_5	Control systems failure
H_6	All systems failure

2004) are not consistent with the nomenclature used by Laprie (see section 2.2.1.2), the attribute safety defined by Winfield is essentially a combination of reliability, safety, security and maintainability as defined by Laprie.

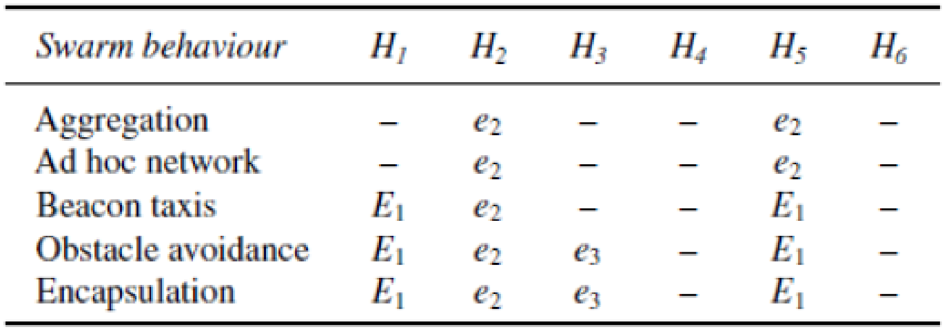
Swarm robotics literature frequently asserts that swarm robotic systems are dependable (Winfield and Nembrini, 2006). For example, as the decisions made in swarms are only based on local sensing and communications, swarm robotic systems are often assumed to have high levels of scalability. Swarm robotic systems are also assumed to be very robust due to the tolerance to failures of individual robots. However, no empirical evidence or theoretical analysis shows that dependability is automatically a property of all swarm robotic systems (Winfield and Nembrini, 2006).

To prove that the dependability of swarm robotic system does not come naturally, Winfield et al. (Winfield and Nembrini, 2006) explored fault tolerance of swarm robotic systems using Failure Mode and Effect analysis (FMEA). Winfield et al. in (Winfield and Nembrini, 2006) identified all possible internal hazards (faults of an individual robot), and then analysed the effect of these internal hazards on the overall behaviour of the swarm robotic system. Table 2.1 shows the internal hazards for a single robot, and Table 2.2 shows the effect of each internal hazards on overall swarm behaviours. The fault effects in Table 2.2 are defined as following (Winfield and Nembrini, 2006):

- Effect E1 (serious): *Motor failure anchoring the swarm.*
- Effect E2 (non-serious): *Lost robot(s) in the environment.*
- Effect E3 (non-serious): *robot collisions with obstacles or target.*

Table 2.2 shows that only hazard H1 and hazard H5 cause serious effect on high level collective behaviours, e.g. beacon taxis behaviour which

TABLE 2.2: SUMMARY OF FAILURE MODES AND EFFECTS (Winfield and Nembrini, 2006)



Swarm behaviour	H_1	H_2	H_3	H_4	H_5	H_6
Aggregation	–	e_2	–	–	e_2	–
Ad hoc network	–	e_2	–	–	e_2	–
Beacon taxis	E_1	e_2	–	–	E_1	–
Obstacle avoidance	E_1	e_2	e_3	–	E_1	–
Encapsulation	E_1	e_2	e_3	–	E_1	–

is the behaviour that a swarm moving towards a light source in a flock. When there is motor failure or control system failure in a robot, the faulty robot will lose its movability but its communication system continues to function. As the faulty robots remain within the ad hoc network of the swarm and become stationary in the environment, it will have the effect of anchoring the swarm at its location and thus compromise the formation of high level collective swarm behaviour. As both hazard H1 and hazard H5 are partial failures of a robot, the experimental results show that the tested swarm robotic system is not fault tolerant to partial failure of robots.

One interesting from the experimental results is that although hazard H6 (total system failure) is the most serious failure that an individual robot can have, results from Table 2.2 show that H6 has no effect on overall swarm behaviours at all. This is because a totally failed robot will be treated as a static obstacle and be avoided by other robots in the swarm. Hence, the experimental results show that the tested swarm robotic system is fault-tolerant to complete failure of robots.

Note that the effect of each internal hazard listed in Table 2.2 does not represent the importance of its associated mechanism. For example, hazard H2 (communication failure) is listed as non-serious effect for all five swarm behaviours. However, communication is one of the most important mechanisms for swarm robotic systems and *"in order to accomplish a given task (e.g., finding an object), robots must share information (e.g., about what they are sensing)."* (Couceiro et al., 2013b).

Winfield et al. (Bjerknes and Winfield, 2013) presented an analysis based on reliability modelling in order to explore how fault tolerance of a swarm robotic system is related to the size of the swarm. In this paper, the reliability of a swarm robotic system is modelled using k-out-of-N reliability modelling, in which N stands for the total number of robots in the swarm and K stands for the minimum number of operational (no faults)

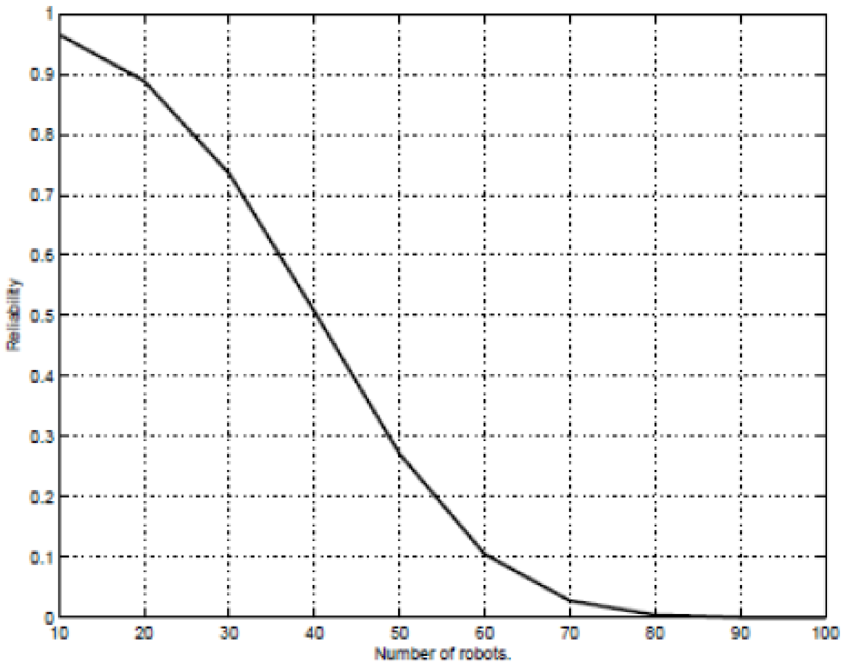


FIGURE 2.2: Reliability for a swarm with partially failed robots (Bjerknes and Winfield, 2013)

robots required for the overall swarm to operate correctly. The *reliability* is defined as the probability which the swarm will operate without failure. Figure 2.2 plots the reliability of a swarm robotic system against swarm size N when hazard H1 (motor failure) is induced and the average time before failure of an individual robot is set at 8 hours. It is obvious from Figure 2.2 that the reliability of the tested swarm decreases sharply as its size increases.

In conclusion, the work of Winfield et al. (Winfield and Nembrini, 2006) (Bjerknes and Winfield, 2013) showed that the current swarm robotics is:

- fault-tolerant to complete failure of robots
- less tolerant to partial failure of robots
- less reliable when the number of robots increases in case of partial failure of robots

2.2.3 Fault detection in swarm robotics

In order to improve the dependability of swarm robotic system, Winfield et al. (Winfield and Nembrini, 2006) suggested developing designed-in measures to neutralize the effect of partial failures. One way to counter the effect of partial failures is to use fault detection mechanism to detect and respond to failures in the swarm. Winfield (Bjerknes and Winfield, 2013) suggested a new robot behaviour from which the individuals should be able to identify neighbours who have partial failure, then 'isolate' those robots from the rest of the swarm. Christensen (Christensen, OGrady, and Dorigo, 2009) derived an exogenous fault detection algorithm (the swarm robotic systems require external equipment for fault detection) for swarm robotics by taking inspirations from the synchronized flashing behaviour observed in some species of fireflies. In his algorithm, the robots flash periodically. When a robot detects failure inside itself, it stops flashing. When a robot detects a failed robot, it will try to drag the failed robot into a safety area so that the whole swarm will continue to operate correctly. Therefore, the swarm can operate with relatively high failure rates. However, Christensen only considers the case of complete failure of robots. In case of partial failure of robots, Christensen's algorithm will not be able to detect and isolate the failed individuals. In order to increase the dependability of the swarm robotic system, a better exogenous fault detection mechanism is needed for detecting partial failed robots.

Winfield (Winfield, Harper, and Nembrini, 2006) suggested that in order to develop a dependable swarm, the system should meet the standards of analysis, design, and testing. Both Winfield (Winfield and Nembrini, 2006; Bjerknes and Winfield, 2013) and Christensen (Christensen, OGrady, and Dorigo, 2009) list a few principles for analysis and design of the swarm robotic system:

- the system should be able to operate efficiently under varying swarm size.
- the system should be able to operate when encountering disturbances from the environment.
- the system should be able to operate when losing individuals or being split apart.

2.2.4 Summary

This section has discussed the concepts of dependability in the context of both computer systems and swarm robotic systems. The work of Winfield et al. (Winfield and Nembrini, 2006) shows that the dependability of the swarm robotic system does not come naturally. Even though current swarm robotic systems are robust to complete failure of a single robot, they are most vulnerable to "partial failure" such as motor failure of a single robot. Fault detection methods such as the fireflies algorithm in

(Christensen, OGrady, and Dorigo, 2009) have been proposed to achieve a dependable swarm by means of fault tolerance (see section 2.2.1.3). However, these fault detection methods are still unable to detect and isolate partial failed robots in a swarm.

Moreover, there are no attempts in the literature to achieve a dependable swarm by means of fault removal (see section 2.2.1.3). Fault removal can be achieved by detecting faults using proper testing methods and then removing the faults by improving the algorithm. Up to the time of writing, there is no mention in the literature of principals for testing swarm robotic systems. Testing can help designers to identify faults in the swarm algorithm, and the algorithm might be improved by removing the faults identified. Consequently, it is worth developing a systematic swarm robotic testing method. Moreover, the dependability of swarms can be doubly assured by using a combination of testing and exogenous fault detection mechanisms.

2.3 Analysis and Testing

Performance assessment test-beds and formal analysis techniques to verify and guarantee the properties of swarm robotic system are still lacking. In order to guarantee the dependability of a swarm robotic system, Winfield (Winfield, Harper, and Nembrini, 2004) suggested that a set of disciplines should be applied during the processes of analysis, design and testing of the system. The design problem has already been discussed in section 2.1.5; the processes of analysis and testing for swarm robotic systems will be discussed in the following sections.

2.3.1 Analysis

Analysis is a fundamental phase in an engineering process (Pohl, 2010). In the analysis phase, the swarm robotic system development focuses on the analysis of whether a general property of the designed collective behaviour holds or not (Lerman, Martinoli, and Galstyan, 2004). In swarm robotics, the analysis of properties of the collective behaviours is usually carried out using models (Lerman, Martinoli, and Galstyan, 2004).

Lerman et al. (Lerman, Martinoli, and Galstyan, 2004) categorizes models of swarm robotic system into two levels: the microscopic level, which models the characteristics of the single robots, and the macroscopic level, which models the characteristics of the entire swarm. Modelling both the microscopic and the macroscopic level at the same time is extremely difficult due to the nature of self-organized systems (Abbott, 2006), so most modelling techniques only focus on one level at a time. Note that microscopic models are usually used for test analysis while macroscopic models are used for design analysis.

2.3.1.1 Microscopic level

At microscopic level, the models treat the robot as the fundamental unit of the model (Lerman, Martinoli, and Galstyan, 2004). The model not only describes the characteristics of the single individuals, it also describes the interactions between the robots and between the robots and the environment (Lerman, Martinoli, and Galstyan, 2004).

In microscopic models, simulations are always used for analysing the swarm robotic system. Friedmann (Friedmann, 2010) suggests that different levels of abstraction can be used when simulating the characteristics of a robot and the interaction between the robot and environment (including other robots). For example, a robot can be treated as point-masses in a 2D world in the simplest model while it can also be modelled using dynamic physics in a complex model with details of each sensor and actuator in a 3D world. As the behaviour of each individual robot is explicitly modelled at microscopic level, the microscopic model with high level of abstraction can also be used for design purposes in the behaviour-based design method. As the number of robots of the swarm robotic system is much larger than that of other mobile robotic systems, the simulators developed for other mobile robotic systems are not suitable for simulating swarm robotic systems. As a result, particular simulators that are scalable should be used when modelling swarm robotic systems at microscopic level.

Microscopic model can also give researchers an understanding of the global behaviour of the swarm robotic system. For example, Martinoli et al. (Martinoli, Ijspeert, and Gambardella, 1999) use a microscopic model to study the aggregation processes in a swarm of robots and to improve the understanding of the influence of the robot control parameters on collective aggregation by comparing the results between different collective aggregation mechanisms. However, due to the immaturity of swarm robotic systems, there are few successful examples of microscopic model for swarm robotic systems.

2.3.1.2 Macroscopic level

At macroscopic level, the models treat swarm robotic system as a whole (Lerman, Martinoli, and Galstyan, 2004). The model directly describes the collective behaviour of the swarm despite the characteristics of the individuals. Some of the macroscopic model techniques are discussed below.

Differential equations Differential equations can be used to model collective behaviour of a swarm at the macroscopic level. For example, Winfield et al. (Winfield et al., 2008) propose a probabilistic finite state machine (PFSM) which describes the collective behaviour of the swarm at macroscopic level. Winfield et al. (Winfield et al., 2008) managed to use PFSM and a number of differential equations to model a swarm of robots which is able to stay together while avoiding collisions. In a later study (Liu and Winfield, 2010), Liu and Winfield managed to use differential equations to model a collective foraging swarm.

As the first two steps of developing a macroscopic model for swarm robotic systems involving modelling the behaviour of individual robots as a finite state machine (FSM) (Lerman, Martinoli, and Galstyan, 2004) and then transforming the FSM into a PFSM (Liu and Winfield, 2010), the differential equation approach is able to transform microscopic models into macroscopic models systematically. However, since the positions of the robots in space are not explicitly modelled and the time is usually assumed in the differential equation approach, the major limitation of the differential equation approach is the difficulty of modelling spatial and temporal concepts.

Other modelling techniques Massink et al. (Massink et al., 2012) present a methodology of analysing consensus achievement in swarm robotic systems based on Bio-PEPA. Bio-PEPA (Biological Performance Evaluation Process Algebra) is a modelling and analysis framework developed for biological system. Its major advantage is that different types of analyses of a swarm robotic system based on the same system specification can be carried out (Massink et al., 2012). This will reduce the effort for developing multiple models when carrying out different analyses and while preserving the mutual consistency of the results. The major limitation of using Bio-PEPA is the difficulty of modelling and analysing spatial and temporal concepts. Moreover, another limitation is the lack of a direct link between a Bio-PEPA model and a physics-based simulations of the same system (Brambilla et al., 2013). Note that due to the lack of well-defined metrics, there are no modelling methods which are good enough to analyse swarm robotic systems precisely. When modelling a swarm robotic system, the developer has to choose the method that fits the system best on the basis of his/her own experience.

2.3.2 Testing

From section 2.1.5 and section 2.3.1, it is obvious that swarm robotics researchers have paid great attention to the processes of design and analysis of swarm robotic systems. However, there is very little literature focusing on swarm robotics testing methods. One reason for this might be that the current swarm robotic systems are still immature so that there is no need for tough testing to make them fail. The goal of testing is not only to find faults in a system, but also to support fault prevention. For example, Beizer (Beizer, 2003) concluded that bugs can be detected and eliminated at every stage in the software construction process by tests. Furthermore, Beck (Beck, 2003) proposed a style of development called test-driven development which provides the opportunities of improving the quality of a system by tackling the faults detected by tests. Consequently, research into testing methods of swarm robotic systems will not only be able to identify the faults, but also provide developers with opportunity of improving the dependability of the swarm (e.g. by applying an improved swarm control algorithm). In spite of the lack of literature in testing methods for swarm robotic systems, related fields will be reviewed in the following subsections in a general-to-specific order.

2.3.2.1 Software Testing

Software testing is a software development activity, which focuses on evaluating and improving the quality of a system by finding errors and problems (Abran et al., 2004). Testing is an important activity and should be applied throughout the whole development and maintenance process (Adrion, Branstad, and Cherniavsky, 1982). Inadequate testing will not only affect the quality of the system, but may also bring negative consequences to its users. For example, the failures occurred in the Therac-5 radiation therapy machine are known to give massive overdoses of radiation to the patients resulting in deaths or serious injuries (Leveson and Turner, 1993).

When carrying out a test for a system, testing can be applied to different levels of targets. The targets are often classified as follows:

- **Unit testing:** *"Testing of individual hardware or software units or groups of related units."* (Radatz, Geraci, and Katki, 1990) Unit testing is used for verifying that each individual unit of the source code (or a software component which is composed of related units) behaves exactly as expected. Each individual unit (or software component) should be tested separately.

- **Integration testing:** *"Testing in which software components, hardware components, or both are combined and tested to evaluate the interaction between them."* (Radatz, Geraci, and Katki, 1990) Integration testing is used for verifying the interaction between individual units (or software components).
- **System testing:** *"Testing conducted on a complete, integrated system to evaluate the compliance of the system with its specified requirements."* (Radatz, Geraci, and Katki, 1990) System testing is used for verifying whether the behaviour of the whole system is consistent with the customer's requirements specification.

The above classification of software testing is based on the scope of the testing. Note that the testing method developed in this thesis is system testing as the behaviour of the whole swarm is the test target. Pan (Pan, 1999) classified software testing into the following categories according to the purpose of testing:

- **Correctness testing** is used to test whether the system performs the correct behaviour. The testing techniques in correctness testing can be separated into two categories:
  - Black-box testing: The internal structure of the tested component or system is unknown and the test is only focus on the functionality of the system.
  - White-box testing: The test cases are derived from the internal structure of the tested component or system.
- **Performance testing** is used to evaluate whether a system meets the performance requirements. The evaluation of the performance of a software system usually includes: resource usage, throughput, and reaction time (Pan, 1999).
- **Reliability testing** is used to estimate the reliability of a system. The aim of this testing is to discover and remove all detected failures of the system before the system is released.
- **Security testing** is used to find out weaknesses of a system which might cause major harm to the users or the system.

Software testing is expensive and time-consuming. In order to cut down cost and time, one of the search-based testing techniques (Clarke et al., 2003) called Evolutionary testing is proposed (Pargas, Harrold, and Peck, 1999). Evolutionary testing is inspired by the theory of evolution in biology and can automatically generate test cases by using optimizing search techniques (Genetic Algorithm (Davis, 1991)). Evolutionary testing might be able to find a near-optimal solution but due to the random nature of a genetic algorithm, an optimal solution is never guaranteed.

2.3.2.2 Autonomous Agents Testing

Franklin et al. (Franklin and Graesser, 1996) defined an autonomous agent as *"a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future."* Even though an autonomous agent might communicate with other agents, current autonomous agents testing methods only focus on testing the behaviour of a single agent. Swarm robotic testing methods should focus on the overall behaviour of the whole swarm, therefore autonomous testing method can not be applied to test swarm robotic systems directly. Wooldridge et al. (Wooldridge and Jennings, 1994) pointed out the properties that an autonomous agent should have; they are:

- **Autonomy:** *agents are able to operate without the participation of humans or other systems;*
- **Social ability:** *agents are able to communicate with other agents or humans;*
- **Reactivity:** *agents are able to perceive and respond to the changes in the environment;*
- **Pro-activeness:** *agents are able to choose the actions on their own in order to reach their goals.*

Due to the peculiar nature of autonomous agents, testing autonomous agents is challenging as autonomous agents make decisions for themselves. Due to the autonomy of the agents, their behaviours are unpredictable. Unlike non-autonomous agents which will often have the same output for a given input, autonomous agents may behave in different ways. Consequently, in order to apply thorough testing to autonomous agents, a wide range of test cases is required. As traditional testing techniques for non-autonomous agent system will not work for testing autonomous agents (Rouff, 2002), the following testing methods are all able to generate test cases autonomously.

Soe et al. (Seo, Araragi, and Kwon, 2004) presented a testing method for checking the specification of an agent system by using Statecharts. A state-based model for an agent system is first built using extended Statecharts, and then the tests are generated. However, this testing method can only be used to test if the system meets the stakeholder's requirements and is unable to detect any faults existing in the system. Another method proposed by Zheng et al. (Zheng and Alagar, 2005) is also used to check the stakeholder's requirements. In order to detect faults in a system, Zhang et al. (Zhang, Thangarajah, and Padgham, 2007) presented a testing method for generating suitable test cases autonomously and detecting faults in a system. One drawback of this testing method is that the test cases generated is only suitable for unit testing.

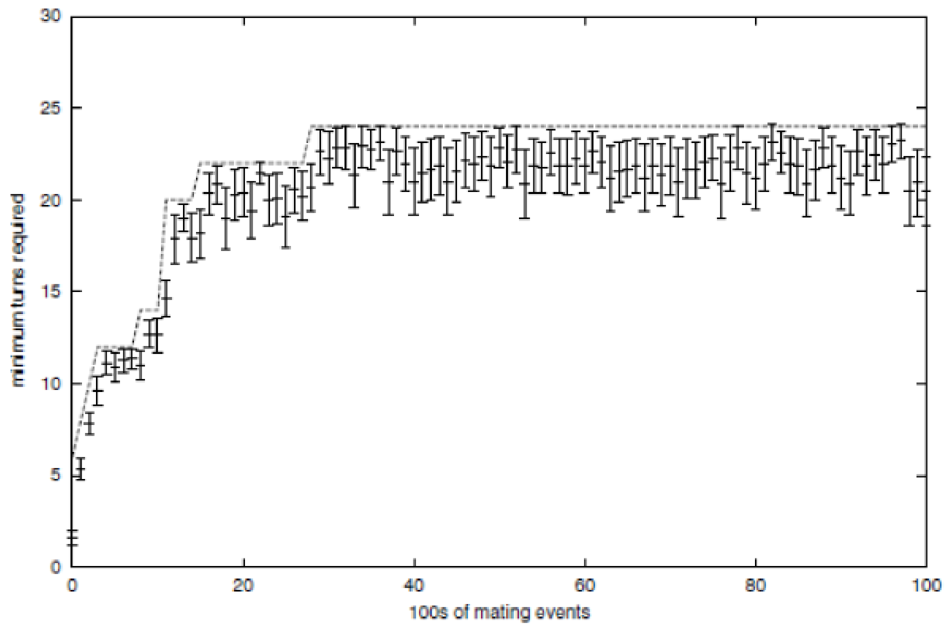


FIGURE 2.3: Maximizing the minimum number of turns for 100 generations (Ashlock, Manikas, and Ashenayi, 2006)

Ashlock et al. (Ashlock, Manikas, and Ashenayi, 2006) present an evolutionary computation system that can generate varies collection of test cases which can be used for testing path planning algorithms for autonomous robots. They use cellular representation in which directions are used to represent a group of objects and these in turn are used to construct obstructions in a test case. In study (Ashlock, Manikas, and Ashenayi, 2006), the initial test cases are generated randomly following a set of rules, and then an evolutionary algorithm is applied to generate evolved test cases. The fitness value of each test case is calculated via the dynamic programming algorithm stated in (Ashlock, Manikas, and Ashenayi, 2006) and it is the minimum number of turns (either return right or left) required for a mobile robot to reach the destination. Figure 2.3 shows the average number of the minimum number of turns (fitness value) required for a mobile robot to reach the destination for 100 generations of test cases in the evolution. The results shown in the graph reveal that the minimum number of turns of test cases has increased through the application of the evolutionary algorithm. Therefore, in testing a single autonomous robot, the evolutionary test case generator is able to generate large numbers of test cases which is more effective than random test cases.

Nguyen et al. (Nguyen et al., 2012) proposed an evolutionary testing method for evaluating the dependability of an autonomous agent. In this evolutionary testing method, the function used for evaluating the fitness value of each test case depends on the soft-goal of interest of the stake-holder and the problem domain. The initial test cases can be generated randomly or created by testers. A monitoring mechanism is used for observing and recording the behaviours of the mobile robots during the evolution. The data collected by the monitoring mechanism is used to compute the fitness values of choosing test cases. In the experiments, the robot has two built-in faults F1 and F2. The experimental results reported in the paper show that the evolutionary testing method is effective as the evolutionary technique detects both faults F1 and F2 while the random technique can only detect the easy fault, that is, F2.

2.3.2.3 Multi-agent System Testing

Multi-agent systems (MAS) are computational systems composed of multiple autonomous agents within an environment (Ferber, 1999). Study (Siciliano and Khatib, 2016) treats swarm robotics as an subset of multi-robotics which is a subset of multi-agent system, therefore the testing method for multi-agent system might be adapted when developing swarm robotic testing method. When testing a MAS, not only the agents inside the MAS should be tested, but also the interactions between the agents. Consequently, the test targets can be classified into the following classes based on the scope of the testing (Nguyen, 2009):

- **Unit:** the units that make up an agent are tested to ensure that each unit works as designed;
- **Agent:** the single autonomous agent is tested;
- **Integration:** the interactions between agents and the interactions between the agents and the environment are tested;
- **System:** the MAS system is tested as a whole in the target operating environment. The tests aims to find out whether the expected

emergent properties hold and whether the expected qualities are achieved;

- **Acceptance:** the MAS system is tested in the customer's execution environment in order to find out whether it meets the goals of the stakeholders.

When carrying out unit testing and agent testing in MAS, the testing methods that have been developed for testing single autonomous agent can be used. So only those testing methods testing MAS as a whole are discussed in this section.

Houhamdi et al. (Houhamdi and Athamena, 2011) introduced an approach for generating test suites for system testing of MAS. In this approach, a goal-oriented requirements analysis artefact is treated as the core element for test case derivation (Houhamdi and Athamena, 2011). Nguyen (Nguyen, Perini, and Tonella, 2009) proposed a comprehensive testing methodology which is able to generate test cases for all target testing levels, from unit to acceptance. This method derives test suites by complementing and exploiting goal-oriented analysis and design. However, for both of the above approaches, no realistic case studies are carried out to verify their usability on testing MAS.

In conclusion, the testing methods for MAS are still immature at system/acceptance level. Nguyen (Nguyen, 2009) pointed out some further research that can be carried out in order to improve the current testing methodology:

- *Testing MAS at system and acceptance level: how do developers build and end-users have confidence in autonomous agents?*
- *Test inputs definition and generation to deal with the open and dynamic nature of software agents and MAS.*
- *Test oracles: how to judge autonomous behaviour? How to evaluate agents that have their own goals from the subjective perspectives of a human tester?*
- *Testing emergent properties at macroscopic system level: how to judge whether or not an emergent property is correct? How to check the mutual relationship between macroscopic and agent behaviours?*
- *Deriving metrics such as safety, efficiency, and openness to assess the qualities of the MAS in test conditions.*
- *Reducing/removing side effects in test execution and monitoring because in many approaches introducing new entities in the system, e.g. mock agents,*

tester agents, and monitoring agents can influence the behaviour of the agents during tests and so the performance of the system as a whole.

2.3.2.4 Test Beds for Multi-robot Systems

A multi-robot system is system which consists of multiple robots that coordinate with each other to achieve a goal (Ferber, 1999). In order to verify the quality of a multi-robot system, experimental validation is usually carried out. Several test beds which are developed for multi-robot experimental validation are discussed in this section. Due to the similarity between multi-robot system and swarm robotic system, the test beds for multi-robot system can be treated as candidates for testing swarm-robotic system.

Pinciroli et al. (Pinciroli et al., 2012) presented a multi-robot simulator named ARGoS, which is developed to simulate large number of robots of various types. The experimental results in study (Pinciroli et al., 2012) showed that the simulation run-time increases linearly with the number of robots and the simulator is able to simulate 10,000 3D-dynamic e-puck robots is in real time. ARGoS is also be able to add new features to the experiment. As a result, the author declared that *"ARGoS is the first multi-robot simulator that is at the same time both efficient (fast performance with many robots) and flexible (highly customizable for specific experiments)"* (Pinciroli et al., 2012).

However, the work of Pinciroli et al. (Pinciroli et al., 2012) can only test a robotic system in the simulation. Therefore, how the robotic system performs in the physical world is unknown. In order to test robotic systems in the physical environment, Azamasab et al. (Azarnasab and Hu, 2007) presented the development of an integrated multi-robot test bed; they used the incremental simulation-based design methodology which includes the following phases:

- **Conventional simulation:** all of the robots are simulated in the simulator.
- **Robot-in-the-loop simulation:** some of the robots in the simulator are replaced with real robots which use a combination of virtual and real sensors/actuators so that they can sense other simulated robots and communicate with them.
- **Real robot testing:** all of the robots are real robots and they are tested under the real physical environment.

By using robot-in-loop simulation, the simulation-based study becomes one-step closer to reality (Azarnasab and Hu, 2007). As real robots are involved in the simulation, the confidence of the designers in final real system working increases. And finally, robot-in-loop simulation makes it possible to carry out system-wide tests and measurements without waiting for all of the real robots to be available for the large-scale robotic system (Azarnasab and Hu, 2007). However, only four real robots are used in the real robot testing in study (Azarnasab and Hu, 2007). In conclusion, the scalability of this incremental simulation-based test bed is still unclear and so is its usability.

2.3.3 Summary

Analysis techniques used in swarm robotics were reviewed in the first part of this section and they are divided into two levels: the microscopic level and macroscopic level. Due to the immaturity of swarm robotic systems, there are few successful examples of analysis techniques for swarm robotic systems. Due to the lack of studies in testing methods in swarm robotics, the second part of this section reviewed the testing methods in other fields which are relevant to swarm robotics. Testing methods for autonomous agents and multi-agent system were reviewed and methods proposed in studies (Ashlock, Manikas, and Ashenayi, 2006; Nguyen et al., 2012) might be adapted in order to develop a swarm robotic testing method. Test beds for multi-robot systems are the potential candidates for executing swarm robotic systems.

2.4 Swarm Behaviours

In this section, three swarm behaviours, which are flocking behaviour, foraging behaviour, and task partitioning behaviour, tested in our case studies (Chapter 4, 5, and 6) and the related work of these behaviours are introduced.

2.4.1 Flocking Behaviour

It is familiar to us that a group of birds fly together in the sky. There is also an old proverb saying that "Birds of a feather flock together". The reason that birds tend to fly together is not only because it can help them to stay safe from their predators, but also because it helps them to conserve energy by reducing wind resistance. For example, it is easier for a predator to attack one bird than attacking a group of birds. When a flock of birds is facing attack, more of them can stay out of the predator's sight and it is more convenient to spread the attacking information through the whole flock. Under these circumstances, a flock of birds can be treated as a natural swarm. The birds in a flock use only local communication and have limited environmental information. Hence, flocking is a collective behaviour of a group of interacting agents, e.g. birds in nature, working together to achieve a common goal without any centralized coordination.

2.4.1.1 Flocking Rules and Scenario

The first flocking simulation on a computer, named Boids, was accomplished by Craig Reynolds in 1987 (Reynolds, 1987). The agents in Boids use three simple rules to achieve the basic model (agents flock in a borderless environment without any obstacles) of flocking behaviour in the simulator, and described as follows (Reynolds, 1987):

- Separation: steer to avoid crowding local flockmates;
- Alignment: steer towards the average heading of local flockmates;
- Cohesion: steer to move toward the average position of local flockmates.

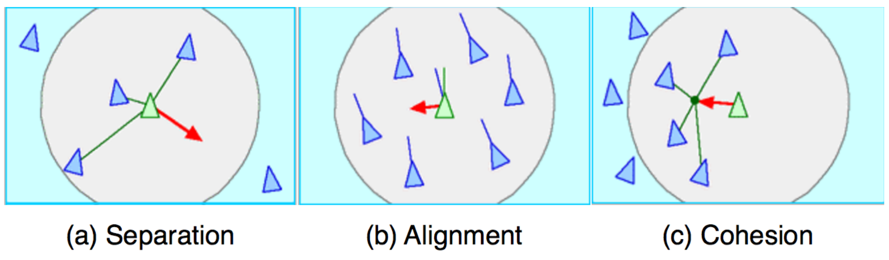


FIGURE 2.4: Steering rules of Boids (Reynolds, 1987).

Figure 2.4 shows these three basic steering rules of Boids. The blue agents within the grey circle are the local flockmates of the green agent. As the agents in the flock only use local communication, any agents that are outside the grey circle will be ignored. In order to make the flocking behaviour more complete (realistic), the following two rules can be added so that the agents in the flock can stay together to chase a target or reach a destination while avoiding collisions with static obstacles (Wei, Timmis, and Alexander, 2017):

- Obstacle avoidance: steer to avoid obstacles in the environment;
- Goal seeking: steer towards the direction of the goal.

There is no single definition of a standard flocking scenario in swarm robotic research. The following shows two commonly used flocking scenarios currently used in either multi-robotic research or swarm robotic research:

- **(1)** The agents of a flock move together in random directions in an empty environment;
- **(2)** The agents of a flock move towards a global target together or follow informed robots into which the moving directions are commanded or pre-defined.

Scenario 1 was commonly used at the beginning of flocking behaviour research in swarm robotics. In study (Moeslinger, Schmickl, and Crailsheim, 2011), flocking has been achieved in scenario 1 without sharing global information among the swarm. From then on, scenario 2 was widely used as it is more challenging. At the current level of swarm robotic research, certain global information, such as the heading direction of each robot, is still needed in order to achieve the task in scenario 2. Due to the challenging and extensive use of scenario 2, it will be used for carrying out the experiments in this case study. Figure 2.5 shows an example of scenario 2 with 4 informed robots in the swarm (Çelikkanat and Şahin, 2010).

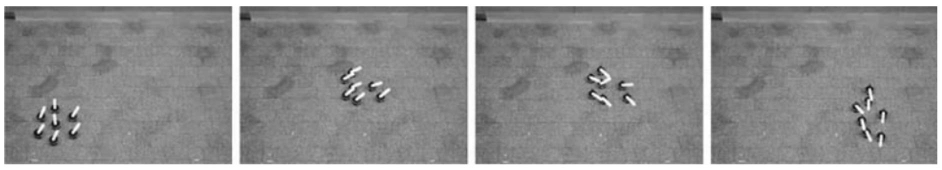


FIGURE 2.5: Snapshots of steered flocking with 7 robots. At the time of the second snapshot, 4 of the robots are commanded to turn 90 degrees to the right of their current direction. *White lines* indicate the heading directions of the robots (Çelikkanat and Şahin, 2010).

2.4.1.2 Related Work of Flocking Behaviour

Since Boids was implemented in simulation, several approaches for applying flocking behaviour to robotic swarms have been proposed. Even though there are lots of successful examples of applying flocking behaviour to swarm robotic systems, most of them (e.g. (Ferrante et al., 2012; Kelly and Keating, 1996)) violate one basic rule of swarm robotic systems, which is that only local communication and limited environmental information can be provided to the agents.

For the three basic steering rules of Boids, separation and cohesion are easy to implement, but alignment control is very difficult without sharing heading directions of each robot to other robots. In order to achieve alignment control, each agent needs to know its own heading direction and the heading directions of all its flockmates. In swarm robotics, there are usually two ways of implementing flocking behaviour. The first is to ignore the alignment rule but induce alignment control by using other approaches, such as following a light source in (Spears et al., 2004). However, in this category, none of the experimental results in the literature shows that the implemented swarm is robust. The second method is to implement alignment control by using global information (Stranieri et al., 2011; Holland et al., 2005; Ferrante et al., 2012). These flocking behaviours are more robust but they are not truly swarm systems according to the criteria established by Erol Sahin (Şahin, 2004). There are some controversies in this category as some researchers believe that sharing alignment information between the robots is not accessing global information.

2.4.2 Ant Foraging Behaviour

The ant is one of the most cited examples of a creature that can cooperate in large groups (Dorigo, Birattari, and Stutzle, 2006; Lambrinos et al., 2000). Ants can do many things, which are beyond the ability of any single ant, collectively: retrieve big prey, find the shortest path to the food, defend the colony, and so on. Even though there is a queen for each ant colony, the queen does not give instructions on how the ants should cooperate. The focus in this thesis is on ant-like foraging (search and return) behaviour. In such foraging behaviour, the individual agents should be able to explore the environment collectively to find the targets (food source) and progressively bring those targets to their nest.

2.4.2.1 Basic Foraging Algorithm

In ant foraging behaviour studies, a basic foraging algorithm (Matarić, 1995) is often used as a basis for extension to more complex foraging algorithms (Winfield, 2009). It is also used as a baseline to show the efficiency of a new proposed foraging algorithm. A new proposed foraging algorithm is efficient if it performs better than the basic foraging algorithm in the same testing environment. Finite state machine are often used to control the behaviour of an individual robot in ant foraging behaviour. Figure 2.6 shows the finite state machine of the basic foraging algorithm and the four states are defined as follows:

- **Wandering:** this state provides the robots with the ability to move around in the environment while avoiding collisions with obstacles and each other. If there is a food source within the detectable range of the robot, the robot changes its state from wandering to gathering.
- **Gathering:** the robot in this state gathers food item from the food resource. Once the robot is carrying the food item, it changes its state to homing.
- **Homing:** the robot in this state moves towards the location of the nest. When the robot reaches the nest, it changes its state to dropping.
- **Dropping:** the robot in this state drops the food item at the nest and then changes its state to wandering.

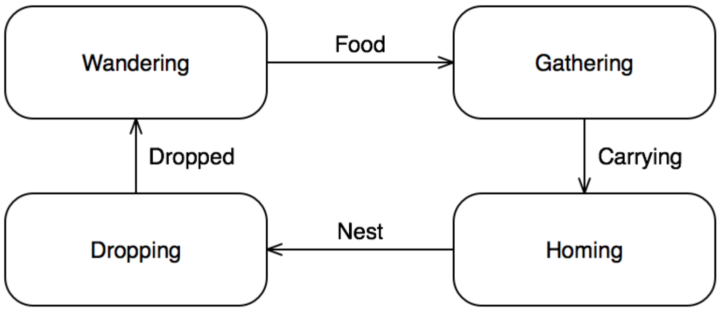


FIGURE 2.6: The finite state machine of the basic foraging algorithm

2.4.2.2 Path Finding using Pheromones

Even though there are no individual ants knowing the location of the nest, the location of the food, or even the location of itself in the environment, they can still find the shortest path from the nest to the food (Dorigo, Birattari, and Stutzle, 2006). This is because ants use pheromones to mark trails in the environment. When ants are looking for food, they wander around their nest randomly. Once they find the location of the food, they start to return to their nest while laying down pheromones trails. If other wandering ants meet the trails, they start to follow the trail instead of wandering randomly. If these ants eventually find the food by following the trail, they will lay down more pheromones to reinforce this trail while bringing the food to their nest. However, the pheromones evaporate as time elapses. The more time it takes for an ant to travel from the food to the nest, the larger the amount of pheromones evaporate. Over the course of time, the density of the pheromones will be higher on a shorter trail than a longer one. Eventually, the shortest path appears.

In terms of ant foraging behaviour in swarm robotics, if the robots are not equipped with a good path planning algorithm which allows them to reach the nest when carrying a food item, pheromone can be used to find the shortest path to the nest. By finding the shortest path in the environment, the efficiency of the system can be improved. In this situation, one of the major difficulties of this algorithm is to create pheromones in the environment. The most common way is to use sensors of the robot, e.g. chemical sensor, or use communication, e.g. a wireless network, to build the pheromone system. The following approaches are some representative examples of solving the pheromones problem:

- **Physical Mark - Evaporating Chemicals**: The agents in (Sharpe and Webb, 1998) find the shortest path with the help of alcohol (evaporating chemicals). Alcohol was laid down on the floor, and the chemical sensors on the robot are able to detect alcohol vapours, which lead the robot towards the greater concentration of vapour.

- **Visual Mark - Fluorescence Chemicals**: The agents in (Svennebring and Koenig, 2004) are able to create a trail by dropping drops of fluorescence chemicals. As the dropped chemicals emit light, visually marked trails are formed. The sensors on the robot detect the intensity of the light so that the robot can head towards the brighter trail.

- **Virtual Mark - Virtual Pheromones**: Payton et al. (Payton et al., 2001) proposed a virtual mark called virtual pheromone. Virtual Pheromones are implemented using infrared-based communication and can be transmitted from one robots to another. Virtual Pheromones are not fixed locations in the environment but are symbolic messages embedded in the robots.

Each of the above approaches has its own shortcomings. Physical marks, such as temporary marks, are difficult to implement physically. Visual marks which leaves permanent physical marks in the environment are usually not acceptable in most situations. Although virtual marks do no harm to the environment while it is relatively easy to implement, spare robots are needed because these robots are used as pheromones themselves. Fortunately, there is another approach called **Deployable Beacons**

(Barth, 2003) which uses robots to deploy beacons, which can be used as pheromones, in the environment. This would not be a problem in the simulation even though this approach has its shortcomings, for example, the robots need to be able to carry a large number of beacons and should also have the ability to retrieve beacons which have previously been deployed in the environment.

2.4.3 Task Partitioning Behaviour

In (Ratnieks and Anderson, 1999), Ratnieks et al. describe task partitioning as "*the phenomenon in which a piece of work is divided among two or more workers*". Task partitioning is observed in many species of social animals. One example of task partitioning, observed in nature, is leaf harvesting of leaf-cutting ants (Hart, Anderson, and Ratnieks, 2002). In leaf-cutting ants' foraging, some individuals cut and drop leaves from the tree to the ground while other individuals collect and transport the dropped leaves to the nest. During the whole task, each individual can choose whether to cut the leaves, or transport the leaves, or even both. The advantage of partitioning the tasks is that, once some individuals have climbed the tree, the energy consumed by climbing up and down the tree can be saved for other individuals. The disadvantage is that energy has to be used to search for the leaves on the ground. Nonetheless, some studies (Hart, Anderson, and Ratnieks, 2002; Hart and Ratnieks, 2001; FOWLER and Robinson, 1979) suggest that the gain from using task partitioning often outweighs its costs.

Swarm robotic systems face similar situations as leaf-cutting ants. In this case, it is more convenient and efficient to partition one complex task into multiple simpler subtasks. These subtasks are *sequentially interdependent* (Brutschy et al., 2014): subtasks have to be achieved in sequence so that the overall task can be accomplished. For example, in leaf-cutting ants' foraging, the leaves can only be transported to the nest after they are cut from the tree. Task partitioning is usually not fixed, these subtasks can either be tackled by different individuals at the same time, or by the same individuals at different times.

2.4.3.1 Related Work of Task Partitioning Behaviour

Task partitioning and task allocation are always a source of confusion in swarm robotics. Task partitioning is about deconstructing one complex task into multiple simpler subtasks, while task allocation focuses on the organization of the workforce. Task allocation has received a considerable amount of attention in both biology and swarm robotics. However, task partitioning is not as extensively studied in the context of swarm robotics as it is in the field of biology (Pini et al., 2014). In this section, the work on robotic (both multi-robotic and swarm robotic) task partitioning is reviewed.

In ants foraging behaviour, studies (Hart and Ratnieks, 2001; FOWLER and Robinson, 1979) show that, as the number of robots increases, the throughput of the system does not always increase, due to interference among the robots. When the size of a robotic system is small, its performance will benefit by adding robots to the system. However, experimental results in (Shell and Mataric, 2006) have demonstrated that, after the system reaches a certain size, adding more robots will no longer be of benefit but will harm its performance. This is because the robots will spend more time on non-task-relevant behaviours, such as obstacle avoidance, competition to access a shared resource, and object manipulation, when the density of the robots increases. In order to neutralize such interference, a particular strategy can be used to force the robots to either stay in different areas all the time (Schneider-Fontán and Mataric, 1996) or only use the same area at different times.

2.4.3.2 Task Partitioning in Multi-Robotic System

In study (Drogoul and Ferber, 1993), task partitioning was first introduced in multi-robotic system to reduce physical interference among the robots near the food resource and the nest. The robots form a chain between the food resource and the nest so that the food items can be passed along the chain until they reach the nest. The experimental results showed that chain formation can reduce physical interference between the robots and increase the efficiency of the system.

In study (Ostergaard, Sukhatme, and Matari, 2001), a method named bucket-brigade (another form of chain formation) is proposed to reduce interference among the robots in a multi-robotic system. Bucket-brigade forces the robots to stay in their own territory. When a robot is carrying an object, it passes this object to the following robot once it has crossed the boundaries of its territory. The object will then be passed to the next robot, and so on, until it reaches the destination. The experimental results showed that bucket-brigade displayed better performance than regular foraging in a maze-like environment.

Study (Shell and Mataric, 2006) studied bucket-brigade by varying the size of working area of each robot and showed that, when the number of robots increases, the performance of the system can be increased by reducing the working area of each robot. Study (Vaughan, 2008) improved the bucket-brigade algorithm by allowing the robots to adapt the size of their territory based on interference. This improved algorithm was shown to perform better in multi-robotic systems with a large population (up to 500 robots).

2.4.3.3 Task Partitioning in Swarm Robotic System

In swarm robotic research, task partitioning was first studied in (Pini et al., 2011b) to reduce sources of interference for swarm robotic system. The entire environment is split into two parts, harvest area and transport area. The robots in the harvest area search for food resources and deliver the food items they found to the boundary between the harvest and transport areas. The robots in the transport area then transport the food items to the nest. In follow-up study (Pini et al., 2011a), a task partitioning strategy is proposed for the robots to decide whether to partition a given task or not. In another follow-up work (Pini et al., 2014), an algorithm called the static partitioning strategy is proposed so that the robots can partition the environment themselves. In the static partitioning strategy, the robots transport the food item for a limited distance (called *partition length*) so that there is no need to demarcate the environment any more. The robots also transfer the food item directly to other robots in order to improve the performance of the task partitioning behaviour. One disadvantage of the static partitioning strategy is that the size of the environment must be known in advance in order to define a proper partition length for the swarm before the experiments. Later on, another study (Buchanan, Pomfret, and Timmis, 2016) proposed an algorithm named the dynamic partitioning strategy in which the robots have the ability to determine the *partition length* themselves. The results show that the dynamic partitioning strategy is able to converge to the *partition length* provided in (Pini et al., 2014).

2.5 Chapter Summary

Despite the great deal of attention received in recent years, current swarm robotic systems are still at a very early stage. Current swarm robotic systems are not as robust as were first thought and most of the potential real-world applications only work in the simulator. Winfield et al. (Winfield, Harper, and Nembrini, 2006) suggested a combination of analysis, design, and testing methods can be used to achieve a dependable swarm. The design method and analysis techniques of swarm robotics were reviewed and both of them are still open problems. Due to the lack of testing methods in the field of swarm robotics, testing techniques in related fields were reviewed. The successful examples of testing single autonomous robot in studies such as (Nguyen et al., 2012) motivated the author's desire to meet the need for a swarm robotic testing method.

Chapter 3

Evolutionary Testing Method for Swarm Robotic System

Following on from the literature reviewed in previous chapter, it is clear that swarm robotics research has received a great deal of attention in recent years. Swarm robotics has great potential to solve real-world problems (Şahin, 2004). However, due to the limitations of current hardware technology and behavioural control algorithms, most of the potential real-world applications only work in principle (in the simulator). There are also other open problems in the current state of development of swarm robotic research which prevent the system from being flexible, scalable, and robust. This chapter specifies one of the current open problems in swarm robotics research and propose a solution for this open problem. This chapter also introduces the simulators used for carrying out experiments in this thesis.

3.1 Problem Analysis

System development life cycle shows the process for developing a complex system, which can either be hardware only, software only, or both. During a system development life cycle, the whole system development process is split into different sub-processes. Figure 3.1 shows an example model of system development life cycle. The work in (Siciliano and Khatib, 2016) shows that system development life cycle can also be applied to the development of either a single physical robot or a multi-robotic system. There are various models of system development life cycle such as waterfall model, spiral model, v-model, prototyping (iterative-incremental) model, and so on. Each model has its own combination of sub-processes and its own workflow (the order of how the sub-processes are carried out). No matter how the models change, there are four basic sub-processes which always exist in every model of system development life cycle which are the following:

- **Requirement Analysis:** analyse the requirements of the system in order to find out the expectations of the system.
- **Design and Implementation:** design the components of the system based on the requirement. Then develop the system by realising the design.
- **Testing (Verification and Validation):** make sure that the system is built according to the design specifications and meets the system requirement.
- **Maintenance:** maintain the system according to the problems found during the operation of the system and modify the system according to requirement changes to ensure that the system does not become obsolete.

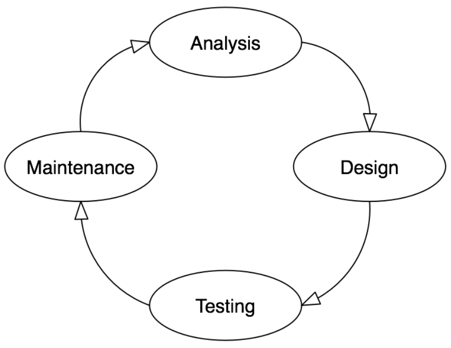


FIGURE 3.1: model of system development life cycle

Due to the lack of real-world swarm robotic applications, there is little literature which directly studies requirement analysis, testing, and maintenance. According to the literature reviewed in Chapter 2, most of the literature focuses on the design method (see section 2.1.5) and a tiny portion of the literature focuses on the analysis techniques (see section 2.3.1) of swarm robotic systems. Some of the literature mentions testing in passing but none of these present a testing method. As previously reviewed in Chapter 2, behaviour-based design method depends on the experience of the designer too much. Even though automated design methods are able to reduce the amount of work for the designers, these methods are still immature so that the best collective behaviour obtained by automation design can also be obtained by behaviour-based design method. Despite the large amount of literature which focuses on the design method, there are still no methods which are able to design the desired collective behaviour precisely. As a result, from a system development perspective, swarm robotic systems are still at an early stage, and all four basic sub-processes of the system development life cycle are open problems.

Up to the time of writing, most of the swarm literature develops a swarm robotic system as follows:

- **Requirement analysis:** as there are few successful examples of analysis techniques for swarm robotic system (see section 2.3.1). The requirement of a particular swarm behaviour is usually obtained by reviewing relevant literature or observing a related natural swarm.
- **Design and Implementation:** Propose and implement a coordinated (swarm control) algorithm (either behaviour-based or automated design).
- **Testing:** Test the developed swarm robotic system in a manual developed environment.
- **Maintenance:** maintenance is rarely mentioned.

As collective behaviour is emerged according to the design of a swarm control algorithm, most swarm robotics research is oriented to design and implementation. Even though the outcome of requirement analysis or testing is not as obvious as that of design implementation, having good methods in these two processes might provide swarm robotic researchers with a better understanding of their systems and help these researchers to figure out weaknesses of their systems. Another reason for the lack of requirement analysis methods is that most collective behaviours studied, such as aggregation, foraging, and so on, have already been heavily studied in other fields. Consequently, the demand for requirement analysis is relatively low at the time of writing.

Due to the immaturity of current swarm robotic systems, relatively simple test environments is able to make collective behaviour, i.e. flocking, fail. Most of the current swarm robotics research is carried out (tested) in empty environment (no obstacles). Even when obstacles are involved, study (Pini et al., 2011b) uses only one test environment for all experiments and study (Hoff et al., 2010) uses three test environments. For this case, the system developed is only guaranteed to work in the tested environments.

A real-world swarm robotic system should be able to deal with various challenges in different environments. Hence, systematic testing methods are required in order to build real-world applications of swarm robotic systems.

A maintenance method maintains a system in operation. As real-world applications of swarm robotic system are lacking, there is no need to develop any maintenance methods at the time of writing. All the same, alongside the development of the swarm robotic systems, the need for maintenance methods will increase in the future.

According to the problem analysis above, a testing method is the most needed system development process apart from design and implementation at the time of writing. To develop a testing method for swarm robotic systems, either an existing testing method from other fields can be adapted or otherwise a new one can be developed.

3.2 Proposed solution - Evolutionary Testing Method

An evolutionary testing method for swarm robotic system is proposed in this section. Testing process usually contains two sub-processes, which are, verification and validation (Roache, 1998). Verification guarantees that the system is built according to the system design, while validation guarantees that the system works correctly in the intended environment. During verification, the correctness of requirements, design and implementation, and any other processes carried out before testing are checked. The main objective of verification is to find out whether the developers built the product correctly. During validation, the newly developed system is tested in the intended environment (or similar environment). In this thesis, the term "testing method" only refers to the validation part of testing process in system development life cycle. Note that in some studies, such as (Blanchard, Fabrycky, and Fabrycky, 1990; Jacobson et al., 1999), testing is equivalent to validation.

As mentioned earlier, most current swarm robotic research executes the system in manually developed test environments. However, due to the autonomy of the individual robots, the quantity, diversity, and challenge required by test environment for swarm robotic systems are enormous. As manual test environment generation is costly and time consuming, it is not appropriate for testing along the path to real-world swarm robotic applications.

In the field of automated testing, there are various of testing methods such as random testing, testing using simulated annealing (Eglese, 1990), testing using tabu search (Glover, 1990), and testing using genetic algorithm (GA) (Ammann and Offutt, 2008). Random testing generates test cases/data using random search (Bird and Munoz, 1983). Experimental results in study (Forrester and Miller, 2000) show that random testing can be effective at finding bugs in software testing. However, the test coverage of random testing is very low and it is ineffective if the overall solution space is large (Godefroid, Klarlund, and Sen, 2005).

Simulated annealing uses a fitness function to guide the process of random search (Eglese, 1990). It moves from one solution to a fitter solution until the stopping criterion has been satisfied. Simulate annealing is well-known as a local search method because it only considers one solution at a time and it only searches the neighbour areas of that solution. As a result, testing using simulated annealing is not suitable for swarm robotic system due to the quantity and diversity required by the test cases. Tabu search uses a tabu list (the moves that are forbidden) to prevent the search being stuck at a local minimum (Glover, 1990) but the diversity of the solutions is not guaranteed.

GA provides an intelligent exploitation of a random search, and is widely used to solve optimisation problems (Mitchell, 1998). GA is widely used not only because of its global optimization capabilities but also because it is able to search various optimal solutions at the same time. GA have been applied to automated software testing in conventional software applications (Srivastava and Kim, 2009) and in evolving control algorithms in swarm robotic systems (Dorigo et al., 2004) but not to the time of writing in testing swarm robotic systems.

A GA evolves solutions by selecting, reproducing, and mutating a population over many generations (Holland, 1992). A good chromosome (the representation of each individual in the population), an appropriate fitness function, and appropriate GA parameters are needed to be defined before the use of a GA (Wei, Timmis, and Alexander, 2017).

3.2.1 Chromosome

The term *chromosome* in a GA refers to a candidate solution to a problem (Mitchell, 1998). Each test case is treated as a chromosome in this thesis. Various test cases contain obstacles with different layout. An obstacle can be treated as a gene in the chromosome.

A cellular representation represents an object by using its position and orientation (Gruau, 1994), see figure 3.2. It is used to represent the chromosomes in this thesis (Wei, Timmis, and Alexander, 2017) in order to keep the representation of the chromosome simple. Each chromosome is

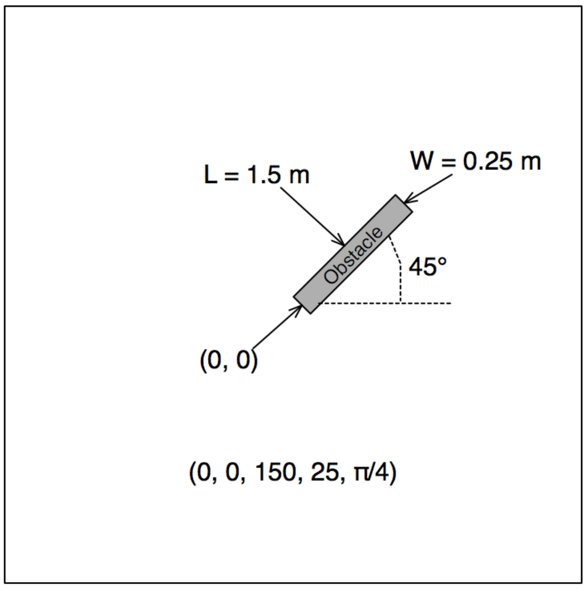


FIGURE 3.2: A bird's-eye view of a chromosome containing only one gene (obstacle). In the simulator, the height of each obstacle is set to be 1 meter.

composed of multiple genes (named single descriptor in cellular representation). Each descriptor has five parameters (x, y, l, w, o) which specify its central position (x, y) in the environment, the length l, the width w, and the orientation of the descriptor o. Note that the height of each obstacle is 1 meter. There is no need to concern much about the height of obstacles because no flying or climbing robots are used in the simulator in this thesis. Figure 3.2 shows an bird's eye view of a chromosome in which there is only one gene (obstacle). The descriptor for this obstacle is shown in the figure as $(0, 0, 150, 25, \pi/4)$. A random test case with N obstacles can be generated according to the following rules (Wei, Timmis, and Alexander, 2017):

- Randomly generate N single descriptors (obstacles).
- Process single descriptors in the order they are generated, and place a corresponding obstacle in the environment.
- If part of the obstacle is outside the edge of the environment, split this part from the obstacle, but leave the descriptor for this unchanged.
- If an obstacle is totally inside another obstacle, its descriptor will be regenerated.
- If adding the obstacle means that there are no paths between the starting point of the swarm and the destination for flocking behaviour

or no paths between the nest and the food sources for foraging and tasks partitioning behaviour, its descriptor will be regenerated.

3.2.2 Fitness Function

Fitness function is also called objective function (Haupt and Haupt, 2004). In swarm robotics research, different collective behaviours require different metrics to measure the performance of the swarm. Fitness function uses one or more metrics as inputs to calculate the fitness value, which represents the performance of the swarm in this simulation with a given chromosome. Therefore, when developing a testing method for a particular collective behaviour, metrics should first be defined either using existing metrics or by developing a new one. Note that the objective of the testing method is to reveal failures in a swarm control algorithm. This means that the worse a swarm performs in a test case the better the test case is.

When building a fitness function, either single-objective optimization or multi-objective optimization can be used. Singe objective function attempts to optimize one aspect of a problem while multi-objective function optimizes more than one aspect of the problem. For real-world swarm robotic applications, the overall performance of a system contains various aspects, such as efficiency of the system, dependability of the system, energy consumption, and so on. In these circumstances, the optimal solution for one aspect might not be the best one for another. Consequently, single-objective function is used to optimize only one aspect of the system, and multi-objective function will be the best candidate if a set of optimal solutions of multiple aspects is desired.

3.2.2.1 Fitness Landscape

In evolutionary research, fitness landscape is used to visualize the relationship between the chromosomes and reproductive success in the process of evolution (Mitchell, 1998). In evolutionary optimization problems, each chromosome represents a solution. All candidate solutions (chromosomes) of a particular evolutionary optimization problem form the land (searching space) of fitness landscape in which each solution is a single point. The chromosomes that are similar are close to each other in the fitness landscape, while those are different are far away from each other. The height of each chromosome is directly proportional to the fitness value (calculated by the fitness function) of the chromosome. As some chromosomes have higher fitness values than their surrounding chromosomes, mountains are formed in the landscape. *"The task of finding the best solution to the problem is equivalent to finding the highest mountain in the fitness landscape." (Langdon and Poli, 2013)* The highest mountain is called the global optimum, and those mountains whose fitness values are less than that of the highest mountain are called local optimum (neutral landscape). In order to converge the solutions to the global optimum, mutation operators described in 3.2.3.6 are often used to help escaping the neutral landscape.

3.2.3 Genetic Algorithm Parameters

The general framework of GA is to use a population, manipulated by selection, reproduction, and mutation operators, toward optimal solutions. So designing a GA is the process of finding good values for these parameters. The parameters in GA in this thesis are chromosome length, population size, parent selection method, crossover type, crossover probability, mutation probability, and number of generations in evolution. The following sections describe each parameter and the respective potential candidates.

3.2.3.1 Chromosome Length

Chromosome length is the number of genes in a chromosome. In this thesis, each gene of a chromosome represents a single obstacle in a test case. Consequently, the number of genes is the number of obstacles in an environment. In section 3.2.1, cellular representation is selected to represent the chromosome, but the length of each chromosome remains undecided. As in nature, a chromosome contains many genes (from hundreds to thousands), so the number of obstacles in an environment can vary from one to as many as required.

3.2.3.2 Population Size

Population size represents the number of chromosomes in a population. Many studies in the literature examine population size for GA (Goldberg, Deb, and Clark, 1991; De Jong, 1975; Grefenstette, 1986; Schaffer et al., 1989). Even though different papers have different optimal population size for GA, the range of 10 to 50 is always a subset of their optimal solutions. So the potential candidates for the population size in this thesis range from 10 to 50.

3.2.3.3 Parent Selection Method

Parent selection method is about how to select the chromosomes from a population for later reproduction. It is a crucial process in GA as good parents might reproduce better off-spring which can lead to optimal solutions. The following are three commonly used parent selection methods for GA and are treated as the potential candidates in this thesis (Whitley, 1994):

- **AllParent-BestHalf:** All the individuals in the population become parents and are used to mate and recombine to reproduce the same amount of off-spring as themselves. The new generated off-spring are merged with their parents to form a new population. Half of the individuals with lower fitness value in the new population are then abandoned, and the other half are kept in the population for later breeding.

- **2BestParents-2WorseAway:** In this method, only the two fittest individuals are selected as the parents and they produce two children. The new created children are placed in the population to form a new population. The two individuals with the lowest fitness value are thrown away, and the rest are preserved for later breeding.

- **2RandomParents-2WorseAway:** Two individuals are randomly chosen as the parents and then reproduce two children. The new reproduced children are merged with the original population to form a new population. The two individuals with the lowest fitness value are put away, and the rest are kept for later breeding.

3.2.3.4 Crossover Type

Crossover is a genetic operator which use the chromosomes selected by parent selection for reproduction. It is the process of reproducing later generations by using genes from selected parents. At the time of writing, the following are three kinds of crossover techniques which are often used in GA (Whitley, 1994):

- **Single Point Crossover:** Both chromosomes of parents are cut at a randomly chosen point and the sections after the cuts swap to form two new children.

- **Double Point Crossover:** Both chromosomes of parents are cut at two randomly chosen points and the sections between the cuts swap to form two new children.

- **Uniform Crossover:** Each gene of a child is randomly chosen from one or the other parent sequentially.

Experiments are used to choose to move between crossover types according to the method described in section 3.2.4. From the experimental results, whether the crossover operator is needed or not can be decided. In all three cases studies in this thesis, the experimental results show that the crossover operator is needed in order to converge to optimal solutions.

3.2.3.5 Crossover Probability

Crossover probability means how often crossover is likely to be performed (Whitley, 1994). If no crossovers occur, next generated off-spring are exact copies of their parents. If a crossover occurs, off-spring are reproduced using the chromosome of their parents. The probability of crossover can vary from 0% to 100%. If crossover probability is 100%, all the individuals in new population are reproduced from previous generation (no old individuals are left). If the probability is 0%, no crossovers are performed.

3.2.3.6 Mutation Probability

Mutation alters one or more gene values in a chromosome from its initial state (Mitchell, 1998). Mutation probability means how often genes are likely to be mutated. The benefit of mutation is to prevent the GA being trapped in a local optimum (Mitchell, 1998). Mutation probability also varies from 0% to 100%. If the probability is 100%, all of the genes in a chromosome are altered. If the probability is 0%, nothing is changed.

3.2.3.7 Number of Generation

The number of generations before the evolution terminates which can vary from 1 to a particular number. In this thesis, the number of generations is set to a constant number according to the parameter analysis (section 3.2.4) executed before the experiments.

3.2.4 Parameter Analysis

Designing a GA is the process of finding good values for GA parameters. A parameter robustness technique from Spartan (Alden et al., 2013; Alden et al., 2014) is employed in this thesis which assesses the sensitivity of parameters in the simulation. This technique investigates the sensitivity of each parameter by changing the values of this parameter while keeping the value of all other parameters the same. If its sensitivity is high, experimental results for different values of each parameter are compared to determine at which value of the parameter the GA performs best. Otherwise, changing this parameter does not lead to a statistically significant behavioural alteration of the GA.

The robustness technique is performed during parameter analysis by perturbing parameters individually using a 'one at a time' approach (Alden et al., 2013). When evaluating the different values of one parameter, all of the other parameters remain unchanged. Twenty evolutions are executed for each candidate value of each parameter. The number of 20 is chosen by using the Consistency Analysis technique provided by Spartan (Alden et al., 2013). The Consistency Analysis technique is designed for determining the number of repetitions of the experiments that is needed to be carried out in order to reduce the uncertainties of the experimental results in the simulation. In order to analyse one parameter of a testing method with 10 test cases as its initial population, there are 10,000 evaluations in order to carry out 20 evolutions if each evolution contains 50 generations. For parameters such as population size and number of generations, it is infeasible to evaluate all potential candidates. Accordingly, only a few candidates (the increment is 10) are selected and evaluated for population size and number of generations. The final results for all potential candidates are compared using the Vargha-Delaney A-Test (Vargha and Delaney, 2000) to determine if a statistically significant behavioural alteration has occurred, and if yes, then the candidate which performs best is selected.

Different swarm robotic control algorithms behave diversely, therefore, the values of parameters of different swarm control algorithms vary. For each of the following case studies in Chapter 4, 5, and 6, the values of each parameter will be analysed according to the fitness function defined for each swarm behaviour.

3.2.5 Evolutionary Process of Test Cases

The following procedure shows how to generate a test case using the genetic algorithm (Wei, Timmis, and Alexander, 2017):

- **Step 1:** Initialize population: randomly generate an initial population using the rules stated in section 3.2.1 for generating test cases;
- **Step 2:** Compute fitness: evaluate the fitness value of each test case;
- **Step 3:** Select parents: follow the parent selection method to choose parent test cases;

- **Step 4:** Crossover: Use a crossover procedure to produce offspring. If the offspring produced do not contain a clear path from the starting point to the destination, redo Step 4;
- **Step 5:** Mutate: Allow the offspring to mutate with mutation probability p_m. If the offspring produced do not contain a clear path from the starting point to the destination for flocking behaviour or from the nest to food sources for foraging and task partitioning behaviour, redo Step 5;
- **Step 6:** Check for termination: Terminate the algorithm if N generations have been run (N is determined by applying parameter analysis). Otherwise, go to Step 2.

3.3 Experimental Infrastructure

The ideal situation for testing swarm robotic system is to implement the whole testing system on real robots in the physical environment. However, there are some constraints which make the ideal situation impossible at the state-of-the-art at the time of writing this thesis. The most significant constraint is that running experiments with real assets is expensive. In order to test a physical swarm, money not only needs to be spent on the robots and the tracking systems like regular swarm robotics research; more also needs to be spent on building the physical environment. One key aspect of the testing method presented in this thesis is the various shapes of obstacles. For example, if the maximum length and width of an obstacle is 2 meters and 0.5 meters respectively, there will be 20 different lengths and 5 different widths for a 10 cm increment. As a results, one hundred obstacles of different shapes are required. Moreover, if there are 5 obstacles in each test case, a total number of 500 obstacles are required in case of certain test cases contain 5 obstacles of the same shape. This might be acceptable for some large and well-funded projects, but it is too costly for this project.

There are several advantages derived from using simulator in the study of swarm robotics. Firstly, data collection is easier and faster in simulations as a monitoring mechanism is easier to build in the simulator and experiments can be simulated faster than in real-time. In addition, robots which are currently impossible to build in the physical world due to the cost or current technique can be developed in simulators. Lastly, test cases in the simulator are easier to build and there are no risks of damaging the hardware platform (both robots and the environments) while carrying out the experiments. Consequently, all the experimental work in this thesis, is carried out in simulation in order to show that the evolutionary testing method works in principle.

Future work which is beyond the scope of this thesis can be carried out to find out whether the evolutionary testing method proposed in this thesis is suitable for physical robots. The work in (Vargas et al., 2014) shows that the swarm robotics research carried out in the simulation has the potential to perform as good as it does in the simulator. The incremental design process proposed in (Vargas et al., 2014) is said to be *"surprisingly robust, and hence more likely to be robust enough to cross the reality gap" (Vargas et al., 2014)*.

3.3.1 ARGoS Simulator

Pincicroli et al. (Pinciroli et al., 2011) presented a multi-robot simulator named ARGoS. ARGoS is open-source and is developed using C++. It is specifically designed to simulate large swarms of robots and is still being maintained by its creators at the time of writing. The testing method presented in this thesis is implemented in ARGoS.

There were two other candidates for the simulator at the beginning of this project, which are, Player/Stage (Gerkey, Vaughan, and Howard, 2003) and V-Rep (Rohmer, Singh, and Freese, 2013). Due to the following advantages, ARGoS was chosen as the simulator for conducting the experiments in this thesis:

- **Easy to install:** The process of the installation of ARGoS is relatively simpler and less time-consuming than Player/Stage. ARGoS can be easily installed by following the instructions on the official website of ARGoS in less than one day. However, for Player/Stage, more than 20 dependencies (software which is the required by Player/Stage) are required to be installed before carrying out the installation of Player/Stage. Even all the dependencies are installed in the correct version, there is still a high probability that the installation of Player/Stage may fail. The installation of Player/Stage is a process of trial-and-error. The duration for installing Player/Stage may varies between a few days to a few weeks.

- **Easy to maintain:** ARGoS is easier to maintain than Player/Stage. As long as the version of ARGoS is unchanged, updating its dependencies or the operating system does not affect the execution of ARGoS. However, any changes of the dependencies or the operating system may affect the execution of Player/Stage. If this happens, Player/Stage has to be reinstalled in order to execute correctly.

- **Support large-scale swarm:** The benchmark results in (Pinciroli et al., 2011) show that ARGoS can perform physics-accurate simulation involving thousands of robots in a fraction of real time. However, the performance of V-Rep degrades fast when the number of robots increases. Benchmark experiments were carried out in this thesis on both ARGoS and V-Rep on the same computer under same condition (same operating system, same CPU, same RAM, and same Graphics card). When the number of robots reaches 5 in a 6m$\times$6m square area, the simulation in ARGoS runs 1.5 times faster than real time while that in V-Rep is around 2 times slower than real time (the time required for simulating 1 second in V-Rep is equal to 2 seconds in the real world). When the number of robots reaches 10 in a 6m$\times$6m square area, the simulation in ARGoS runs 1.3 times faster than real time while that in V-Rep is around 9 times slower than real time.

There are also a few disadvantages when using ARGoS (version 3.0.0 - beta29). First of all, global information in not available in ARGoS. The author of ARGoS (Pinciroli et al., 2011) states that the reason for denying global information is to prevent users cheating, i.e. to achieve emergent behaviour in ARGoS by sharing global information in the swarm. Secondly, there are no monitoring mechanisms or statistical tools for observing and analysing the performance of the behaviours of the swarm. Two external assistance tools are developed (see section 3.3.3) to address those disadvantages.

Note that version 3.0.0 - beta29 of ARGoS is used in this thesis. The next version 3.0.0 - beta 30 was released 5 months later after the release of beta29. After upgrading ARGoS to version-beta30, the external assistance tools developed in this thesis (see section 3.3.3) did not function correctly. As the external assistance tools are not fixed in beta30 for a few weeks, beta29 was used again. Later on, it turned out that version-beta30 was faulty. Five more versions of ARGoS (from beta31 to beta35) were released in 12 days attempting to fix the fault (about 2 months after the release of beta30). However, due to the undesirable experience with beta30, beta29 is used throughout this thesis. Up to the time of writing, the external assistance tools have never been upgraded on any versions of ARGoS later than beta30.

3.3.2 Foot-bot robot model

Two different robots, foot-bot and eye-bot, were modelled in version 3.0.0 - beta29 of ARGoS(Pinciroli et al., 2012) (More robots, such as hand-bot, and e-puck were modelled later). Here foot-bot is used to meet the requirement of this thesis. There is a large range of sensors and actuators which can be used on foot-bot to achieve various tasks in ARGoS. The sensors or actuators which are used in this project are now listed (Pinciroli et al., 2012):

- **Beacon** - A LED which is positioned at the top of the robot body;
- **Distance Scanner** - detects objects around the foot-bot (long range);
- **Proximity sensor** - detects objects around the foot-bot (short range);
- **Light Sensor** - allows foot-bot to detect light sources in the environment;
- **Omnidirectional Camera** - allows foot-bot to detect light bulbs (LEDs);
- **Range-and-bearing system** - allows foot-bot to perform localized communication. If an obstacle or another robot is between two robots, the robots can not communicate;
- **Ground Sensor** - reads the colour of the floor;
- **Gripper** - allows foot-bot to grip other objects, such as food resource;
- **Wheels** - for each foot-bot, there are two sets of wheels and tracks which are called *treels*. These wheels allow the foot-bot to move around in the environment.

In later chapters, different combinations of these sensors and actuators are used to achieve various swarm behaviours required in each cases study. The testing method identifies weaknesses in swarm control algorithms listed in section 2.4 by reveal failures during the execution of swarm robotic systems. In swarm robotics, the cause of a failure varies. Not only weaknesses in the swarm control algorithm can cause failures; issues, such as noises in sensors or actuators, and partial or total failures of the individuals can also trigger failures. The major disadvantage for ignoring individual failures and noises in the simulation is that the "reality gap" might be increased between the simulation and the real world. However, ignoring individual failures and noises helps the testing method focus on identifying weaknesses in swarm control algorithms. Therefore, individual failures and noises of sensors and actuators are ignored in this project.

3.3.3 External Assistance Tools

Two disadvantages of using ARGoS were mentioned in section 3.3.1. In order to address those disadvantages, two external assistance tools have been developed in this thesis. The first tool is named Observer and it monitors and records the behaviour of the entire swarm. The second tool is named Headquarters and it makes global information available to all the robots in the swarm. Note that in section 3.2.4, an analysis tool called Spartan is described that can analyse the performance of the swarm behaviours.

3.3.3.1 Observer

Observer is an external assistance tool written in C++ to monitor and record the movements of each robot in the swarm. In ARGoS, the robots have no access to any global information. They can only explore the environment using the sensors provided and have no idea about their own exact positions in the environment. This would not be a problem if the objective of using ARGoS were to design and implement a swarm control algorithm, but this makes it difficult for the testing method in this thesis to analyse the overall behaviour of the swarm. The Observer records the position, facing direction, and moving speed of each robot throughout each experiment, data which is then used for analysis later.

3.3.3.2 Headquarters

Headquarters is an external assistance tool written in C++ in order to share global information among the robots of the swarm. In swarm robotics research about control algorithms, sharing global information among the robots is usually prohibited. One of the most important features of swarm robotic system is that the robots involved should only have local and limited sensing and communication abilities. Sharing global information makes the coordination between the robots centralized, and therefore the robotic system is no longer a swarm.

After deploying Headquarters in the simulator, the whole simulator operates as a LAN (Local area network). The role of Headquarters is the same as the role of the server in client-server computer system. In ARGoS, the range-and-bearing system enables localized communication for the robots. The robots can only send and receive data in a limited range while there are no objects (either obstacles or other robots) between the sender and receiver. In this project, the range-and-bearing system is modified so that it is able to receive data from the Headquarters. Headquarters

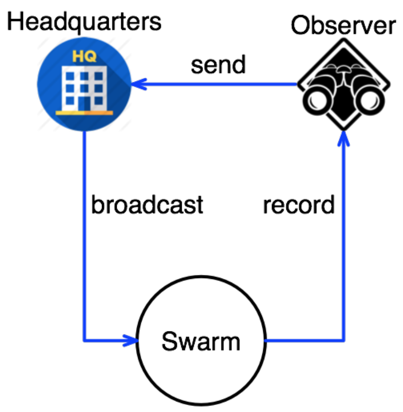


FIGURE 3.3: A model of how Headquarters and Observer operate.

first receives information of the entire swarm from the Observer and then broadcasts related information to each swarm members according to its requirement. Figure 3.3 shows a model of how Headquarters and Observer operate.

At the early stage of this thesis, Headquarters is used to create a swarm-like behaviour in ARGoS which is used as a baseline behaviour for detecting failures. All the swarm control algorithms used in this thesis do not share global information and do not have global communication abilities. For example, in Chapter 4, Headquarters only broadcasts the global information to the "prey" robot (see section 4.1).

3.3.4 Test Case Generator

The test case generator is the core program of the testing method presented in this thesis which is used to generate test cases. It contains two parts, the first named Streamer and the second Operator. Streamer reads and writes environment files of ARGoS (file extension is .argos) and is developed in Java. Streamer was written in C++ in the early stage of this project. During the optimization phase of this project, different libraries in both C++ and Java were used to develop Streamer. After several comparisons, the one developed in Java was found to be the most efficient and stable one and it is used throughout the rest of the thesis. Operator is developed in C++ and is used to generate test cases. Operator can either generate test cases randomly or take a set of test cases as input and then apply the genetic algorithm to the input in order to create a set of test cases as output. The github link for test case generator is https://github.com/hw967/EvolutionaryTestingMethod and the setup instructions are introduced in "README" file. Figure 3.4 shows a model of the working procedure of the test case generator. The procedure of how the test case generator works is described as follows:

- **Step 1:** Operator randomly generates a set of test cases as the initial population;
- **Step 2:** Streamer creates the environment files (.argos file) according to the test cases and imports those files into ARGoS;
- **Step 3:** The swarm is executed in test cases (environment files) in ARGoS and the experimental data are recorded by Observer;
- **Step 4:** The ARGoS sends environment files with the corresponding experimental results to Streamer;
- **Step 5:** Streamer transfers the environment files to test cases and then sends them to Operator;
- **Step 6:** The Operator applies step2 to step 5 of evolutionary process of test cases in section 3.2.5 to the test cases in order to generate the offspring;
- **Step 7:** Terminate the algorithm if N generations have been run. (N is determined during parameter analysis phase) Otherwise, go to Step 2.

3.3.5 Cluster Computing

As many of the experiments in this thesis can be carried out in parallel, the use of cluster computing considerably increases the efficiency of the testing method. In this thesis, the York Advanced Research Computing Cluster (YARCC) is used to carry out the cluster computing.

The most time-consuming step in test case generator (see section 3.3.4) is step 3, that is, testing the swarm in each test cases in ARGoS. The whole execution time of each generation can be reduced by executing the experiments in parallel in cluster computing. For example, if the population size is 20, 20 independent experiments need to be executed in order to record the performance of the swarm in all 20 test cases. These experiments can be executed in the same time as independent programs in cluster computing. As a result, the overall execution time of test case generator will be shorter by a large margin.

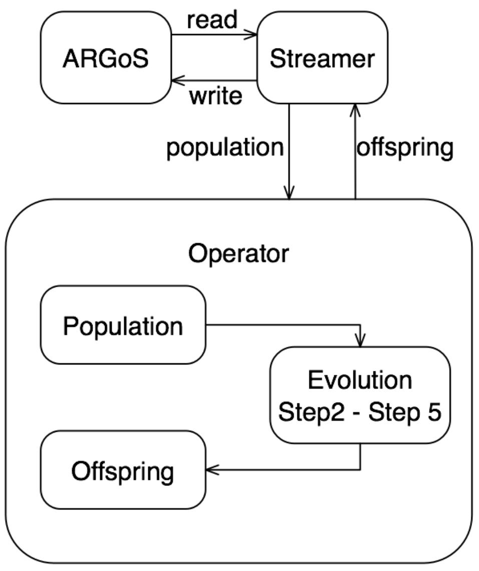


FIGURE 3.4: A model of the working procedure of the test case generator. GA step2 to step 5 is listed in section 3.2.5.

3.4 Chapter Summary

This chapter proposes a solution to the testing problem in current swarm robotics research. The testing method proposed generates swarm test environments by using GA. The chromosome, fitness function, and parameters of the testing method are defined. A robustness technique from Spartan (Alden et al., 2013; Alden et al., 2014) is adapted to conduct parameter analysis for the testing method. The experimental infrastructure such as the simulator, external assistance tools, and test case generator are introduced at the end of this chapter. Moreover, cluster computing can be used to increase the efficiency of the testing method.

Chapter 4

Testing Method for Flocking Behaviour

In this chapter, the method for testing flocking behaviour is presented. This chapter shows the procedure by which the testing method was developed step by step and therefore can be used as a guideline for applying this testing method to test other swarm behaviours. This chapter begins with the representation of the experimental scenario for testing flocking behaviour. The metrics for measuring the performance of flocking behaviour in swarm robotic systems are proposed in the next section. It then moves on to talk about the control algorithm used to achieve flocking behaviour in the simulation. Failure classification will be discussed to show how the failures discovered by evolved test cases can be sorted into different categories. Following this is a discussion of how the genetic algorithm can be used for generating test cases for the swarm. Parameter analysis and experimental results for testing flocking behaviour are presented. Finally, an example of swarm control algorithm improvement is shown at the end of this chapter.

4.1 Experimental Scenario for Testing Flocking Behaviour

In the first case study, the evolutionary testing method is used to identify weaknesses of the flocking control algorithm in swarm robotics. In section 2.4.1, the basic rules of flocking behaviour are introduced. In this section, the experimental scenario for testing flocking behaviour is represented.

The experimental scenario is designed based on the second flocking scenario discussed at the end of section 2.4.1.1. For each experiment, two types of robots are involved. One type is called the *"prey"* and the other type is called the *"bird"*. The experimental scenario can be treated as the situation which a flock of birds are preying on one prey, such as insect pest. At the beginning of each experiment, one prey and a certain number of birds are deployed in one corner of the environment. When the experiment begins, the prey moves towards the opposite corner while avoiding obstacles. The task for the birds is to flock together and follow the prey until the prey reaches the destination (the opposite corner). Due to the rule of no global information should be shared for swarm robotics, the birds are unable to see the prey if the prey is beyond the range of the sensors of the birds. In such circumstances, the birds should follow other birds.

4.1.1 Total Failures

One of the criteria for distinguishing swarm robotics research is that the system should have a large number of robots. However, there are no formal definitions for the minimum number of robots in order to form a swarm and it is difficult to justify the lower bound for the number of robots in a swarm robotic system. In study (Şahin, 2004), Erol Sahin declares that most researchers would accept that the lower bound for the group size of a swarm should be 10 to 20. There are many studies in swarm robotics that use a swarm with a size smaller than 10; for example, only 7 robots are treated as a swarm in study (Çelikkanat and Şahin, 2010), and only 6 robots are used in study (Pini et al., 2014).

In this case study, it is assumed that the minimum number of robots in order to form a swarm is N and N can be assigned any value from 2 to as many as the total number of robots in the environment. If there are fewer than N robots in the cluster, this cluster should not be treated as a swarm. When the prey reaches the destination, if the number of the robots which are following it is less than N, a total failure occurs.

4.2 Metrics for flocking behaviour

There are various studies around flocking behaviour, but most of them only develop algorithms that produce a flocking behaviour and then test the behaviour through visual observation of the swarm (Kwong and Jacob, 2003; Lindhé, Ogren, and Johansson, 2005; Olfati-Saber, 2006). A few studies (Turgut et al., 2008; Antonelli, Arrichiello, and Chiaverini, 2008; Gu and Hu, 2008) define metrics for measuring the performance (quality) of flocking behaviours, but there are no formally defined metrics for measuring the performance of the system. Moreover, there are no formal definitions of what is a good flocking behaviour. Based on several papers, for example (Moeslinger, Schmickl, and Crailsheim, 2011; Olfati-Saber, 2006;

Xiong et al., 2010), a good flocking behaviour is concluded to have at least some of the following properties (Wei, Timmis, and Alexander, 2017):

- 1) The agents of the swarm should always face in approximately the same direction;
- 2) The agents or flocks that meet (within the detectable range) should stay together;
- 3) The swarm should neither lose agents nor separate into different swarms;
- 4) The agents should not collide with each other or with obstacles;
- 5) The agents should be able to follow or move towards a target (optional).

In order to develop metrics for assessing the performance of a flocking swarm, the collective behaviour at the swarm level needs to be evaluated. As no individual levels are evaluated, failures of single agents, such as motor failures, communication failures, control system failures and so on, are ignored when carrying out experiments in the simulation. Agents in the swarm are programmed (in the simulator) not to be damaged by colliding with other agents or the obstacles, and therefore property 4 is ignored in this study. In the simulation, there are collisions among the agents and the obstacles and both agents and obstacles can block the way of an agents. On such an occasion, only the damage caused by collision is ignored but not the collisions occurred during the experiments.

In this case study, three metrics are used: angular order to take into account property 1, the proportion of the robots remaining in the swarm to take into account properties 2-3, and total failures occurred to take into account property 5. One of the keys to success when developing a testing method is a good selection of metrics. If there are no formal metrics for a swarm robotic system, developing metrics is a trial-and-error process. Different combinations of various metrics are attempted by carrying out experiments until an effective combination occurs. During the development of metrics in this case study, the following metrics were also attempted:

- social entropy proposed in study (Shannon, 2001)
- cohesion radius proposed in study (Gu and Hu, 2008)
- the deviation energy proposed in study (Antonelli, Arrichiello, and Chiaverini, 2008)

- the average speed of the swarm
- the maximum speed of the swarm
- the time needed to reach the goal

Note that the combination of angular order, the proportion of the swarm left and total failure is not the only effective one, but it is more effective than the other combinations that have been attempted. In study (Wei, Timmis, and Alexander, 2017), a combination of angular order, social entropy, and the time needed to reach the goal was used to measure the performance of the flocking behaviour. By comparing the experimental results of these two combinations, the one used in this thesis represents the performance of the flocking behaviour more accurately than the one used in study (Wei, Timmis, and Alexander, 2017).

4.2.1 Angular Order

The equation which is used for calculating the angular order of a flocking swarm is proposed in this section. The angular order of a swarm can be used to indicate whether the agents are moving in the same direction (Ferrante et al., 2012). In nature, birds in the same flock always face in approximately the same direction. The flock is more likely to be in a steady state if all its agents are moving in a similar direction. Hence, one important metric for evaluating flocking behaviour is whether the agents are moving in the same direction. In (Mogilner and Edelstein-Keshet, 1996), a mathematical model is proposed for the measurement of the angular order of self-aligned objects. By combining it with the model proposed in (Vicsek et al., 1995), an equation for calculating the angular order (ψ) of a group of objects can be derived - see equation 4.1 (Wei, Timmis, and Alexander, 2017):

$$\psi = \frac{1}{N}\left|\sum_{n}^{N} e^{i\theta_n}\right| \tag{4.1}$$

where N is the total number of objects in the group, θ_n is the facing direction of the n^{th} object in the group, where $\theta \in [-\pi, \pi]$, and i is the imaginary unit (complex number, in which $i^2 = -1$). In a two-dimensional case, the angle describing the facing direction of an object is θ and $\theta \in [-\pi, \pi]$.

The value of the angular order can vary between 0 and 1. In Figure 4.1 (a), all the agents are moving parallel and facing the same direction. This indicates that the group is perfectly aligned and is in a completely ordered state, therefore the angular order for this situation is 1. In Figure 4.1 (b),

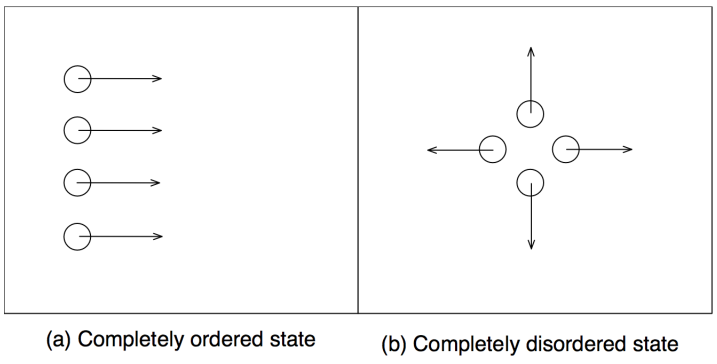


FIGURE 4.1: Angular Order

the agents are facing totally different directions. This indicates that the group is in a completely disordered state, and therefore the angular order for this situation is 0. If the angular order is low for a swarm, it is very likely that the swarm might lose some of its agents within a short time.

4.2.2 The Proportion of the Swarm Left

The proportion of the swarm left indicates the ratio (remains/total) of agents remaining in the swarm when the prey robot reaches the goal. In swarm robotic research, the major goal of most flocking studies (all of the flocking citations above) is to make the agents move in one cluster from the beginning of the experiment to the end. The swarm should lose no agents during a flocking and any agents met should be able to flock together afterwards. In this case study, the proportion of the swarm left is used to measure the cohesion and goal-seeking abilities of the flocking control algorithm.

4.3 Flocking Control Algorithm

The control algorithm used to achieve flocking behaviour in ARGoS is discussed in this section. At the early stage of this thesis, flocking behaviour was achieved in ARGoS by using both methods mentioned in section 2.4.1.2. However, a group of robots with global information is not treated as a swarm robotic system in this thesis, therefore the flocking algorithm used should not share global information.

In current swarm robotics research, there are no flocking control algorithms that are able to follow an informed robot while avoiding obstacles in the environment without any failures. In this case study, the flocking algorithm is developed based on the one proposed in (Moeslinger, Schmickl, and Crailsheim, 2011). The agents have no access to global information in this algorithm. As this algorithm only functions correctly in an empty environment and is unable to move towards a goal, obstacle-avoidance and goal-seeking mechanisms are added to this algorithm in this case study.

The foot-bots in ARGoS are able to emit light of different colours using the LEDs on the top of the robot. The omni-directional camera on foot-bot can locate the position of a light source with respect to the centre of it. In order to distinguish between the robots and the obstacles, all foot-bots emit light using the LEDS so any objects without a light source will be treated as an obstacles. The robots avoid the obstacles by moving away from the obstacles while moving towards the average position of their flockmates. In order to avoid accessing global information, the range of the omni-directional camera is limited so that each robot can only detect the position of its flockmates which are within its detectable range. Note that at the end of section 2.1.1, Scalable Swarm Robotic and Minimalist Swarm Robotics were discussed. Scalable Swarm Robotics in this thesis is used as an omni-directional camera is added to each robot.

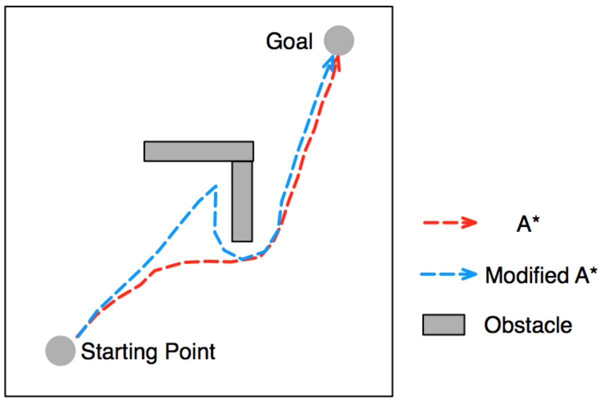


FIGURE 4.2: A comparison between the path found by A* algorithm (red) and the path found by modified version of A* algorithm (blue).

4.3.1 The Control Algorithm for The Prey

This section represents the control algorithm for the prey (the robot which is being preyed) discussed in section 4.1. Before each experiment starts, modified version of A* algorithm is used to calculate a clear path from the starting point to the destination for the prey. Figure 4.2 shows two paths found by A*algorithm and the modified version of A* algorithm. The path found by modified version of A* algorithm is more realistic as the prey keeps moving towards the destination unless encountering an obstacle. The prey will follow the clear path to move towards the destination when the experiment starts. The LEDs on top of the prey emit a different colour from the robots. If the prey is in the detectable range, the birds move towards the position of the prey. If not, the birds move towards the average position of their flockmates. If there are no other robots in the detectable range, the robots wander around in the environment. By testing the swarm using the improved flocking algorithm in empty environment, the experimental results show that the swarm is able to reach the goal without losing any individuals.

4.4 Failure Classification

The objective of the testing method presented in this thesis is to reveal failures during the execution of a swarm. A criterion of identifying the diversity of the failures found is required. Winfield (Winfield and Nembrini, 2006) addressed the failure modes of a single robot in swarm robotics, but few papers discuss classifications of failures for overall swarm behaviour.

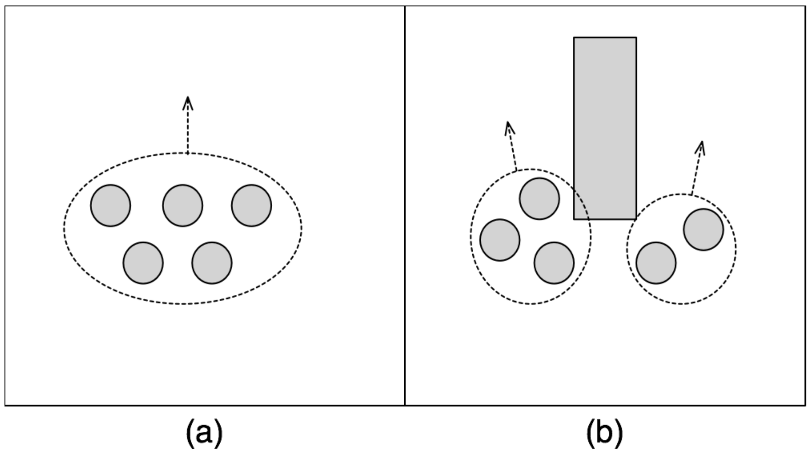


FIGURE 4.3: An example of how a swarm is split into two clusters.

A failure classification is proposed according to the various causes of failures. In this case study, the main reason for a swarm splitting up is the obstacles. The velocity (both speed and moving direction) of a robot is affected once encountering obstacles. The robot might lose track of the rest of the swarm while it attempts to avoid obstacles. The failure classification is therefore in terms of how robot speeds and directions vary when a swarm fails by splitting into clusters.

When a swarm is split into two clusters, the speeds and moving directions of those two clusters are compared. The split is classified in terms of relative speeds and relative angles. In this case study, two speed categories (whether or not the slower cluster is moving at more than half of the faster cluster's speed) and ten angle categories (in 10-degree increments until 90 degrees, and then 90 degrees plus) are used. At a result, there are 20 different categories of failures. Figure 4.3 shows an example of how a swarm is split into two clusters. In Figure 4.3 (a), five agents are moving together as a flock. The dashed circle around these five agents indicates that these agents are in the same cluster. The dashed arrow indicates the velocity (the average speed and moving direction of all the agents) of the current cluster. Figure 4.3 (b) shows the moment when one cluster is split into two clusters. The relative angle and relative speeds can be determined by comparing the velocities of those two new clusters. Note that only the last cluster which is split from the swarm is taken into account.

4.5 Evolutionary Testing method

This section shows the procedure of developing a testing method for flocking behaviour. Following the framework proposed in section 3.2, chromosome and fitness function are first designed. Then the parameters of the GA are analysed according to the new designed chromosome and fitness function by following the procedure proposed in section 3.2.4.

4.5.1 Chromosomes

A chromosome is a test case in which the swarm is tested. According to scenario 2 listed in section 2.4.1.1, the agents are deployed near the upper-left corner of the environment when the experiment starts. The prey robot moves towards the goal following the clear path (see section 4.3.1). The rest of the swarm attempts to follow the prey robot until the prey robot reaches the goal. In this scenario, the only variable is the shape and layout of each obstacle. Cellular representation is used to represent the chromosome and a random test case with N obstacles can be created using the rules listed in section 3.2.1.

4.5.2 Fitness Function

The fitness function (described in section 3.2.2) uses a combination of the metrics defined in section 4.2 to measure the performance of a flocking control algorithm. Note that the objective of the testing method is to identify faults in a swarm control algorithm, therefore, the worse a swarm performs in a test case, the better the test case is.

For each experiments, the swarm attempts to reach the destination by following the prey without losing any individuals. In section 4.2, two metrics for measuring the performance of a flocking control algorithm are defined. The fitness function is defined by using those three metrics as follows:

$$FV = \begin{cases} \frac{(1+\psi)R}{2T} & \text{for} \quad R - N \geq 0 \\ 0 & \text{for} \quad R - N < 0 \end{cases} \tag{4.2}$$

where ψ is the angular order of the swarm, R is the number of agents remaining in the swarm, T is the total number of agents in the environment, and N is the minimum number of agents in order to form a swarm.

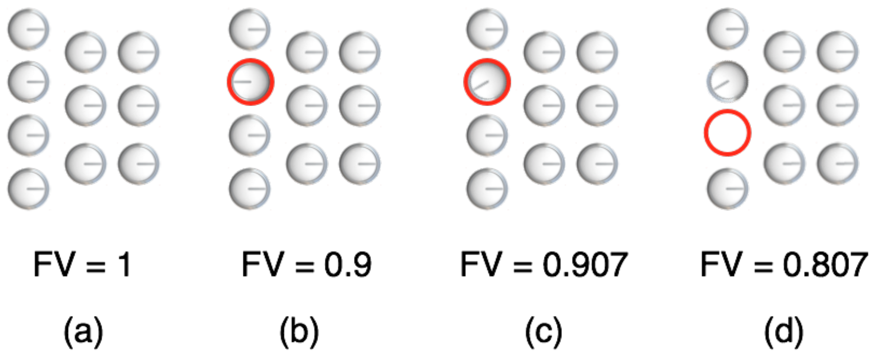


FIGURE 4.4: Fitness values for four different orientations of a swarm. The white dot with grey boundary represents a robot. The grey line in the robot shows the facing direction of the robot. The total number of robots in the environment, T, is 10. There are 10 robots in (a), (b), and (c), and there are 9 robots in (d). The red circle highlights the robot that varies compared to the swarm on the left.

Figure 4.4 shows the fitness values for four different orientations of a swarm. The fitness value ranges from 0 to 1. When the fitness value is equal to 1, all the robots are facing the same direction and no robots get lost (see Figure 4.4 (a)). When a total failure occurs, the fitness value is 0. The red circle in Figure 4.4 highlights the robot that varies compared to the swarm on its left. For example, the robot with red circle in Figure 4.4 (b) is facing left while it is facing direction in Figure 4.4 (a) is right. As this single robot turned 180 degree in the swarm, the fitness value drops from 1 to 0.9. Figures 4.4 (b) and 4.4 (c) shows the reflection on the fitness value when one robot only turns 30 degrees. Figure 4.4 (c) and figure 4.4 (d) show how the fitness value changes when one robot gets lost. By carrying out experiments on the fitness value, the experimental results show that it is very

TABLE 4.1: PARAMETER VALUES OF GENETIC ALGORITHM (Wei, Timmis, and Alexander, 2017)

Name of Parameter	Analyzed Candidates	Best Candidates
Number of genes	[4, 6, 8, 10]	8
Population size	[10, 20, 30, 40, 50]	20
Parent selection method	AllParent-BestHalf 2BestParent-2WorseAway 2RandomParents-2WorseAway	2BestParent-2WorseAway
Crossover type	Single-point-crossover Double-point-crossover Uniform-crossover	single-point-crossover
Crossover probability	[0.1, 0.2, 0.4]	0.1
Mutation probability	[0, 0.05, 0.1, 0.2]	0.05
Number of generation	[10, 30, 50, 100]	50

sensitive (the fitness value changes even when a robot turns only 1 degree) and is able to reflect the actual performance of the swarm (the higher the fitness value, the better the swarm performed).

4.5.3 Parameter Analysis

Parameter analysis can be applied once the chromosome and fitness function are defined. The robustness technique in Spartan (Alden et al., 2013) is adopted to decide the value of each parameter. In section 3.2.4, Parameter Analysis, 7 parameters are discussed and they are analysed one by one. While analysing one parameter, all other parameters remain unchanged. Due to the huge number of evaluations required, cluster computing (YARCC in this thesis) is recommended for executing parameter analysis. Table 4.1 shows the parameter values determined according to the analysis from Spartan (Wei, Timmis, and Alexander, 2017). All the parameter values in table 4.1 are only optimized for a 6m×6m environment with a swarm of 10 foot-bots. Figure 4.2 can also be considered as an example of a test case for testing flocking behaviour.

4.6 Experimental Results

In this section, the performance of the evolutionary testing method is measured and compared with that of the random testing method. A better set of test cases should not only find more failures occurring (or more severe failures), but also find more types of failure. Consequently, the performances of the testing methods are compared in terms of the following two aspects: the severity of the failures found and their diversity. Some parts of this section was shown in study (Wei, Timmis, and Alexander, 2017).

The fitness function used in this chapter is different from the one used in (Wei, Timmis, and Alexander, 2017) (see section 4.5.2).

In this section, the total number of the birds in each experiment is 10 (see section 4.5.3) and the number of the prey is 1 (see section 4.1). In the early stage of this case study, a small number (100) of experiments are carried out for N (defined in section 4.1.1) with the range of 2 to 10. The experiments for N is set to 1 are not carried out because 1 robot can never be treated as a swarm. From the experimental results, if the number of N is too high, a total failure occurs too easily. If the number of N is too low, it is too hard to trigger a total failure. Due to the time limitation, large numbers (more than 10,000 for each N) of experiments are carried out only when N is set to 3, 5, and 7 (first quartile, the median, and third quartile). In order to avoid repetition when showing the experimental results and as 5 is the median for N, the results shown in the figures and tables in this section are measured when N is set to 5.

4.6.1 The Selection of Statistical Tests

In statistics, statistical hypothesis testing is the most common way to make statistical decisions when comparing two sets of experimental data (Box, Hunter, and Hunter, 2005). In this case study, two different testing methods, which are the evolutionary testing method and the random testing method, are compared. By applying statistical hypothesis testing, these two testing methods can be compared by comparing the data (results) collected from the experiments of these two testing methods. Hence, statistical hypothesis testing is used for carrying out statistical analysis for this case study.

The formal process for applying statistical hypothesis testing usually contains the following four steps (Marusteri and Bacarea, 2010):

- 1. Stating the null hypothesis and alternative hypothesis. In statistics, the null hypothesis H_0 describes certain statistical behaviour of data and it is treated as true unless the actual behaviour of the data contradicts the hypothesis. The alternative hypothesis H_1 is the hypothesis which is contrasted against the null hypothesis.

- 2. Choosing the significant level. The significant level can be defined as the probability of rejecting the null hypothesis when the null hypothesis is actually true. Popular levels of significance are 5%, 1%, and 0.1%, empirically corresponding to a "confidence level" of 95%, 99%, and 99.9%.

- 3. Obtaining the "P value". The P-value is calculated using the experimental data (or sample data). If it is less than the significance level, the null hypothesis can be rejected.

- 4. Deciding to either reject the null hypothesis in favor of the alternative hypothesis or "fail to reject" it.

The experimental data in this case study has the following properties:

- **Two samples:** One sample is collected for the evolutionary testing method, and the other sample is collected for the random testing method.

- **Independent samples:** The experiments carried out for both testing method are independent of each other.

- **Distribution-free:** The samples are not drawn from a normal distribution.

- **Large sample size:** The size for each sample is at least 100.

- **One-tailed test:** The prediction for the experiments is that the evolutionary testing method has a better performance than the random testing method.

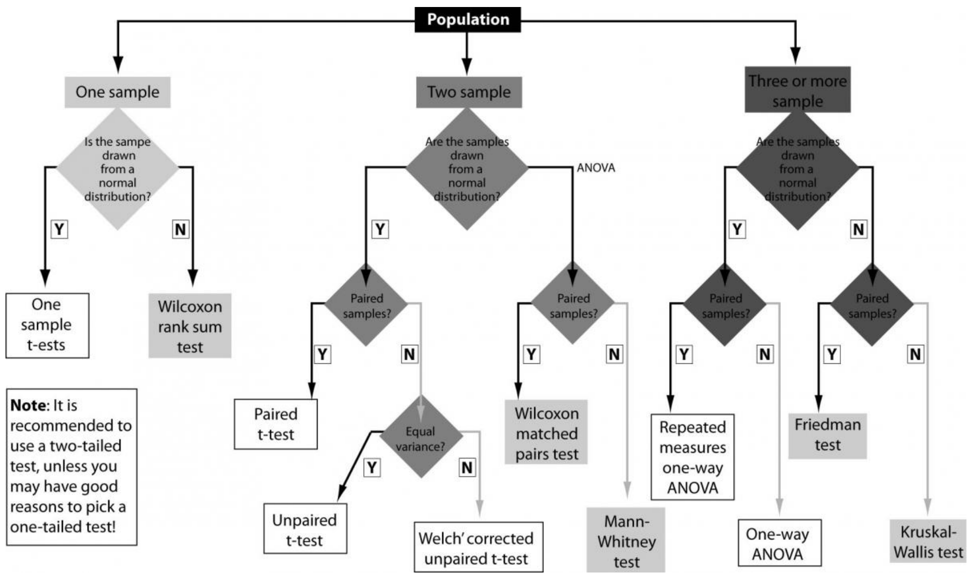

Population
One sample
Two sample
Three or more sample
Is the sampe drawn from a normal distribution?
Are the samples drawn from a normal distribution?
ANOVA
Are the samples drawn from a normal distribution?
Y
N
Y
Y
N
One sample t-ests
Wilcoxon rank sum test
Paired samples?
Paired samples?
Paired samples?
Paired samples?
Y
N
Y
N
Y
N
Y
N
Note: It is recommended to use a two-tailed test, unless you may have good reasons to pick a one-tailed test!
Paired t-test
Equal variance?
Y
N
Wilcoxon matched pairs test
Repeated measures one-way ANOVA
Friedman test
Unpaired t-test
Welch' corrected unpaired t-test
Mann-Whitney test
One-way ANOVA
Kruskal-Wallis test

FIGURE 4.5: The selection process for the right statistical test (Marusteri and Bacarea, 2010).

According to the data properties listed above and the selection process displayed in figure 4.5, one-tailed Mann-Whitney U-test is the best choice for applying statistical analysis to our experimental data. One disadvantage

TABLE 4.2: DETAILS OF THE EXPERIMENTS OF FAILURE SEVERITY

	No. generated	No. compared	No. of evaluation
Evolved	100	100	5000
Random	5000	100	5000

of using statistical hypothesis test is the lack of visualization for the experimental results. Box-and-whisker plot graph is often used in explanatory data analysis (Box, Hunter, and Hunter, 2005), and it is used as a complement to statistical hypothesis test in this thesis.

4.6.2 Comparison of Severity of Failures

In this section, the performance of the swarm control algorithm, which is mentioned in section 4.3, is compared by testing in both the evolutionary testing method and the random testing method. The means of comparison is the fitness value. A high fitness value means that fewer failures occurred during the execution of the swarm, and therefore stands for a good performance of the swarm. The worse the swarm performs, the tougher (better) the test cases are. From the above, the lower the fitness value, the better the test cases are. A null hypothesis is proposed here:

> *H_0: the use of evolved test cases makes no difference to the ability to identify the severity of failures when compared to the random testing strategy, i.e., $f_{evolved} = f_{random}$.*

The alternative hypothesis is :

> *H_1: the use of evolved test cases has a positive effect on the ability to identify the severity of failures when compared to the random testing strategy, i.e., $f_{evolved} < f_{random}$.*

where $\mathrm{f}_{\mathrm{evolved}}$ and $\mathrm{f}_{\mathrm{random}}$ represent the fitness values of the evolved test case and the random test case, respectively. Note than, the lower the fitness value, the better the test cases are.

Table 4.2 shows the details of the experiments carried out for the comparison of severity of failures. When testing the flocking behaviour using the random testing method, 5000 random test cases are generated. The flocking swarm is tested in all 5000 random test cases and the experimental results are recorded. In order to make the competition between the random testing method and the evolutionary testing method equitable, the number of fitness evaluations for both methods should be the same.

TABLE 4.3: DETAILED RESULTS OF MANN-WHITNEY U-TEST

Test Case	avg	std	SoR	MoR	U-value
Evolved	0.215	0.218	5828.5	58.28	9221.5
Random	0.654	0.220	14271.5	142.72	778.5

For random testing method, 5000 evaluations are carried out. For each evolution of which the population size is 20 and the number of generations is 50, 1000 evaluations are carried out. In order to keep the fitness evaluations the same, five independent evolutions are needed, and therefore, 100 evolved test cases are generated. The swarm is then tested in 100 evolved test cases and the experimental results are recorded. When comparing the results, 100 random test cases are randomly chosen.

Table 4.3 shows the detailed results of applying the one-tailed Mann-Whithey U-Test to the experimental results when N is 5. The level of significant chosen for one-tailed Mann-Whithey U-Test is 1%, corresponding to a value of 0.01. The p-value of the test is 0.00001, which is less than the value (0.01) of significant level. In this case, the null hypothesis H_0 is rejected in the favor of the alternative hypothesis H_1 with a confident level of 99%. Hence, the experimental results show that the use of evolved test cases has a positive effect on the ability to identify the severity of failures when comparing to the random testing strategy.

The same statistical hypothesis testing is also applied to the experimental results collected when N is 3 and 7. Both sets of experimental results shows that the use of evolved test cases has a positive effect on the ability to identify the severity of failures when comparing to the random testing strategy.

In this section, the box-and-whisker plot is used for the visual inspection of the experimental results collected. Figure 4.6 shows the distribution of the fitness value of the test cases generated by the evolutionary testing method and the random testing method. From the graph, there are no overlaps in spreads (75% of the experiments in random test cases perform better than 75% of the experiments in evolved test cases), therefore both the mean and median of the fitness value of the evolved test cases are lower than those of the random test cases.

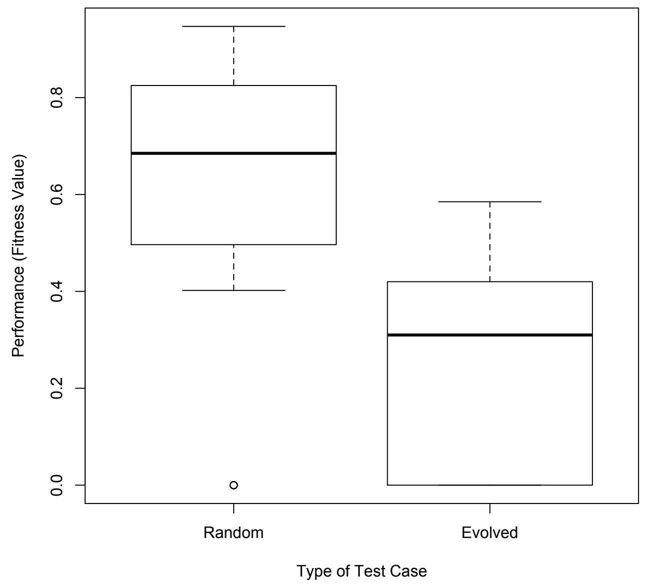


FIGURE 4.6: The fitness value of evolved test cases compared with that of random test cases.

4.6.3 Comparison of Diversity of Failure Type

The purpose of the evolutionary testing method is not only to discover as many instances of undesired behaviours as possible, but also to discover as many distinct types of undesired behaviours as possible. In order to measure the diversity of failures found by the evolutionary testing method, 10 independent evolutions are carried out. For each evolution, there are 20 initial chromosomes in the population and the evolution continues for 80 generations. Figure 4.7 shows that the number of failure types does not keep increasing as the number of generations increases. The number of failure types will reach a peak after a certain numbers (around 30) of generations. Hence, to keep the computing costs to a minimum, the evolutions in this section run for 30 generations.

In this section, 400 evolved test cases are produced. The population size for each evolution is 20, therefore 20 independent evolutions are carried out. To keep the fitness evaluations between the two testing methods the same, 12,000 random test cases are generated. When comparing the results, all types of failures found in all 12,000 random test cases are compared with all types of failures found in the final 400 evolved test cases.

For failure classification, failures are categorized into 20 categories, formed by combining two speed categories with 10 angle categories. The

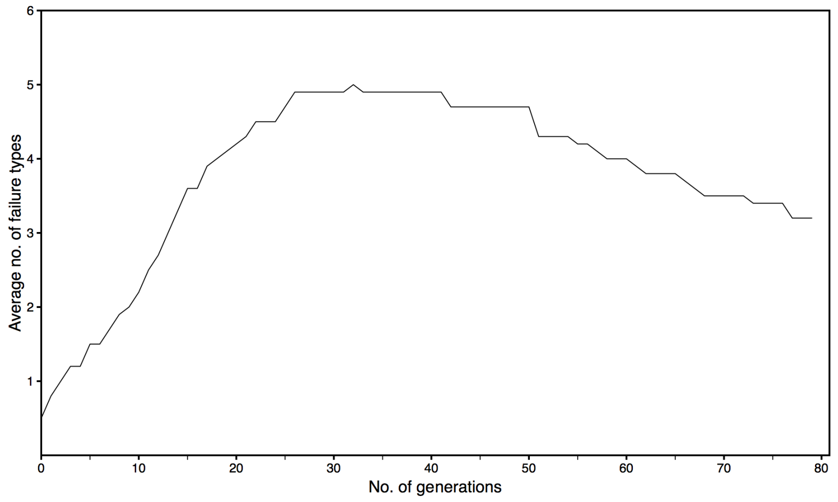


FIGURE 4.7: A line graph shows the average number of failure types of 10 independent evolutions in 80 generations.

TABLE 4.4: DETAILS OF THE EXPERIMENTS OF FAILURE TYPE

	No. generated	No. compared	No. of evaluation
Evolved	400	400	12000
Random	12000	12000	12000

two speed categories are "<50% of average robot speed" and ">=50% of average robot speed". Angles are measured with respect to the mean headings of the swarm members. The angle categories range between 10 and 90 degrees at increments of 10 degrees, along with a tenth "90 degrees plus" category.

Figure 4.8 shows the total failure types found for both evolved and random test cases. In this graph, it is assumed that the executions of 20 different evolutions are independent and parallel. As previously discussed in section 4.4, there are 20 different types of failures in total. Figure 4.8 shows that evolved test cases discover 17 different types of failures, while random test cases discover only 8. Table 4.5 shows the specific failure types discovered by both evolved and random test cases. From the table, it is clear that the set of failure types discovered by random test cases is a subset of the set of failure types discovered by evolved test cases.

Recall that all experiments above in this section were carried out when N is set to 5 for a swarm which contains 10 robots. Experiments were also carried out when N is set to 3 and 7. From the experimental results, even though the number of failure types for both evolved and random test cases changes, the set of failure types discovered by random test cases is always a subset of those discovered by evolved test cases.

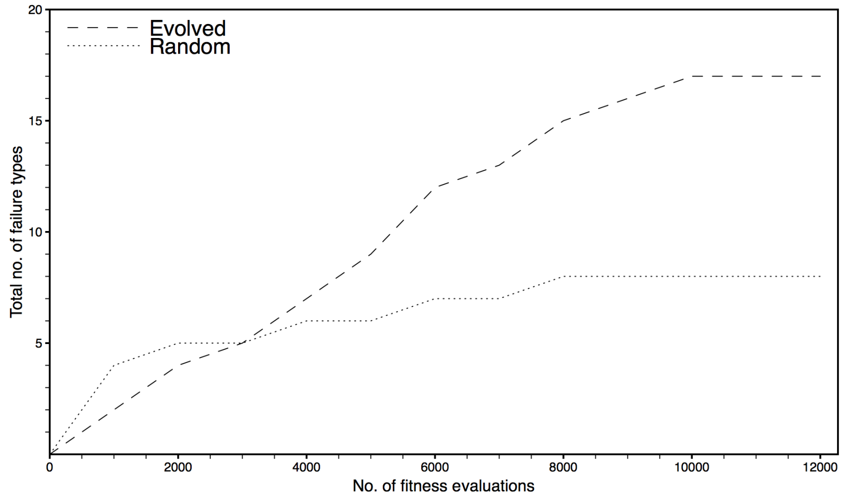


FIGURE 4.8: A line graph shows the total number of failure types discovered during 12000 fitness evaluations.

TABLE 4.5: FAILURE TYPES DISCOVERED.
Grey stands for failure type discovered by the random testing method and Cyan stands for failure type discovered by the evolutionary testing method.

	10°	*20°*	*30°*	*40°*	*50°*	*60°*	*70°*	*80°*	*90°*	*90°+*
Speed≥50%				🟦	🟦	🟦	🟦	🟦	⬛ 🟦	⬛ 🟦
Speed<50%	🟦	🟦	🟦	🟦	⬛ 🟦	⬛ 🟦	⬛ 🟦	⬛ 🟦	⬛ 🟦	⬛ 🟦

Hence, the experimental results from all three sets of experiments show that, after a certain number of fitness evaluations and under the criteria developed for failure classification, the evolved test cases not only cover all the failure types which random test cases identify, but also identify more types of failures.

4.6.4 Discussion

According to experimental results from the two sections above, the evolved test cases lead to worse swarm performance and cover more failure types than the random test cases, suggesting that they are better tests when testing the flocking algorithm mentioned in section 4.3.

4.7 Improving Control Algorithm by Fault Removal

The goal of testing is to find faults and prevent them. In swarm robotic systems, increasing the dependability of the system can help to prevent faults. The dependability of a swarm robotic system is dependent on the quality of its control algorithm. As previously discussed in Chapter 2, there are

TABLE 4.6: NUMBER OF TOTAL FAILURES DISCOVERED BY EVOLVED TEST CASES FOR EACH FAILURE TYPE.

	10°	20°	30°	40°	50°	60°	70°	80°	90°	90°+
Speed≥50%	0	0	0	2	4	5	9	7	9	29
Speed<50%	3	2	10	5	4	15	6	12	15	56

two major methods for designing the control algorithms of swarm robotic systems: the behaviour-based design method and the automatic design method. In this section, the control algorithms which have been designed using the behaviour-based design method is improved. The control algorithm used in section 4.3 was developed using the behaviour-based design method. In this section,fault removal method is used to improve the control algorithm in section 4.3 by tackling the failure types discovered in section 4.6.3.

In the behaviour-based design method, the behaviour of each individual robot is designed iteratively until the desired collective behaviour is obtained. Hence, the quality of the swarm control algorithm depends on the expertise of the developers. Analysing the failure types occurring during the execution of the swarm might give designers some clues about the faults in their control algorithm.

4.7.1 Total Failure Analysis

In study (Moeslinger, Schmickl, and Crailsheim, 2011), there are three discrete reactions for agents in movement: move straight, turn left by a certain angle or turn right by a certain angle. Table 4.6 shows the number of total failures discovered by evolved test cases for each failure type. From the experimental results, there are more total failures occurring when the relative angle is large and the relative speed is less than 50%. This means that the swarms are more robust when the split angle is small, and the agents are more likely to be split from the swarm if their speeds are relatively low.

By reviewing the procedures of the experiments in which total failure occurred, the following is discovered: when the prey robot turns through a certain angle to avoid an obstacle, if the obstacles are within the detectable range of a robot, the turning angle and moving speed of this robot will be affected. This is the major reason for a robot to get split from a swarm. From the above analysis, the weakness of the flocking control algorithm is that the swarm is not robust enough when encountering obstacles.

4.7.2 Algorithm Improvement

A swarm control algorithm might be improved by replacing the weak strategy with a better one. Note that if there are no existing strategies that are better than the current one and the developers are unable to develop a better one, the swarm control algorithm can not be improved.

According to the weakness that the swarms are less robust when the splitting angle is large, the easiest way of overcoming this is to increase the turning speed (turn through a larger angle in the same time period) of the agents when the angle between an agent and its flockmates is large. However, as mentioned in section 2.4.1.2, it is hard to obtain the heading directions of the agents without cheating, therefore the easiest way will not work. In order to improve the control algorithm, the following can be implemented: Adding one more threshold for the agents: If the distance between the agents in front and itself is larger than the new threshold, increase both the turning speed and moving speed of the agent. The value of this new threshold is determined by a trial-and-error strategy in experiments.

4.7.3 Experimental Results of the Improved Flocking Control Algorithm

Experiments that are carried out to measure the performance of the improved flocking control algorithm are described in this section. This improved algorithm will be tested in both test cases evolved based on the original flocking control algorithm and test cases evolved based on the improved algorithm. 100 test cased was evolved based on original flocking control algorithm in section 4.6.2. 100 test cases will be generated based on improved flocking control algorithm. A new null hypothesis is proposed here:

> *H_0: equipping the swarm with the improved flocking control algorithm has no effect on the performance of the swarm, i.e., $f_{improved} = f_{original}$.*

The alternative hypothesis is:

> *H_1: equipping the swarm with the improved flocking control algorithm has a positive effect on the performance of the swarm, i.e., $f_{improved} < f_{original}$.*

where f_{improved} and f_{original} represent the fitness values of the test cases when testing the improved algorithm and when testing the original algorithm, respectively. The lower the fitness value, the better the test cases are.

TABLE 4.7: THREE PAIRS OF RESULT DETAILS FOR THE MANN-WHITNEY U-TEST

Test Case	avg	std	SoR	MoR	U-Value
F/F	0.215	0.218	7834	78.34	7216
I/F	0.398	0.22	12266	122.66	2784
F/F	0.125	0.218	8723	87.23	6327
I/I	0.331	0.216	11377	113.77	3673
F/I	0.193	0.194	8172	8172	6878
I/I	0.331	0.216	11928	119.28	3122

Table 4.7 shows the details of applying the one-tailed Mann-Whitney U-Test to three pairs of experimental results at a significance level of 1%. For all three pairs of experimental results, the p-value of the U-Test is less than 0.00001, which is less than the value (0.01) of significant level. This means the null hypothesis H_0 is rejected in the favor of the alternative hypothesis H_1 with a confident level of 99%. Hence, the performance of the swarm can be improved by replacing the weak mechanism with a better one.

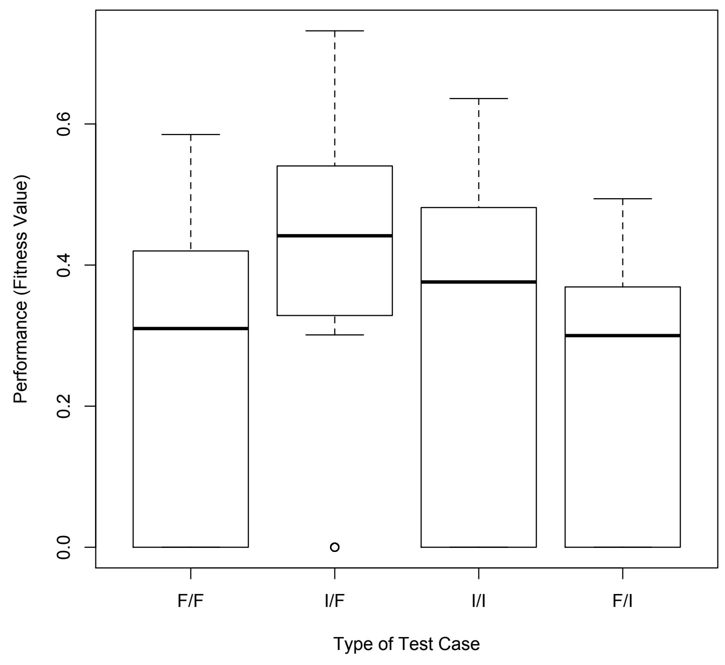


FIGURE 4.9: A line graph showing the total number of failure types discovered during 12000 fitness evaluations.

Figure 4.9 shows the distribution of the fitness values for four sets of experiments. F/F shows the box-plot of testing the original algorithm on test cases evolved based on itself (in section 4.6.2). I/F stands for testing the improved algorithm on test cases evolved based on the original algorithm. Both the original and improved algorithm are also tested on test cases evolved based on the improved algorithm (F/I and I/I in Figure 4.9). Note that there are no fitness values in the range between 0 (exclusive) and 0.3 (exclusive) due to the total failure rule. If the number of the robots in a swarm is less than 5, a total failure occurs and the fitness value is 0.

From the experimental results in Figure 4.9, the fitness values of the improved algorithm are higher than those of the original algorithm in both sets of test cases. The swarm equipped with the improved flocking control algorithm has better performance than the swarm equipped with the original flocking control algorithm.

4.8 Chapter Summary

The work presented in this chapter shows the usefulness of the evolutionary testing method presented in Chapter 3. The experimental results presented in section 4.6 show that the testing method is able to reveal failures in flocking algorithm. The testing method can lead to worse swarm performance and it covers more failure types than the random testing method despite the minimum size of a swarm. The work presented in section 4.7 shows the procedure of fault removal for the flocking control algorithm presented in section 4.3. The experimental results presented in subsection 4.7.3 show that the flocking control algorithm performs better after improvement. Thus the dependability of the swarm robotic system developed for this case study improves. It also shows the evolutionary testing method can help to solve the swarm robotic design problem.

Chapter 5

Testing Method for Foraging Behaviour

One more case study for the testing method proposed in Chapter 3 is carried out in this chapter. The experimental results from previous chapter have shown that the evolutionary testing method is effectively finds failures in the flocking algorithm in section 4.3. The purpose of this chapter is to find out whether the testing method is adaptable to other swarm control algorithms or is only suitable for flocking algorithm. Accordingly, the testing method is modified so that it can be used to test ant foraging behaviour in swarm robotics.

The metrics for measuring performance of ant foraging behaviour and also the foraging algorithm that is tested against are proposed at the beginning of this chapter. These metrics are used with the testing method and then parameters of the testing method are defined. Experiments are carried out and experimental results for both the random testing method and the evolutionary testing method presented in this thesis are compared and analysed to gauge the ability of the evolutionary testing method to identify failures of ants foraging algorithm. At the end of the chapter, an algorithm improvement is applied to the original ant foraging behaviour and the experimental results are discussed.

5.1 Metrics for Ant Foraging Behaviour

Foraging behaviour is a well studied control algorithm in swarm robotics because of its strong biological basis (Liu and Winfield, 2010; Campo and Dorigo, 2007; Hoff et al., 2010) and its potential real-world applications such as instance cleaning, harvesting, search and rescue, land-mine clearance, or planetary exploration (Winfield, 2009). Due to the popularity of foraging behaviour, there are various studies providing metrics for foraging behaviour. Consequently, there is no need to develop a metric in this case study.

One large difference between foraging behaviour and flocking behaviour is that the agents of foraging behaviour in the environment do not have to stay in one cluster (There are some exceptions, for example, in some studies (Werger and Mataric, 1996; Goss and Deneubourg, 1992) the agents are designed to stay close to each other so that a chain can be established between the nest and the target).

The experimental results from (Matarić, 1995) show that the interactions between the agents in a bounded environment have a negative effect on the performance of the group. The study in (Lerman and Galstyan, 2002) proved that the performance of the agents improves as the number of agents grows at first, but declines when the number of agents reaches a certain threshold because of the interference between the agents. Therefore, as long as the number of agents in each test case remains fixed, there is no need to worry about the interactions between the agents.

Due to the the reasons given above, the metric for the performance of a foraging behaviour is quite straightforward and widely used in studies about ant foraging (Hoff et al., 2010; Winfield, 2009; Labella, Dorigo, and Deneubourg, 2006) and is: **the total amount of food** that has been returned to the nest by the entire swarm during a certain amount of time.

This metric is modified in some studies according to the energy consumption (the cost of movement and communication action) of the agents. The main function of the testing method presented in this thesis is to measure the performance of a given control algorithm, and energy consumption is not concerned in this thesis so there is no need to use those modified metrics.

5.1.1 Total Failure in Ant Foraging Algorithm

Section 2.4.2.2 introduced the idea that certain ant foraging algorithms are able to find the shortest path between the nest and the food sources by creating trails in the environment. A concept named total failure for flocking behaviour was introduced in Chapter 4. A total failure in ant foraging behaviour occurs in either of the following two conditions:

- **(1)** the swarm was not able to find any path between the nest and the food resource; (The experiments conducted in this thesis revealed that it takes too long for the swarm system to find the shortest path. In order to reduce computing time, the algorithm is considered to be a success if it found at least one path between the nest and the food sources.)

- **(2)** there are no food items in the nest when the experiment terminates.

Note that condition 2 is severer than condition 1 as condition 2 includes condition 1. In study (Hoff et al., 2010), a path is said to be found if and only if the swarm has established a robot chain between the nest and the food resource. When only condition 1 occurs, there might be some food items which were brought back to the nest by the robots (a robot can wander back to the nest without a robot chain (a path)). If condition 2 occurs, this not only means that no food items were transported to the nest, but also no paths (robot chain) between the food source and the nest were found during the experiment. When condition 2 occurs, there is a large probability that all of the robots in the swarm are trapped.

5.2 Control algorithm for Ant Foraging Behaviour

Apart from the difference mentioned in section 5.1, the constraints for the agents in flocking algorithm, which are local information and only communication without central control, still hold for foraging algorithm. Due to the lack of global information, the agent has no idea about the exact location of the target in the environment when it finds the target. Due to the lack of global communication, the agent is also unable to notify other agents which are beyond its local communication range.

In swarm robotics research, ant foraging behaviour is usually tested in environments without any obstacles. Study (Hoff et al., 2010) is an exception as it tested their systems in obstacle-filled environments. Figure 5.1 shows the test environments which are used in study (Hoff et al., 2010). Study (Hoff et al., 2010) described two foraging algorithms which are virtual pheromone (VP) algorithm and cardinality algorithm. In the case study presented in this chapter, VP algorithm is used to form ant foraging behaviours in the simulation.

5.2.1 Virtual Pheromone Algorithm

Simple local direct communication between robots is used in (Hoff et al., 2010) to transmit a virtual pheromone value. Each robot in VP algorithm can either be a foraging robot or a pheromone robot (beacon as previously

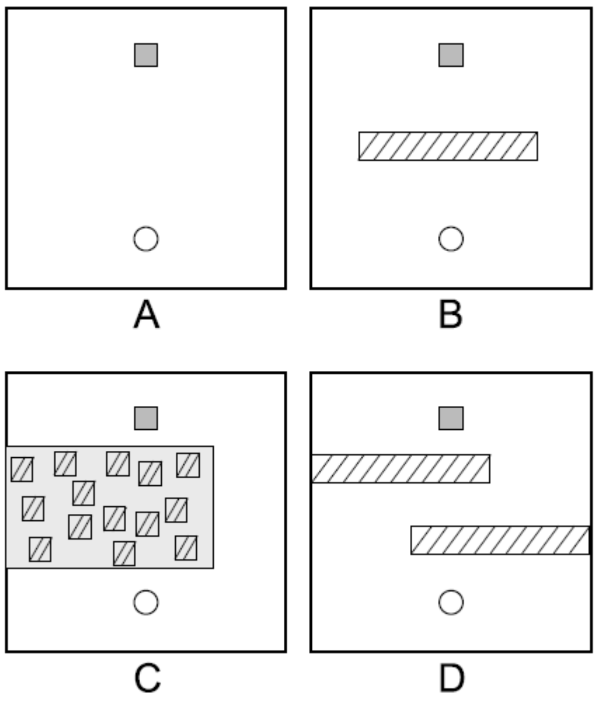


FIGURE 5.1: Four test environments used in study (Hoff et al., 2010). In each test environment, the white circle represents the nest, the grey square represent the food resource, and the rectangles (or squares) with slashes represents obstacles. The large grey rectangle in the middle of (C) means that the grey area is completely occupied by obstacles.

mentioned in 2.4.2.2). A foraging robot follows the basic foraging algorithm (2.4.2.1) to move in the environment. When a robot becomes a beacon, it stops moving and can be used for storing virtual pheromone. The virtual pheromone is a numerical number whose original value is 0. Foraging robot can lay virtual pheromone at a beacon by sending data to the beacon, and can also read the pheromone level by receiving data from the beacon. When there is a network of beacons in the environment, the foraging robots use the distribution of virtual pheromone to decide their next move. Figure 5.2 shows a finite state machine for VP algorithm and the descriptions of the seven states are now given:

- **Wandering:** this state provides the robots with the ability to move around in the environment in order to find the food resource. If there were food resource within the detectable range of the robot, the robot changes its state from wandering to gathering. If there is beacon network nearby, the robot changes its state to Following. If there are fewer than two beacons in its detectable range, it changes its state to wandering.

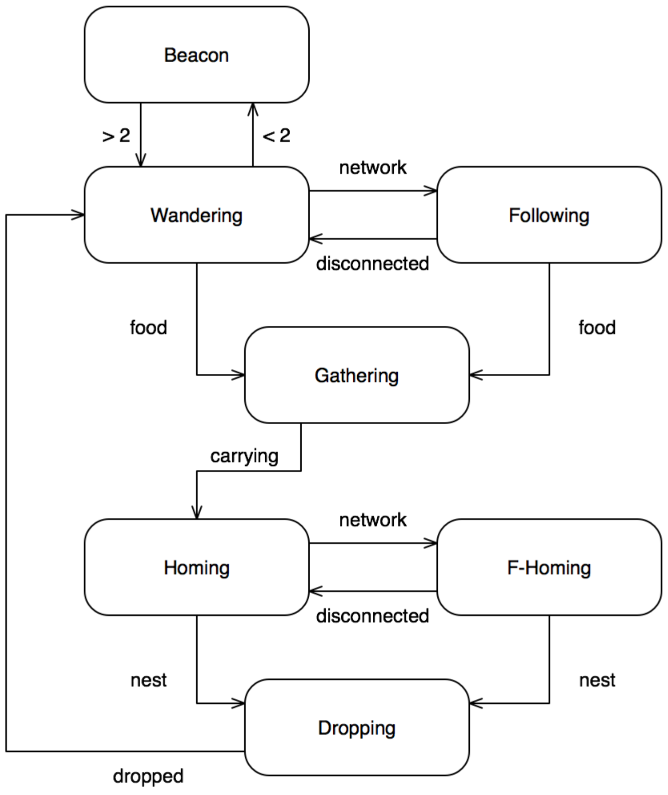


FIGURE 5.2: Finite state machine of virtual pheromone algorithm

- **Following:** the robot follows the beacon network to the food resource. Once the food resource is within detectable range, the robot changes its state to Gathering. If the beacon network is disconnected, the robot changes its state to wandering.
- **Gathering:** the robot gathers a food item from the food resource. Once the robot is carrying the food item, it changes its state to homing.
- **Homing:** the robot moves towards the location of the nest. When the robot reaches the nest, it changes its state to dropping. If there is a beacon network nearby, the robot changes its state to f-homing.
- **F-Homing:** the robot follows the beacon network to the nest. When the robot reaches the nest, it changes its state to dropping. If the beacon network is disconnected, the robots changes its state to homing.
- **Dropping:** the robot drops the food item at the nest and then changes its state to wandering.

- **Beacon:** the robot stops moving and works as a beacon; If there are more than two beacons in its detectable range, it changes its state to wandering with probability P.

The original flocking algorithm was improved in Chapter 4 by adding an obstacle avoidance mechanism to make the algorithm working in the test cases. As the original VP algorithm is able to function correctly in environments with and without obstacles, there is no need to modify it. The VP algorithm in this case study is implemented by using exactly the same parameters used in study (Hoff et al., 2010). The individuals in VP algorithm avoid obstacles as follows: *"When a robot encounters an obstacle, it attempts to avoid it, usually by simply turning left and moving forward"* (Hoff et al., 2010).

5.3 Evolutionary Testing Method

The evolutionary testing method proposed in section 3.2 is a framework of how to test swarm behaviour. Different types of swarm behaviours follow different rules so different sets of parameters need to be defined when testing different swarm behaviours. In this section, the chromosome of GA is defined first, then its fitness function, followed by its parameters of GA. Finally, parameter analysis is carried out at the end of the section.

5.3.1 Chromosome

As in the preceding chapter, a chromosome is the test environment in which the swarm is executed. In ant foraging behaviour, the agents are clustered around their nest at the beginning. When the mission begins, they start to wander around the world to look for food. Once a robot is close enough to a food resource, it will grab a piece of food and start to head towards its nest. After the robot drops the food at the nest, it will continue to look for food in the environment. In this situation, there is one more object compared to the previous chapter in the environment of ant foraging behaviour than that of flocking behaviour, that is, the nest. In order to remain impartial, the linear distance between the food and the nest will always be the same in each test case.

Cellular representation is still used to represent the chromosome in this case study. If there are N obstacles in each test case, a random test case for foraging behaviour can be generated by observing the following the rules listed in section 3.2.1.

TABLE 5.1: PARAMETER VALUES OF GENETIC ALGORITHM

Name of Parameter	Analyzed Candidates	Best Candidates
Number of genes	[4, 6, 8, 10]	8
Population size	[10, 20, 30, 40, 50]	20
Parent selection method	AllParent-BestHalf 2BestParent-2WorseAway 2RandomParents-2WorseAway	AllParent-BestHalf
Crossover type	Single-point-crossover Double-point-crossover Uniform-crossover	Double-point-crossover
Crossover probability	[0.1, 0.2, 0.4]	0.2
Mutation probability	[0, 0.05, 0.1, 0.2]	0.05
Number of generation	[10, 30, 50, 100]	30

5.3.2 Fitness Function

The ultimate goal of the swarm in ant foraging behaviour is to search the environment and return to their nest with food items. Most papers judge the performance of an ant foraging control algorithm by counting the total amount of food that returned to the nest by the entire swarm during a given time. Consequently, the total amount of food returned is used as the fitness value (FV) in this chapter so the fitness function can be represented as follows:

$$FV = TF \tag{5.1}$$

where *TF* is **Total Food** returned to the nest by the entire swarm. In this study, the time limit for the swarm to collect the food is set to 800s (the reason for this is discussed in section 5.4.1).

5.3.3 Parameters Analysis of GA

The parameter robustness technique (Alden et al., 2013) described in subsection 3.2.4 is also used for parameter analysis in this case study. This parameter robustness technique tunes each parameter of the GA individually while keeping all of the other parameters unchanged. The sensitivity of this parameter can be determined by comparing all of the results, which are, the different values for each parameter. If the experimental results change significantly when the value of the parameter changes, this parameter is sensitive in this GA and the value (among the candidate) of this parameter at which the GA performs best can be found. If not, this parameter is insensitive, and therefore altering this parameter will not make a significant difference in the performance of the GA. Table 5.1 shows the parameter values determined according to the analysis using Spartan.

5.4 Experiments

The performance of the evolutionary testing method and the random testing method is measured and analysed in this section. There are two parts in this section. In the first part of this section, the experimental setup, such as the size of the environment, the size of the swarm, and so on, is discussed. The second part of this section compares the experimental results from both testing methods. A test case created by the testing method presented in this thesis is called an evolved test case, and a randomly produced test cases is presented as a random test case. The GA procedure is the same as the procedure listed in section 3.2.5.

5.4.1 Experimental Setup

A few parameters of the testing environment should be decided before the experiments can be carried out in the simulator. Note that the values of parameters of the GA in this case study was chosen according to the experimental setup decided in this section. For different experimental setups, different sets of GA parameters need to be chosen by using the parameter robustness technique proposed in (Alden et al., 2013). The experimental setup is showing below:

- **Size of the Environment**: The testing environment is a 2-dimensional rectangular space with or without obstacles. The length and width of the testing environment can be arbitrarily large. As a matter of convenience, a square space with sides 6 meters long was selected as the testing environment. This does not mean that a 6m$\times$6m square is the best candidate for the testing environment. Figure 5.3 shows an example of a test case for ant foraging behaviour.

- **Size of the Swarm**: One property of swarm robotic system is that the swarm should contain a large number of robots. However, there are no limits for the minimum number of robots in a swarm. Due to the scalability of the swarm robotic system, the number of robots in a swarm should not be a problem to worry about. Twenty is used as the size of a swarm in this case study.

- **Position of the Nest and Food**: In order to be objective, the distance between the nest and the food should always be the same in all testing environments. For convenience, the nest will be placed at the bottom of the environment while the food will be placed at the top. There is only one food resource in the environment, which never runs out of food.

- **Execution time**: The running time for each experiment is 800 seconds. In the simulator, the speed for each robot is 5 cm/s and the distance between the nest and the food resource is 4 meters. The running time of 800 seconds allows a single robot to travel between the nest and food resource 10 times. The experiments terminate when the time has expired.

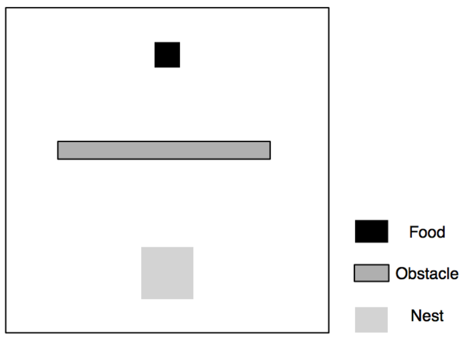


FIGURE 5.3: An example of a test case for ant foraging behaviour. This is a bounded 6m$\times$6m square environment with one obstacle.

5.4.2 Experimental Results of VP Algorithm

Three kinds of test cases are used in this case study. They are empty test case (no obstacles), random test case, and evolved test case. Firstly, the performance of the swarm in no-obstacle test cases is measured in order to set up a baseline for the rest of the experiments. The performance of the simulated swarm, which is equipped with the VP algorithm mentioned in section 5.2, in all three types of test cases is compared during later experiments. The means of comparison is the fitness score which is equal to the amount of food returned to the nest by the swarm. Therefore, the lower the fitness score, the tougher the test case (the worse the swarm performed). As the performance of the swarm in no-obstacle test cases is used as a baseline, only evolved and random test cases are compared. The experimental data collected in this case study shares the same properties listed in section 4.6.1. Hence, one-tailed Mann-Whitney U-test is used for applying statistical analysis in this case study. A null hypothesis is proposed as follows:

> *H_0: the use of evolved test cases makes no difference to the ability to search-and-transport the food to the nest in the ant foraging algorithm compared to the random testing strategy, i.e., $f_{evolved} = f_{random}$.*

TABLE 5.2: DETAILS OF THE EXPERIMENTS OF VP ALGORITHM

	No. generated	No. compared	No. of evaluation
Evolved	100	100	3000
Random	3000	100	3000

TABLE 5.3: DETAILED RESULTS OF MANN-WHITNEY U-TEST OF VP ALGORITHM

Test Case	avg	std	SoR	MoR	U-value
Evolved	1.13	1.01	5255	52.55	9795
Random	4.96	1.37	14845	148.45	205

The alternative hypothesis is:

> *H_1: the use of evolved test cases has a positive effect on the ability to search-and-transport the food to the nest in the ant foraging algorithm compared to the random testing strategy. i.e., $f_{evolved} < f_{random}$.*

where f_{evolved} and f_{random} represent the fitness values of the evolved test case and the random test case, respectively. The lower the fitness value, the better the test cases are.

The performance of the swarm in empty test cases needs to be measured in order to set a baseline for later experiments. The swarm is executed in empty test cases 100 times. The layout of each empty test case is identical. The only difference between each experiment is the starting positions of the robots in the swarm. The distribution of the fitness score for empty test cases is shown in Figure 5.4.

Table 5.2 shows the details of the experiments carried out for testing VP algorithm. When testing the VP algorithm using the random testing method, 3000 random test cases are produced and the swarm is executed in all 3000 of these random test cases with all of the experimental results are being recorded. Consequently, 3000 evaluation are carried out. For each evolution of which the population size is 20 and the number of generations is 30, 600 evaluations are carried out. In order to keep the fitness evaluations between random testing method and the evolutionary testing method the same (at 3000), five independent evolutions are needed. As a result, 100 evolved test cases are generated. The swarm is then executed in all 100 evolved test cases, and the performance in each test case is measured and recorded. In order to compare the results, 100 random test cases need to be chosen at random.

Table 5.3 shows the detailed results of applying one-tailed Mann-Whitney U-Test to the experimental results at a significance level of 1%. The p-value of the test is 0.00001, which is less than the value (0.01) of the significant level. In this situation, the null hypothesis H_0 is rejected in the favor of the alternative hypothesis H_1 with a confident level of 99%. Hence, the use of evolved test cases has a positive effect on the ability to search-and-transport the food to the nest in the ant foraging algorithm compared the random test cases.

Figure 5.4 presents a box-and-whisker plot which shows the distribution of the fitness values of the test cases generated by the evolutionary testing method and the random testing method. From the graph, both mean and median of the fitness value of random test cases is higher than those of evolved test cases. Because the lower the fitness value of a test case is, the tougher the test case, the evolved test cases made the swarm harder to search-and-return the food to the nest.

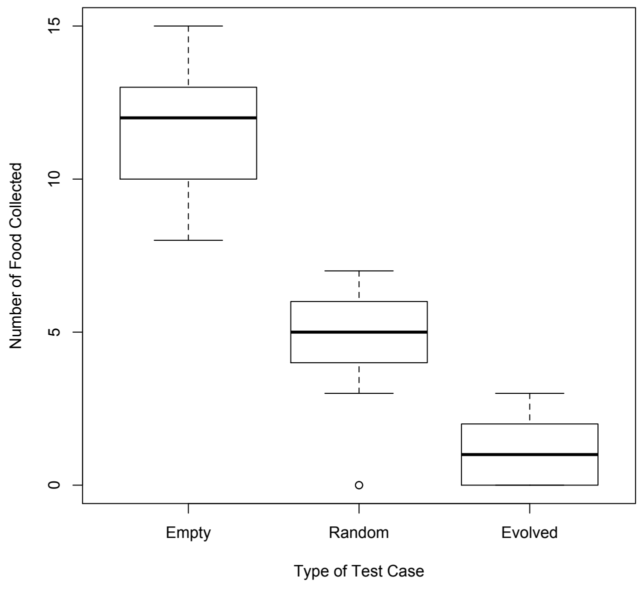


FIGURE 5.4: The fitness value of evolved test cases compared to the random test cases.

5.4.3 Discussion

According to the experimental results shown above, the swarm system applied using the VP algorithm transported much more food items to the nest in the random test cases than in the evolved test cases. This means that it is harder for the swarm to search-and-transport the food to the nest in evolved test cases than in random test cases. Therefore, evolved test

TABLE 5.4: TOTAL FAILURES OF VP ALGORITHM FOUND DURING THE EXPERIMENTS

Total Failure	Random Testing	Evolved Testing
Condition 1	24	91
Condition 2	2	35

cases are better tests when testing ant's foraging algorithm mentioned in section 5.2.

5.5 Improving Control Algorithm by Fault Removal

Total failures in ant foraging behaviour were introduced in section 5.1.1. Table 5.4 shows the total failures found during the experiments in section 5.4.2. The total number of experiments for each testing method is 100. The VP algorithm is able to establish at least one path between the food resource and the nest in 76% of the random test cases and 9% of the evolved test cases.

5.5.1 Total Failure Analysis

It was mentioned in Chapter 3 that involving a testing process in system development helps the developers to find the faults in their coordination algorithm. The developers have to review the experiments in which total failure occurs in order to identify the faults. The rest of this section shows a representative example of identifying faults in the VP algorithm.

Figure 5.5 shows the trajectory of a robot which is trapped by certain obstacles after gathering one food item from a food source. In robotic exploration task, a situation where a robot is blocked by obstacles or another robots and can not achieve its task is called a *deadlock* (Jager and Nebel, 2001). In order not to get confused by the trajectories of all of the robots, only the trapped robot is shown in the testing environment. The black arrow on the robot shows the direction of the robot is facing. The robot reaches point (a) first by following virtual pheromone. It detects the food resource and then moves towards the food resource. After grabbing a food item, the robot starts to move back to the beacon so that it can follow virtual pheromone to return to the nest. However, there is an obstacle between the beacon and the robot. The robot avoids the obstacle by turning left and the beacon is now beyond the detectable range of the robot. As the robot is carrying a food item, it can not become a beacon and has to move on its own until it finds a beacon or the nest. After the robot reaches point (b), it always follows the trajectory which is the loop showing at the top of the figure. In this evolved test case, the maximum number of robots trapped by this trap at the same time is 6. The only way for a trapped robot to escape from the deadlock situation is to be obstructed by another robot so that the moving direction in which it is moving can be changed.

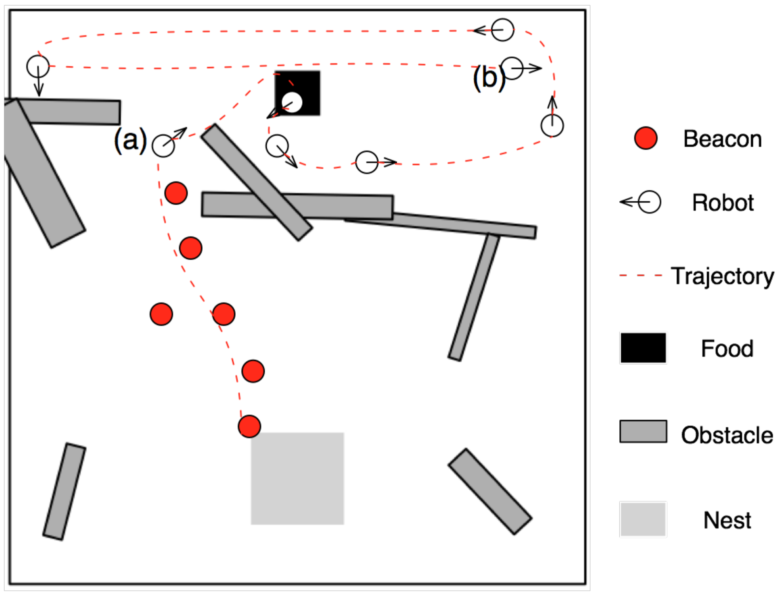


FIGURE 5.5: The trajectory of a trapped robot in an evolved test case for VP algorithm.

It is obvious that only three obstacles are involved to trigger a deadlock situation (the boundaries of this test case are also involved). However, study (Hoff et al., 2010) states that *"the VP algorithm functions correctly in worlds with and without obstacles"*. By reviewing the VP algorithm and the deadlock, one fault of the VP algorithm was found to be its obstacle avoidance strategy. When an individual robot meets an obstacle, it always turns left to avoid collision with this obstacle. This obstacle avoidance strategy worked correctly in the test cases listed in Figure 5.1, but it breaks the swarm system in certain situations.

5.5.2 Algorithm Improvement

A coordination algorithm might be improved by replacing the weak mechanism/strategy which causes the failure. Either an existing strategy which performs better for the coordination algorithm can be adapted or a new one developed. In order to find out whether the algorithm has been improved, the performances of the new algorithm and the original algorithm are compared. If there were no better strategies (a better one could not be developed) to remove the fault, this means that this coordination algorithm can not be improved for the time being.

In the previous section, it was found that the weak strategy in VP algorithm is obstacle avoidance. To improve the VP algorithm, a better obstacle avoidance strategy needs to be deployed. At the beginning, the obstacle avoidance strategy was changed to: when an individual robot meets an obstacle, it randomly turns right or left to avoid collision with this obstacle. This strategy helps the robot to escape from a trap in Figure 5.5. However, after testing the new algorithm using the evolutionary testing method (generate new evolved test cases based on the new algorithm), there is not much difference between the performance of the new algorithm and that of the old one. In this case, it cannot be said that the VP algorithm has been improved.

5.5.2.1 Reactive Behaviour and Non-Reactive Behaviour

In robotics research, there are two types of behaviours, which are reactive behaviour and non-reactive behaviour. Reactive behaviours are behaviours of robot which acts purely on what the sensors detect. Non-reactive behaviour are behaviour of robot which acts not only on what the sensors detect, but also depends on its past experiences or its internal states. For example, the internal states of a self-driving car contains predefined representations, such as traffic lights, traffic signs, markings on the road, and so on, of the world. Note that, even most of current evolutionary robots and reinforcement learning robots can develop their controllers based on past experiences, the behaviours of these robots are still reactive because they do not have internal memories to evolve the controller or learn new behaviours after the deployment.

The behaviour of all the robots used in this thesis are reactive. Due to the lack of concepts of the past, reactive robot acts on what it detects. For example, in Figure 5.5, the robot is trapped in a deadlock situation because its behaviour is depending on the obstacles in the environment. In this situation, non-reactive robots will be able to escape from the deadlock by learning from the past, however, developing non-reactive robots is beyond the scope of this thesis. Hence, the following section improves the coordination algorithm in the field of reactive robotic systems.

5.5.2.2 Path Planning - Potential Field Method

The local path planning method enables a robot to reach a goal in an unknown environment while avoiding the obstacles. Potential field method (Khatib, 1986) is a well-known local path planning method with obstacle avoidance. Figure 5.6 is a demonstration of how potential field method works. It assumes that the robot acts as a particle moving under the influence of two virtual forces produced by the goal and the obstacles. The virtual force produced by the goal attracts the robot towards the goal. The other virtual force produced by the obstacles repels the robot away from the obstacles. The moving direction of the robot is determined by the resultant of the attractive virtual force and the repulsive virtual force.

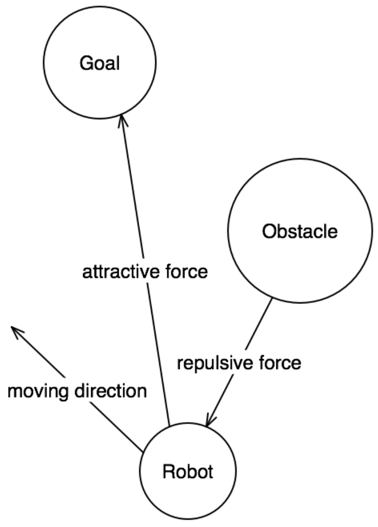


FIGURE 5.6: A demonstration of how potential field method works.

In order to improve the performance of the VP algorithm, its original obstacle avoidance was replaced by the potential field method. The new obstacle avoidance works according to the following rules:

- If any beacons are within detectable range, the robot avoids the obstacle using potential field method by treating the beacon as the goal.
- If beacons are out of range, the robot turns right or left randomly to avoid the obstacle.

5.5.3 Experimental Results using the Improved VP Algorithm

To measure the performance of the IVP (Improved VP) algorithm, the swarm equipped with IVP is tested in both the evolved test cases generated for VP and the evolved test cases generated for IVP. 100 test cases are generated based on IVP. In fact, 100 test cases were generated based on VP in section 5.4.2. VP will also be tested in the evolved test cases generated for IVP. In this section, a new null hypothesis is proposed:

TABLE 5.5: THREE PAIRS OF DETAILED RESULTS OF MANN-WHITNEY U-TEST

Test Case	avg	std	SoR	MoR	U-Value
VP/VP	1.13	1.01	5350	53.5	9700
I/VP	4.16	1.28	14750	147.5	300
VP/VP	1.13	1.01	7585.5	75.86	7464.5
I/I	2.16	1.04	12514.5	125.14	2535.5
VP/I	0.67	0.77	6472.5	64.72	8577.5
I/I	2.16	1.04	13627.5	136.28	1422.5

> H_0: *equipping the swarm with IVP has no effect on the ability of searching-and-transporting the food to the nest for the swarm, i.e.,* $f_{improved} = f_{original}$

The alternative hypothesis is:

> H_1: *equipping the swarm with IVP has a positive effect on the ability of searching-and-transporting the food to the nest for the swarm, i.e.,* $f_{improved} < f_{original}$.

where f_{improved} and f_{original} represent the fitness values of the test cases when testing the improved algorithm and when testing the original algorithm, respectively. The lower the fitness value, the better the test cases are.

Table 5.5 shows three pairs of detailed results of applying one-tailed Mann-Whitney U-Test to the experimental results at a significant level of 1%. For all three pairs of experimental results, the p-value is less than 0.00001, which is less the the value (0.01) of significant level. In such case, the null hypothesis H_0 is rejected in the favor of the alternative hypothesis H_1 with a confident level of 99%. Hence, equipping the swarm with IVP has a positive effect on the ability of searching-and-transporting the food to the nest for the swarm.

Figure 5.7 shows the distribution of the amount of food collected for four groups of experiments. VP/VP stands for testing VP on test cases evolved on the basis of VP. The experiments have already been carried out in subsection 5.4.2. VP/VP will be used as a baseline to find out whether or not there is an improvement in the performance of the swarm after applying IVP.

During the experiments, IVP was first tested on test cases evolved on the basis of VP (I/VP in figure 5.7). Both IVP and VP were then tested on test cases evolved on the basis of IVP (I/I and VP/I in figure 5.7).
Figure 5.7 shows that I/VP performs best of all of four sets of experiments. Even though I/I does not perform as well as I/VP, it still outperforms

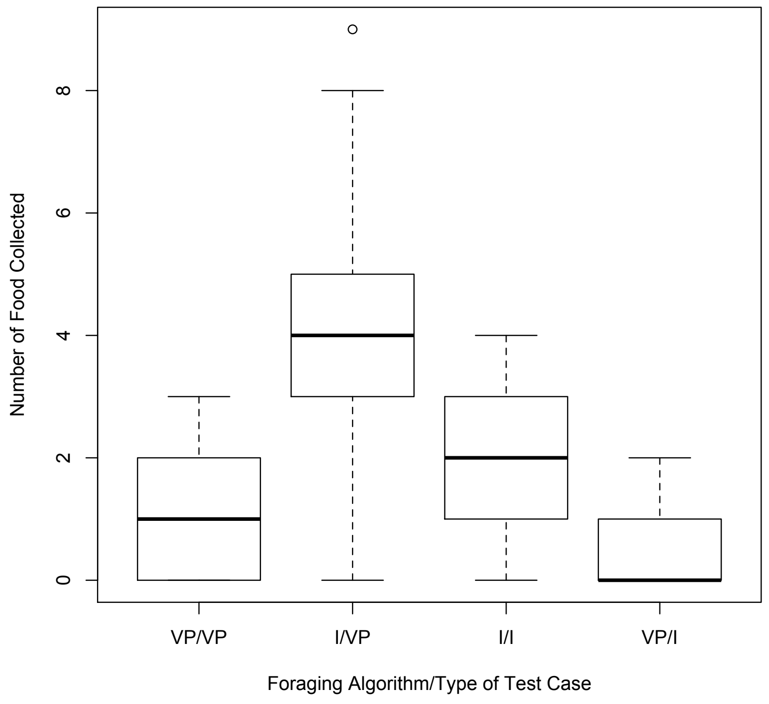


FIGURE 5.7: The distribution of number of collected food for testing VP on test cases evolved based on VP, testing IVP on test cases evolved based on VP, testing IVP on test cases evolved based on IVP, and testing VP on test cases evolved based on IVP

VP/VP and VP/I. The swarm equipped with IVP is better at search-and-transport the food to the nest than the swarm equipped with VP.

5.6 Chapter Summary

The work presented in this chapter shows that the testing method proposed in Chapter 3 is able to reveal failures in more than one type of swarm behaviour. The experimental results presented in section 5.4.2 show that it is more difficult for the swarm to search-and-transport the food to the nest in evolved test cases than it is in random test cases. The work presented in section 5.5 shows that the performance of the ant foraging behaviour presented in study (Hoff et al., 2010) can be improved by removing the fault identified by the evolutionary testing method.

Chapter 6

Testing Method for Task Partitioning Behaviour

Another case study for the testing method proposed in Chapter 3 is presented in this chapter. The control algorithm of task partition behaviour described in 2.4.3 is tested. This chapter starts with the selection of metrics and GA parameters for task partition behaviour. Following this, the task partitioning algorithm is presented. Experiments are carried out and experimental results for both the random testing method and the evolutionary testing method are compared and analysed to determine the ability of the evolutionary testing method to reveal failures in the task partitioning algorithm. Finally, a algorithm improvement is applied to task partition behaviour and the experimental results are discussed.

6.1 Metrics for Task Partitioning Behaviour

So far in this study, the work presented in previous chapters has shown not only that the evolutionary testing method produces better quality of tests than random tests do for certain swarm behaviours, but also that this method can be used for testing different swarm behaviours. As previously mentioned, reusability is one of the defining characteristics of successful test automation. Once the parameters are established, most of the procedures of the evolutionary testing method are automated, which saves a great deal of time and reduces computing costs compared to manual tests. However, the parameter analysis still requires a great deal of human effort. Although it is not possible to reduce human effort on parameter analysis for all swarm behaviours, there is a chance that those effort can be reduced for similar swarm behaviours.

There are many examples of task partitioning observed in the actions of social insects, such as foraging task partitioning in ants. In ants' foraging, they normally search and then transport the food resource to their nest directly. Task partitioning can happen here as one individual collects one part of the food resource and then passes it to another individual which can transport the food resource back to the nest. Under such circumstances, task partitioning can be treated as an extension of foraging behaviour. In many studies (Ratnieks and Anderson, 1999; Pini et al., 2011b), task partitioning is treated as an improved version of ants' foraging behaviour as it is able to reduce the interference between the individuals and increase the efficiency of the swarm. In this situation, task partition and foraging behaviour are considered to be similar swarm behaviours. This chapter uses task partitioning as the studied swarm behaviour to illustrate the procedure of testing a swarm behaviour by using parameters which are chosen based on a similar behaviour (foraging).

As previously detailed in section 5.1, one reason for choosing the total amount food as the only metric for ants foraging behaviour is that there are no interactions among the foraging robots except avoiding each other. Note that there are communications between the foraging robots and beacon robots when leaving pheromones. This is not the case for task partitioning behaviour as the robots need to interact with other robots and transfer food items from one to another in order to complete the task. If this interaction has an effect on the performance of the swarm, another complementary metric might be required in order to measure the performance of task partition behaviour accurately. Note that, when developing a testing method for a similar swarm behaviour, metrics should always be reviewed to check whether they are suitable for the new swarm behaviour. If not, they need to be modified or replaced with a set of new metrics according to the requirements of the new swarm behaviour.

When food items are required to be transferred among different subtasks, there are usually two kinds of transportation: indirect transfer using temporary storage, or direct transfer between the robots. If the items are transferred indirectly, certain locations in the environment are used as temporary storage. The robots drop the items in the temporary storage area so that the following robot can pick the item up. In this situation, there are actually no interactions between the robots. This is the same as the situation in ants foraging behaviour: the robots only interact with the food items. In this situation, no more metrics for indirect transportation are required.

In direct transportation, food items are transferred directly from one robot to another. In current task partitioning research, the major benefit of direct transportation is the increase in the overall rate of items delivered to the nest. This is because the usage of temporary storage is relatively inefficient in indirect transportation, it is time-consuming when the temporary storage is empty and the robots have to wait. The only interaction between the robots is to transfer the item and the robots do not have to achieve some tasks cooperatively. Under such circumstances, even though there exist interactions in direct transportation, there is no need for another metric as the benefit of using transportation is reflected in the number of food items delivered to the nest.

Hence, for current task partitioning research, the metrics used for measuring the performance will be the same as those for ants foraging behaviour: **the total amount of food** that has been transported to the nest by the entire swarm during a certain length of time. In future studies, if the robots have to cooperate to achieve certain tasks, such as transporting a large object which is beyond the capability of each individual robot, in task partitioning, complementary metrics will be needed.

6.1.1 Total Failure in Task Partitioning Behaviour

As the swarm has the ability of finding the shortest path between the nest and the food resource, two types of total failures are defined in VP algorithm in section 5.1.1. For the static partitioning strategy, the swarm has no other abilities than search-and-transport the food to the nest. Hence, there is only type of total failure, which is defined as follows: if there were no food items in the nest at the end of an experiment, a total failure occurs for this experiment.

6.2 Control Algorithm for Task Partitioning

In this section, the task partitioning algorithm which is applied to foot-bots in order to achieve task partitioning behaviour in ARGoS is discussed. According to the reviews in section 2.4.3.1, there are only two candidates here: static partitioning strategy, and dynamic partitioning strategy.

In section 2.4.3.1, a task partitioning algorithm named bucket-brigade is reviewed. However, bucket-brigade is designed for foraging in a multi-robotic system. The territories of the robots need to be predefined, and environmental information is required for the system to function properly. Accordingly, it is not suitable for developing swarm robotic systems.

The advantage of the dynamic partitioning strategy over the static partitioning strategy is that the information about the distance between the nest and food resource is not required before the experiments. However, the experimental results in (Buchanan, Pomfret, and Timmis, 2016) show that the dynamic partitioning strategy takes at least 5 hours in ARGoS to converge to the travel distance needed for the robots, while the travel distance is predefined in the static partitioning strategy. At the current stage of the testing method presented in this thesis, the size of test cases and the distance between the nest and food resource are always fixed. Consequently, using the static partitioning strategy saves lots of computing time. Moreover, once the travel distance has been decided in the dynamic partitioning strategy, the throughput and overall behaviour are the same as those under the static partitioning strategy.

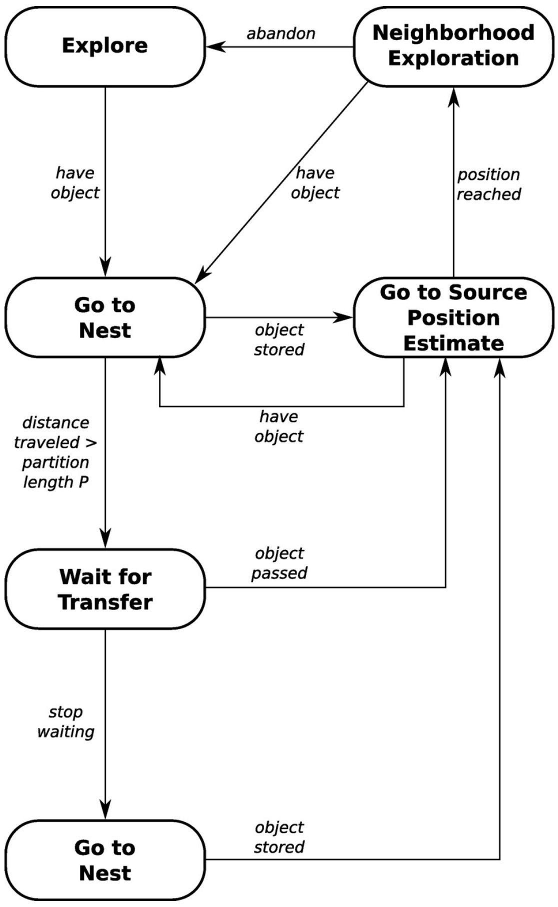


FIGURE 6.1: Finite state machine diagram for static partitioning strategy(Pini et al., 2014)

The static partitioning strategy uses a finite state machine to control the behaviour of each robot. Figure 6.1 shows the finite state machine which represents the behaviour of the robots in the static partitioning strategy. When the robots are first deployed in the environment, they have no idea about the location of the food source. The robots explore the environment to find the source and grab it (the robot is now in **Go to Nest** state). Then the robot follows the following states to form a task partitioning behaviour (Pini et al., 2014):

- **Go to Nest:** the robot enters this state once it is carrying a food item (having either found the food item itself or obtained it from another robot). It will mark this position as the *estimated source position*. Then the robot heads towards the nest while keeping track of its travel distance. After the robot reaches the nest and stores the food item, it enters **Go to Source Position Estimate** state. If the travel distance is equal to *partition length* (a threshold which is used to determine the longest distance that a robot can travel in its current states) before the robot can reach the nest, the robot enters **Wait for Transfer** state.

- **Wait for Transfer:** in this state, the robot stays still and waits for another robot so that it can transfer the food item to the next robot. If the waiting time reaches a threshold, the robot stops waiting and enters **Go to Nest** state. If another robot takes over the food item, it enters **Go to Source Position Estimate**.

- **Go to Source Position Estimate:** in this state, the robot moves towards the *estimated source position*, which is recorded in its memory. If it finds the food resource, it enters **Go to Nest** state. If it can not find the food resource at the *estimated source position*, it enters **Neighbourhood Exploration** state.

- **Neighbourhood Exploration:** in this state, the robot searches around the *estimated source position* for a food resource. If the robot finds a food item, it enters **Go to Nest** state. If the robot can not find any food resource within a certain time, it enters **Explore** state.

- **Explore:** in this state, the robot searches the whole environment with the goal of finding a food resource. Once a food resource is located, it enters **Go to Nest** state.

6.2.1 Path Planning Method

In study (Pini et al., 2014), light sources are added at the location of the nest. The robots use their light sensors to detect the direction of the light sources. The light sensors are long range sensors so that the light sources are always in the detectable range of the robots. When a robot is carrying a food item, it moves towards the light sources in order to reach the nest. Recall that no light sources are required for path planning in the VP algorithm in section 5.2. The robots in VP can turn themselves into beacons, which have the same effect as light sources. The sensors of the robot in VP are short range, the robot has to be around a beacon in order to detect it.

The path planning method for the static partitioning method is described as follows(Pini et al., 2014): the robots move towards the nest by following the direction of the light sources. When a robot performs obstacle avoidance, it uses infrared proximity sensors as bumpers to move around the obstacle or the other robot. Even though the name of the path planning method used in (Pini et al., 2014) was not mentioned, it follows similar coordination rules as the potential field method (Khatib, 1986), which was applied to VP in section 5.5.2.2. In static partitioning strategy, the direction of the light sources is the direction of the attractive force in the potential field method. The obstacle is in the opposite direction to the repulsive force. The actual moving direction of the robot is the resultant of the attractive and repulsive forces.

In section 5.5.3, the experimental results show that the potential field method improves the robots' ability at obstacle avoidance under the VP algorithm. In the static partitioning strategy, the potential field method for obstacle avoidance will be used.

6.3 Evolutionary Testing Method

The parameters for testing task partitioning behaviour will be defined according to the framework proposed in section 3.2. In this section, the chromosome, fitness function, and parameters of GA will be defined. This section can also be used as a guide for how to develop a testing method when there exists a testing method for similar swarm behaviour.

6.3.1 Chromosome

When developing a chromosome according to another chromosome of similar swarm behaviour, the differences between the running environments of those two swarm behaviours are needed to be analysed. The difference or lack of differences will be determined by comparing each element of the environments. Figure 6.2 shows the two testing environments without any robots or obstacles. Figure 6.2 (a) is the testing environment for ant foraging and figure 6.2 (b) shows the testing environment for task partitioning. From the figure, it is clear that there is one more element, which is the light source, for task partitioning. When generating a random test case for task partitioning, the rules listed in section 5.3.1 can be followed and one more rule can be added to the end: place a light source at the centre of the nest.

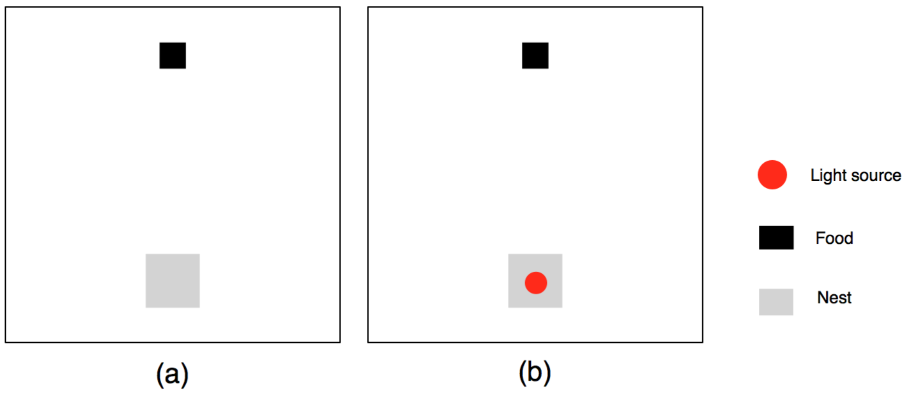


FIGURE 6.2: A comparison of testing environments between ant foraging and task partitioning.

6.3.2 Fitness Function and Parameter Analysis

Fitness function for a swarm behaviour uses the metrics defined for this behaviour to measure the performance of the swarm in a given test case. The differences between the metrics of two swarm behaviours are needed to be compared in order to build the fitness function. By comparing the metrics defined in section 5.1 and section 6.1, it is clear that the metric of task partitioning is the same as that of ant foraging. Hence, the fitness function for task partitioning will be the same as equation 5.1 listed in section 5.3.2.

6.3.3 Parameter Analysis of Genetic Algorithm

As previously mentioned at the beginning of this chapter, there is a chance to reduce the human effort expended on parameter analysis when developing a testing method based on the testing method of a similar swarm behaviour. In this case study, the parameter values selected for ants foraging behaviour (listed in Table 5.1) are used as as the parameter values for the new testing method. If the new testing method were able to reveal failures in task partitioning, the human efforts on parameter analysis can be reduced for similar swarm behaviour. If not, parameter analysis has to be carried out for each new testing method developed.

6.4 Experimental Results

This section presents the experiments and results of testing task partitioning behaviour. There are two independent parts: the first part shows the procedure for carrying out experiments using the parameters established in the previous chapter (section 5.4). The experimental setup will be the same as the setup used in section 5.4 except that the size of the swarm is 6. The reason for the change of the swarm size is that only 6 robots are used in study (Pini et al., 2014). All the experiments in this part are carried out in the steady state of static task partitioning, in which the rate of transporting food items to the nest has converged to a stable rate. When the experiments begin, three robots are deployed at the centre of the environment while the other three are located near the food resources. This approach saves time in converging to the steady state when carrying out experiments. This section compares and analyses the experimental results of empty, random, and evolved test cases. The second part of this section shows the effectiveness of both evolved and random test cases on the convergence times of task partitioning.

6.4.1 Experimental Results for the Static Partitioning Strategy

In this section, the parameter values of the GA and the experimental setup will be the same as those used in section 5.4. At the beginning of this case study, the performance of the swarm in test cases with no obstacles is tested. The experimental results of empty test cases will be compared with those of later experiments to demonstrate the effect of the obstacles. Then the swarm will be tested in both evolved and random test cases, and the experimental results from all three kinds of test cases will be compared. The means of comparison is still the fitness value, which is the number of food items transported to the nest by the swarm, of the test cases. Hence, the lower the fitness value, the tougher the test case (the worse the swarm performed). Statistical hypothesis testing is applied in this case study according to the analysis in section 4.6.1. The following null hypothesis is postulated:

> *H_0: the use of evolved test cases makes no difference to the ability to search-and-transport food to the nest of the task partitioning algorithm when compared to the random testing strategy, i.e., $f_{evolved} = f_{random}$.*

The alternative hypothesis is:

> *H_1: the use of evolved test cases has a positive effect on the ability to search-and-transport food to the nest of the task partitioning algorithm when compared to the random testing strategy, i.e., $f_{evolved} < f_{random}$.*

TABLE 6.1: DETAILS OF THE EXPERIMENTS OF TASK PARTITIONING

	No. generated	No. compared	No. of evaluation
Evolved	100	100	3000
Random	3000	100	3000

TABLE 6.2: RESULT DETAILS OF THE MANN-WHITNEY U-TEST

Test Case	avg	std	SoR	MoR	U-value
Evolved	0.21	0.48	5237.5	52.38	9812.5
Random	4.29	1.58	14862.5	148.62	187.5

where $f_{evolved}$ and f_{random} represent the fitness values of the evolved test case and the random test case, respectively. The lower the fitness value, the better the test cases are.

Firstly, the swarm is tested in empty test cases 100 times. The layout of the food resource and the nest for each experiment is identical, but the starting positions of the robots might vary between each experiment. The distribution of the fitness value for empty test cases is shown in Figure 6.3.

Table 6.1 shows the details of the experiments for testing task partitioning algorithm. For the random testing method, the swarm is tested in 3000 random test cases. To keep the fitness evaluations between the random testing method and evolutionary testing method the same (at 3000), and given that 30 generations for each evolution are executed, 100 evolved test cases are produced. The swarm is tested in all newly generated evolved test cases, and the performance in each test case is measured and recorded. 100 random test cases are randomly chosen for results comparison.

Table 6.2 shows the result details of applying one-tailed Mann-Whitney U-Test to the experimental results at a significance level of 1%. The values of the average (mean) fitness score of the solutions (*avg*), standard deviation (*std*), sum of ranking (*SoR*), mean of ranking(*MoR*), and U-value are listed in the table. The p-value of the test is 0.00001, which is less than the value (0.01) of the significant level. In such case, the null hypothesis H_0 is rejected in the favor of the alternative hypothesis H_1 with a confident level of 99%. Hence, the use of evolved test cases has a positive effect on the ability to search-and-transport food to the nest of the task partitioning algorithm when compared to the random testing strategy.

Figure 6.3 presents a box-and-whisker plot which shows the distribution of the fitness values of empty test cases, random test cases, and evolved test cases. From the graph, both the mean and median of the fitness values of the random test cases are higher than those of the evolved

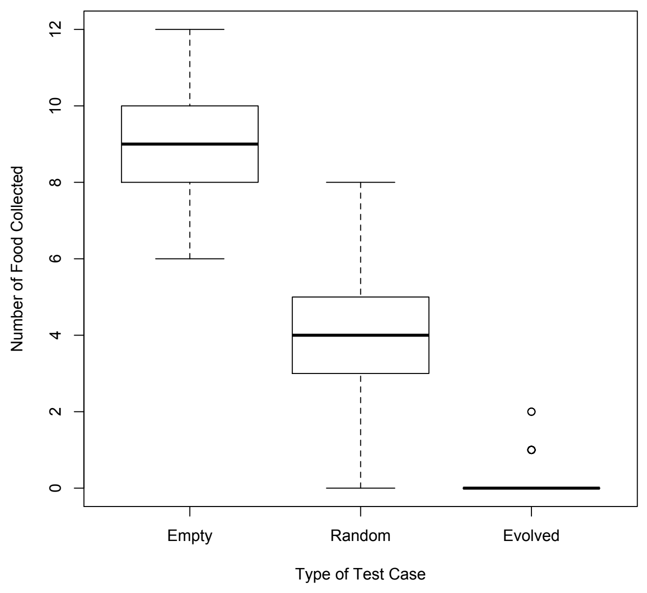


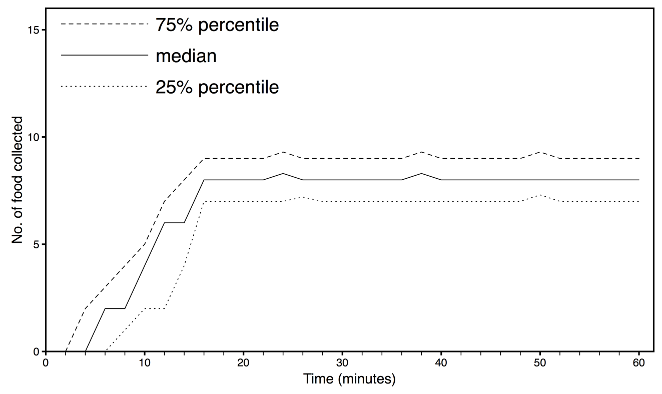
FIGURE 6.3: The distribution of the number of collected food items for empty test cases, random test cases, and evolved test cases.

test cases. Moreover, the static partitioning strategy performed extremely undesirable in evolved test cases as most of the fitness value are equal to 0. From the experimental results, the swarm equipped with static task partitioning has worse ability at search-and transporting the food to the nest in evolved test cases than random test cases.

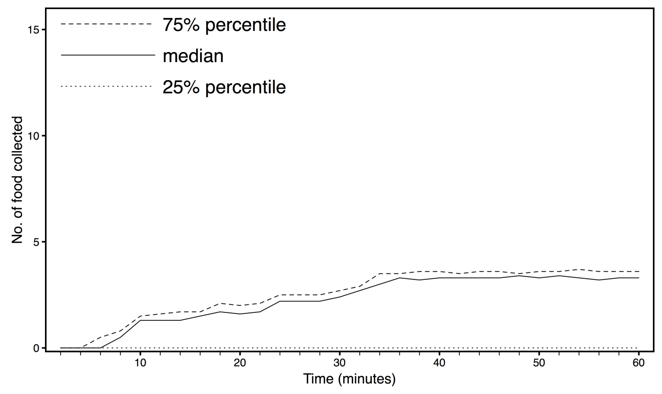
6.4.2 Convergence Time of Task Partitioning

As a matter of fact, it is more likely that the robots of the swarm are in the nest at the beginning of the experiments (the experiments carried out in section 5.4). The swarm needs to explore the environment first in order to converge to a steady state for food transportation. In this section, the effect of the evolutionary testing method is tested on the convergence time of task partitioning.

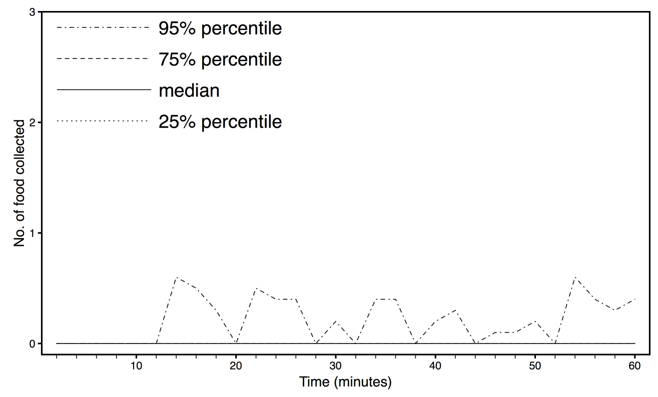
For each experiment in this section, all the robots in the swarm are deployed in the nest at the beginning. The swarm is tested in three kinds of test cases: no-obstacle test cases, random test cases, and evolved test cases. The execution time for each experiments increases from 200 seconds to 1 hour. Figure 6.4 shows the number of food items collected by the swarm every 2 simulated minutes; that is, the count of food items collected is reset to 0 after every 2 minutes.



(a) Empty Test Cases



(b) Random Test Cases



(c) Evolved Test Cases

FIGURE 6.4: The number of food items collected by the swarm every 2 simulated minutes in empty test cases, random test cases, and evolved test cases. The 25%, 50%, and 75% percentiles are plotted in each graph (the 95% percentile is plotted for (c) only). Note that the scale of the y-axis in (c) is different from that in (a) and (b).

For all three test cases, the swarm reaches a steady state. Figure 6.4(a) represents the experimental results when executing the swarm in empty test cases. The swarm is tested in empty test cases 100 times. For each experiment, the locations of the nest and the food resource remain the same, while the starting positions of the robots change (always stay inside the nest but might have different coordinates). From the graph, the swarm takes approximately 16 to 18 minutes to reach the steady state. The median rate of transporting food items to the nest is around 8 items for every 2 minutes.

Figure 6.4(b) shows the experimental results when testing the swarm in random test cases. 3000 random test cases are generated for the random testing method, and 100 of these are randomly selected for testing. From the graph, the swarm takes around 36 to 42 minutes to reach the steady state. Note that the 25% percentile line is always 0. This means that in at least 25% of the experiments, the swarm was unable to bring any food items back to the nest. The median rate of transporting food items to the nest is about 3 items for every 2 minutes.

Figure 6.4(c) plots the experimental results for testing the swarm in evolved test cases. In order to keep the fitness evaluations between the random testing method and the evolutionary testing method the same (at 3000), and given that 30 generations for each evolution are executed, 100 evolved test cases are generated. In the graph, the lines of the 25%, 50% (median), and 75% percentiles are all coincident with the x-axis. The line for the 95% percentile is the only visible one on the graph. From the graph, the swarm never reached the steady state in any of the experiments (duration of each experiment is 60 minutes).

By analysing the experiments for no food items (both random and evolved test cases), the reason for no food items is that most of the robots are trapped by the obstacles and there are too few robots left to search-and-transport the food items back to the nest. Moreover, there are a few extreme cases in which the swarm was unable to locate the food resource.

From Figure 6.4, the convergence time of the task partitioning in evolved test cases is the longest (the swarm never reaches the steady state within 60 minutes) among all three groups of experiments. From the above experimental results, it is reasonable to conclude that the evolved test cases presented in this case study did make it longer and harder for the swarm system to reach a steady state if the robots start in the nest.

6.5 Improving the Control Algorithm by Fault Removal

In section 6.1.1, the concept of total failure for task partitioning behaviour is introduced. Table 6.3 shows the number of total failures found by 100

TABLE 6.3: TOTAL FAILURES OF STATIC PARTITIONING STRATEGY FOUND DURING THE EXPERIMENTS

	Random Testing	Evolved Testing
Total Failures	3	82

random test cases and 100 evolved test cases during the experiments in section 6.5.3. The static partitioning algorithm is able to bring at least 1 food item back to the nest in 97% of the random test cases and in 18% of the evolved test cases.

6.5.1 Total Failure Analysis

In this section, a total failure analysis on the static partitioning strategy is carried out. Figure 6.5 shows an representative example of the trajectory of a robot which is trapped by certain obstacles during the experiment. In order not to be complicated by the trajectories of all the robots, only the trapped robot is shown in the testing environment. The black arrow on the robot shows the facing direction of the robot.

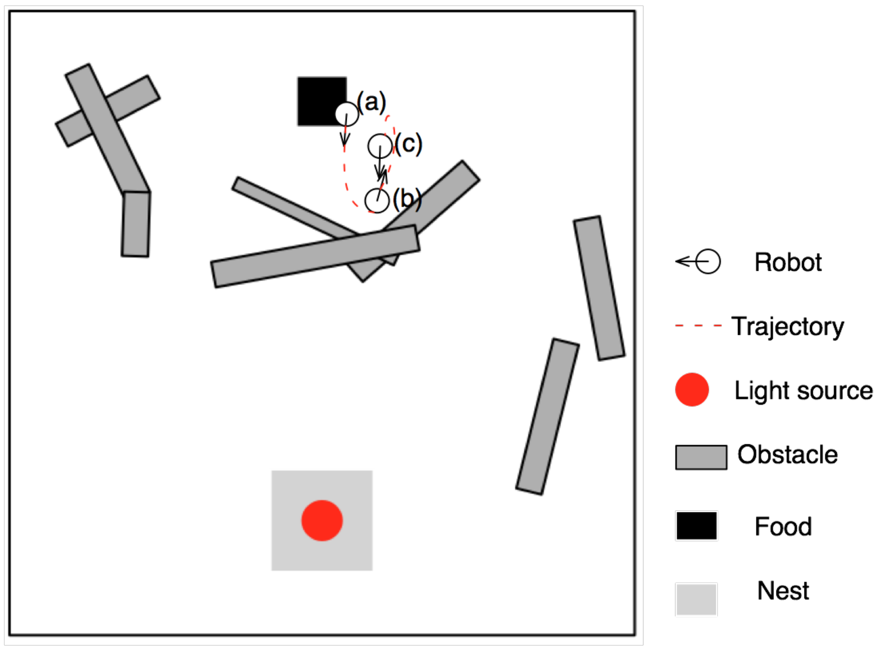


FIGURE 6.5: The trajectory of a trapped robot in an evolved test case for the static partitioning strategy.

After the robot finds the food resource, it grabs one food item (at point (a)) and then moves towards the nest according to the direction of the light source. When the robot encounters the obstacles, it arrives at point (b) by following the resultant force of the potential field method. When the obstacles are not in the detectable range of the obstacle-avoidance sensor, the repulsive force created by the obstacles disappears. The resultant force is now equal to the attractive force created by the light source at point (c), and the robot moves towards the light source again. After the robot meets the obstacles, it will reach point (b) again. From then on, the robot will move from point (b) to (c) then (b) continuously. This situation is named a local minimum in the potential field method. In the current situation, the robot can only get out of the trap if another robot accidentally pushes it out of this local minimum.

In the potential field method, a random walk (wandering around in the environment) might be the simplest way (but not efficient) to escape a local minimum (Barraquand, Langlois, and Latombe, 1992). In section 5.5.2.2, the robot can escape from a local minimum by a random walk if the beacons are out of its detectable range. If the beacons are always within the detectable range, the robot will be trapped until pushed out by other robots. However, for the static partitioning strategy, the light source is always within the detectable range of the robot. The strategy used for improving the VP algorithm does not work here.

6.5.2 Algorithm Improvement

In the potential field method, in order to perform an escaping method, e.g. a random walk, a method is required to detect whether the robot is trapped in a local minimum or not. However, in current potential field method research (Baxter et al., 2007; Koren and Borenstein, 1991; Hassan et al., 2017), certain global information is always required in order to detect a local minimum. There are many studies (Baxter et al., 2007; Hassan et al., 2017) showing that the potential field method is an efficient path planning method for multi-robotic system. As global information is forbidden in swarm robotics, a replacement for the path planning strategy in static partitioning method is required. Note that the control algorithm is improved in the field of reactive robotic systems (see section 5.5.2.1).

6.5.2.1 Path Planning - Bug Algorithm

The bug algorithm is a real-time obstacle avoidance method for mobile robots that was first introduced in (Lumelsky and Stepanov, 1986). In study (Lumelsky and Stepanov, 1986), two bug algorithms, Bug1 and Bug2, are proposed together. In Bug1 (Figure 6.6 (a)), when there is an obstacle between the robot and the goal, the robot circumnavigates (fully circles) the obstacle then leaves it at the point at which the distance to the goal is the shortest. In Bug2 (Figure 6.6 (b)), the robot follows the boundary

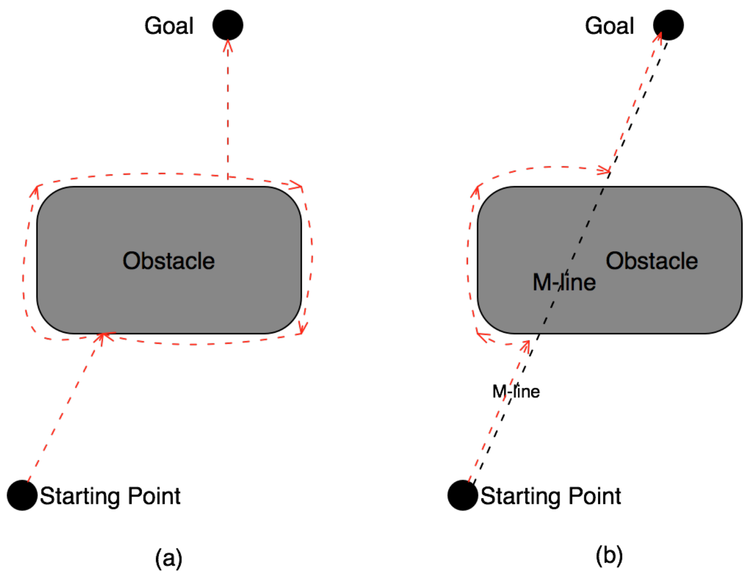


FIGURE 6.6: Illustrations of Bug1 algorithm (a) and Bug2 algorithm (b).

of the obstacle then leaves it when it is on the m-line, which connects the starting point and the goal. Bug2 seems to be more efficient than Bug1, but Bug2 is less effective in some cases, such as local loops. There are around 20 different types of Bug algorithm in robotics research. However, like the potential field method, all variants of the Bug algorithm require some amount of global information.

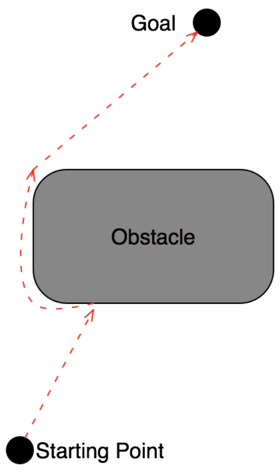


FIGURE 6.7: Illustration of BugS algorithm.

The bug algorithm has one advantage over the potential field method, which is that it has the ability to escape from certain local minima without access to global information. In this study, the Bug2 algorithm is modified in order to improve the path planning ability of the static partitioning strategy. The improved Bug2 is named as BugS (Bug for Swarm). In BugS (figure 6.7), the robot follows the boundary of the obstacle, and if there are no obstacles in the detectable range between the robot and the light source, the robot leaves the obstacle.

6.5.3 Experimental Results for the Improved Static Partitioning Strategy

In this section, three sets of experiments are carried out in order to determine whether the improved static partitioning strategy has a better performance than the original one. Note that, all the experiments in this section are carried out as follows: at the beginning of the experiments, three robots are deployed at the centre of the environment while the other three are located near the food resource. A new null hypothesis is proposed as follows:

> H_0: *the use of BugS algorithm makes no difference to the ability to search-and-transport food to the nest of the static partitioning strategy when compared to the potential field method, i.e.,* $f_{improved} = f_{original}$.

The alternative hypothesis is:

> H_1: *the use of BugS algorithm has a positive effect on the ability to search-and-transport food to the nest of the static partitioning strategy when compared to the potential field method, i.e.,* $f_{improved} < f_{original}$.

where $\mathrm{f}_{\mathrm{improved}}$ and $\mathrm{f}_{\mathrm{original}}$ represent the fitness values of the test cases when testing the improved algorithm and when testing the original algorithm, respectively. The lower the fitness value, the better the test cases are.

Table 6.4 shows the result details (three pairs) of applying one-tailed Mann-Whitney U-Test to the experimental results at a significance level of 1%. For all three pairs of experimental results, the p-value of the test is less than 0.00001, which is less than the value (0.01) of the significant level. This means that the null hypothesis H_0 is rejected in the favor of the alternative hypothesis H_1 with a confident level of 99%. Hence, the use of BugS algorithm has a positive effect on the ability to search-and-transport food to the nest of the static partitioning strategy when compared to the potential field method.

TABLE 6.4: THREE PAIRS OF RESULT DETAILS OF THE MANN-WHITNEY U-TEST

Test Case	avg	std	SoR	MoR	U-Value
S/S	0.21	0.48	14862.5	148.62	187.5
I/S	3.85	1.16	5237.5	52.38	9812.5
S/S	0.21	0.48	14052.5	140.52	997.5
I/I	1.81	1.01	6047.5	60.48	9002.5
S/I	0.11	0.31	14268.5	142.68	781.5
I/I	1.81	1.01	5831.5	58.32	9218.5

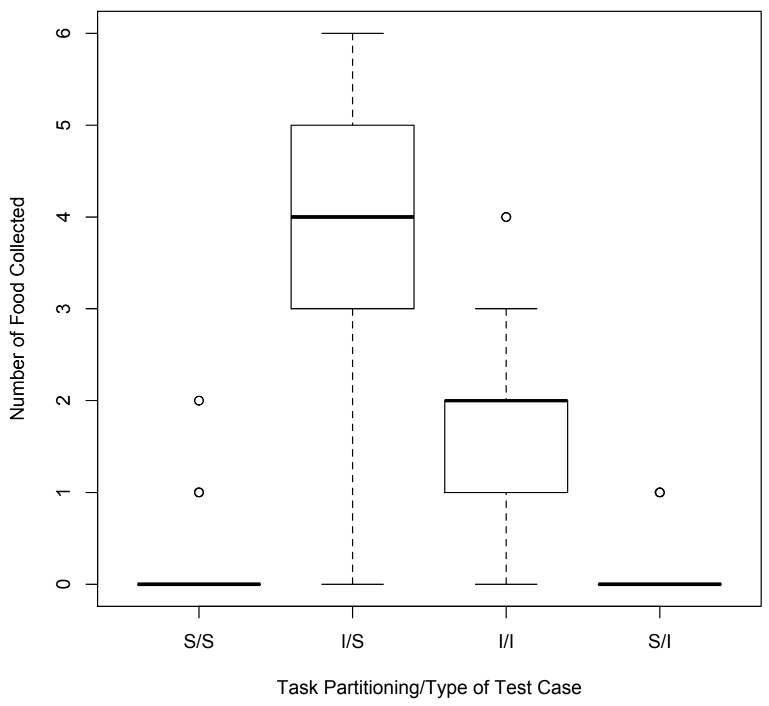


FIGURE 6.8: The distribution of the number of collected food items for testing the static partitioning strategy on test cases evolved based on the static partitioning strategy, testing the improved static partitioning strategy on test cases evolved based on the static partitioning strategy, testing improved the static partitioning strategy on test cases evolved based on the improved static partitioning strategy, and testing the static partitioning strategy on test cases evolved based on the improved static partitioning strategy.

Figure 6.8 shows the distribution of the number of food items collected for four sets of experiments. The S/S is the same as "Evolved" in Figure 6.3: both represent the distribution of the number of food items collected by static partitioning strategy in test cases evolved based on the static partitioning strategy. Recall that 100 evolved test cases were generated in section .

The I/S in Figure 6.8 shows the distribution of the number of food items collected by the improved static partitioning strategy in all 100 test cases evolved based on the static partitioning strategy. Following this set of experiments, 100 test cases are evolved based on the improved static partitioning strategy. Both improved the static partitioning strategy and the original static partitioning strategy are tested on these 100 newly generated test cases evolved based on the improved static partitioning strategy, and the distribution of the experimental results are shown on Figure 6.8 as I/I and S/I respectively.

From Figure 6.8, it can be seen that the improved static partitioning strategy not only performs better than the original one on test cases evolved based on the original strategy, but also performs better on its own test cases than on the static partitioning strategy on both sets of test cases. From the experimental results, the static partitioning strategy using the BugS algorithm has a better ability to search-and transport the food to the nest than that using the potential field method.

6.5.3.1 Convergence Time of the Improved Static Partitioning Strategy

In this section, experiments for testing the convergence ability of the improved static partitioning strategy are carried out. However, the performance of the convergence ability in this section is as disappointing as that in section 6.4.2. The swarm is still unable to converge to a steady state. By analysing the experiments, the major reason for no steady states is that the swarm has a very poor ability to locate the food resource in an environment with obstacles.

At the beginning of each experiment, the robots start from the nest and wander around in the environment with the goal of finding the food sources. Note that, even though the food resource is always in the same place for all experiments, the location of the food sources is always treated as unknown. If a robot meets an obstacle, it will be bounced back in a random direction. This makes it really hard for the robot to be able to find the location of the food resource.

To the best of the author's knowledge, no path planning algorithms are able to navigate to a goal effectively if the direction of the goal is unknown. This is also the reason why light source is always placed at the location of the nest in study (Pini et al., 2014). One solution to this is to place a light source at the centre of the food resource. However, this has nothing to do with improving the swarm control algorithm.

In conclusion, applying the BugS algorithm to the static partitioning strategy has no effect on improving the convergence ability of this strategy.

6.6 Chapter Summary

This chapter shows the feasibility of using the testing method developed on specific swarm behaviour to reveal failures in similar swarm behaviour. The experimental results presented in section 6.5.3 show that the evolved test cases can lead to worse swarm performance of task partitioning behaviour than the random test cases. The experimental results presented in section 6.4.2 show that the evolved test cases make it longer and harder for the task partitioning behaviour to reach a steady state if the robots starts in the nest. Section 6.5 shows the performance of the task partitioning behaviour proposed in study (Pini et al., 2014) can be improved by removing the fault identified by the evolutionary testing method.

Chapter 7

Evaluation and Conclusions

This chapter begins with the evaluation of the research hypothesis proposed in section 1.4. The contributions of the work presented in this thesis are summarized and its limitations are discussed. Potential future research directions are also suggested. Finally, the overall conclusion is discussed in the closing section of this chapter.

7.1 Evaluation of Research Hypothesis

The objective of the research in this thesis is to develop a testing method which is able to reveal failures in the control algorithm of a swarm robotic system. The research hypothesis postulated in section 1.4, which guided the entire thesis, is as follows:

> *It is possible to improve the dependability of a swarm robotic system by involving testing process during its development. The testing method presented in this thesis is more effective in revealing failures during the testing process than the random testing method.*

An evolutionary testing method is proposed in Chapter 3 as a solution to the testing problem in swarm robotics. Following this, three case studies of this testing method are presented. These case studies not only show the feasibility of the testing method presented in this thesis but also show that this testing method is able to reveal failures in three swarm behaviours. The experimental results in all three case studies show that this testing method can lead to worse swarm performance and reveal more total failures than the random testing method with the same number of computing evaluations. Moreover, the case study of flocking behaviour also shows that this testing method covers more failure types than the random testing method. As a result, the testing method presented in this thesis is able to reveal failures during the testing process and it is more effective than the random testing method.

In these case studies, the faults in each control algorithm are identified by analysing the total failures found during the experiments. The experimental results show that the swarm performs better after replacing the weak mechanism with a better one. Consequently, the dependability of the swarm robotic systems used in these case studies has been improved by means of fault removal.

7.2 Thesis Contributions

The main contributions of each chapter in this thesis are the following:

Chapter 3 - Evolutionary Testing Method for Swarm Robotic Systems

The testing method proposed in this chapter is the first known testing method for swarm robotic systems. Evolutionary testing methods in computer systems and autonomous agents are adapted to identify faults in swarm control algorithm. This testing method can be used as the basis of a testing method for specific swarm behaviour, which is the work published in (Wei, Timmis, and Alexander, 2017). The experimental infrastructure introduced in this chapter can be used as an assistance tool in developing a swarm robotic system.

Chapter 4 - Testing Method for Flocking Behaviour

The work presented in this chapter shows the feasibility of the testing method presented in this thesis and can be used as a guideline for the development of testing methods for other swarm behaviour. Both metrics and failure classification are newly defined for flocking behaviour. The experimental results show that the testing method presented in this thesis can lead to worse swarm performance and it covers more failure types than the random testing method. The algorithm improvement section shows that the performance of the flocking control algorithm is improved by removing the faults identified. Consequently, this testing method can be used as a test-driven development tool during the development of a swarm robotic system.

Chapter 5 - Testing Method for Foraging Behaviour

The work presented in this chapter shows that the testing method presented in this thesis is able to test more than one type of swarm behaviour. The experimental results show that this testing method can lead to worse swarm performance of ant foraging behaviour than the random testing method. Finally, the performance of the ant foraging algorithm is improved in the algorithm improvement section by removing the faults identified.

Chapter 6 - Testing Method for Task Partitioning Behaviour

The work presented in this chapter shows the feasibility of using an established testing method to test similar swarm behaviour. The experimental results show that the testing method established can lead to worse swarm performance of task partitioning behaviour than the random testing method. The algorithm improvement section shows that the performance of task partitioning algorithm can be improved by removing the faults identified.

7.3 Limitations

The work presented in this thesis is limited by the fact that all of the experiments are carried out in simulation and they have never been implemented in the real world. However, the limitations in this thesis are different from the limitations of general robotic research, such as control algorithm research, and design method research, which is carried out in simulation. The largest problem found in conducting experiments in simulation in general robotic research is the "reality gap" between the simulation and the real world. Due to the "reality gap", it is frequently assumed that a robotic system which functions correctly in a simulator might function incorrectly in the real world (Varela and Bourgine, 1992; Jakobi, Husbands, and Harvey, 1995). If a swarm control algorithm fails in simulation, the chance that it will fail in the real world is very high. As a result, there is a strong possibility that the testing method presented in this thesis is able to reveal failures during the execution of swarm robotic systems. However, as the testing method presented in this thesis is to test the overall behaviour of the swarm, both failures (both total failure and partial failure) of individual robots and noises of sensors and actuators are ignored in this thesis. The cause of failure in a swarm in this thesis is a fault in the control algorithm. After implementing this testing method on real robots, there is a possibility that a swarm fails due to the failures of individual robots or noises of sensors and actuators before the fault in the control algorithm can be identified.

Another limitation of the research in this thesis is the evolutionary process of the test cases in physical world. Section 3.3.4 shows the working procedure of the test case generator. Test cases are generated on the basis of the control algorithm provided. For each generation, the control algorithm is tested on the offspring of the previous generation. If the testing method is implemented on real robots, human effort is required to move the obstacles in the environment, or even worse, the obstacle has to be replaced with a new one when the descriptor of the obstacles has changed (various shapes of obstacles need to be built). Consequently, implementing the testing method presented in this thesis is particularly expensive and time-consuming, and require a great deal of human effort.

7.4 Future Work

Opportunities for future research based on this thesis are suggested as follows:

Testing physical robots using test cases generated in simulation

> Further research can be carried out to determine the usefulness of testing physical swarm robotic systems using the test cases generated in the simulator. The control algorithm of the physical swarm can be applied to the swarm in the simulator. Then the test case generator described in section 3.3.4 can be used to generate test cases according to the control algorithm used in the physical swarm. The physical swarm can then be tested in real test cases to determine the ability of the test cases generated in simulation to reveal failures in the physical swarm. In this situation, only the evolved test cases need to be built in the real world.

Implementing virtual obstacles using robot-in-the-loop design methodology

> As previously mentioned in section 2.3.2.4, a robot-in-the-loop design methodology is presented in study (Azarnasab and Hu, 2007). Further research which uses a combination of physical robots and virtual obstacles can be carried out. The robots in the simulator in this thesis can be replaced with real robots. The real robots can use a combination of virtual and real sensors where the virtual sensors are used to detect the existence

of the virtual obstacles while the real sensors are used to detect other robots or the food items. No physical test cases are required in this situation. However, the "reality gap" still exists as simulation is involved.

Carrying out more case studies on other swarm behaviours

More swarm behaviours, such as aggregation, pattern formation, collective exploration, collective transportation, and so on, can be studied to determine whether or not the testing method presented in this thesis is only suitable for the three swarm behaviours studied in this thesis.

Adding various elements to the testing method for future swarm robotic systems

At the early stage of this thesis, global information was shared using Headquarters to achieve flocking behaviour in the simulator. The swarm sharing global information functions "perfectly", i.e. no agents get lost, in the evolved test cases generated by the testing method presented in this thesis. In order to make this swarm fail, the following two elements need to be added to the test cases:

- **Moving Obstacle**: is able to move around in the environment at a slow speed
- **Anchoring Obstacle**: one or more light sources are on top of it.

The robots distinguish its flockmates from the obstacles by sensing the light source on its flockmates. If a robot senses a light source with the same colour as its own, it treats the object with the light source as its flockmates. When the anchoring obstacles emit light with the same colour as the robots, the robots treat the obstacles as their flockmates and the robots anchor around the obstacle. A possible solution to this could be the algorithm proposed in study (Christensen, OGrady, and Dorigo, 2009). The robots can flash their LEDs at the same frequency in order to resist the disturbance caused by other light sources. As the swarm robotics becomes more mature, further research could be carried out into various elements that can be added to the testing method in order to identify faults in the swarm robotic systems.

7.5 Conclusion

The objective of the research in this thesis has been to develop a testing method which is able to reveal failures during the testing process when developing a swarm robotic system. The dependability of the system might potentially be improved by removing the faults identified. The case studies carried out in this thesis have shown the feasibility, adaptability, and reusability of the testing method. Moreover, the dependability of all three swarm robotic systems developed in the thesis is improved by means of fault removal.

In conclusion, the objectives of this thesis have been met and the testing method presented in this thesis can be used to help the developers of swarm robotic systems to design and calibrate their control algorithm thereby assuring the dependability of swarm robotic systems.

Bibliography

Abbott, Russ (2006). “Emergence explained: Abstractions: Getting epiphenomena to do real work”. In: *Complexity* 12.1, pp. 13–26.

Abran, Alain, James W Moore, Pierre Bourque, Robert Dupuis, and Leonard L Tripp (2004). *Guide to the software engineering body of knowledge: 2004 version SWEBOK*. IEEE Computer Society.

Adrion, W Richards, Martha A Branstad, and John C Cherniavsky (1982). “Validation, verification, and testing of computer software”. In: *ACM Computing Surveys (CSUR)* 14.2, pp. 159–192.

Akyildiz, Ian F, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci (2002). “Wireless sensor networks: a survey”. In: *Computer networks* 38.4, pp. 393–422.

Alden, Kieran, Mark Read, Jon Timmis, Paul S Andrews, Henrique Veiga-Fernandes, and Mark Coles (2013). “Spartan: a comprehensive tool for understanding uncertainty in simulations of biological systems”. In: *PLoS Comput Biol* 9.2, e1002916.

Alden, Kieran, Mark Read, Paul S Andrews, Jon Timmis, and Mark Coles (2014). “Applying spartan to understand parameter uncertainty in simulations”. In: *The R Journal* 6.2, pp. 1–10.

Ammann, Paul and Jeff Offutt (2008). *Introduction to Software Testing*.

Antonelli, Gianluca, Filippo Arrichiello, and Stefano Chiaverini (2008). “Flocking for multi-robot systems via the null-space-based behavioral control”. In: *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*. IEEE, pp. 1409–1414.

Ashlock, Daniel A, Theodore W Manikas, and Kaveh Ashenayi (2006). “Evolving a diverse collection of robot path planning problems”. In: *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*. IEEE, pp. 1837–1844.

Avizienis, Algirdas, Jean-Claude Laprie, Brian Randell, et al. (2001). *Fundamental concepts of dependability*. University of Newcastle upon Tyne, Computing Science.

Azarnasab, Ehsan and Xiaolin Hu (2007). “An integrated multi-robot test bed to support incremental simulation-based design”. In: *System of Systems Engineering, 2007. SoSE’07. IEEE International Conference on*. IEEE, pp. 1–7.

Barraquand, Jerome, Bruno Langlois, and J-C Latombe (1992). “Numerical potential field techniques for robot path planning”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 22.2, pp. 224–241.

Barth, Eric J (2003). “A dynamic programming approach to robotic swarm navigation using relay markers”. In: *American Control Conference, 2003. Proceedings of the 2003*. Vol. 6. IEEE, pp. 5264–5269.

Baxter, Joseph L, EK Burke, Jonathan M Garibaldi, and Mark Norman (2007). “Multi-robot search and rescue: A potential field based approach”. In: *Autonomous robots and agents*. Springer, pp. 9–16.

Beck, Kent (2003). *Test-driven development: by example*. Addison-Wesley Professional.

Beizer, Boris (2003). *Software testing techniques*. Dreamtech Press.

Beni, Gerardo and Jing Wang (1993). “Swarm intelligence in cellular robotic systems”. In: *Robots and Biological Systems: Towards a New Bionics?* Springer, pp. 703–712.

Bird, David L. and Carlos Urias Munoz (1983). “Automatic generation of random self-checking test cases”. In: *IBM systems journal* 22.3, pp. 229–245.

Bjerknes, Jan and Alan Winfield (2013). “On fault tolerance and scalability of swarm robotic systems”. In: *Distributed autonomous robotic systems*, pp. 431–444.

Blanchard, Benjamin S, Wolter J Fabrycky, and Walter J Fabrycky (1990). *Systems engineering and analysis*. Vol. 4. Prentice Hall Englewood Cliffs, NJ.

Bonabeau, Eric, Marco Dorigo, and Guy Theraulaz (1999). *Swarm intelligence: from natural to artificial systems*. 1. Oxford university press.

Bonabeau, Eric, Guy Theraulaz, Jean-Louls Deneubourg, Serge Aron, and Scott Camazine (1997). “Self-organization in social insects”. In: *Trends in Ecology & Evolution* 12.5, pp. 188–193.

Box, George EP, J Stuart Hunter, and William Gordon Hunter (2005). *Statistics for experimenters: design, innovation, and discovery*. Vol. 2. Wiley-Interscience New York.

Brambilla, Manuele, Carlo Pinciroli, Mauro Birattari, and Marco Dorigo (2012). “Property-driven design for swarm robotics”. In: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent*

Systems-Volume 1. International Foundation for Autonomous Agents and Multiagent Systems, pp. 139–146.

Brambilla, Manuele, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo (2013). "Swarm robotics: a review from the swarm engineering perspective". In: *Swarm Intelligence* 7.1, pp. 1–41.

Brian, Michael Vaughan (2012). *Social insects: ecology and behavioural biology*. Springer Science & Business Media.

Brutschy, Arne, Giovanni Pini, Carlo Pinciroli, Mauro Birattari, and Marco Dorigo (2014). "Self-organized task allocation to sequentially interdependent tasks in swarm robotics". In: *Autonomous agents and multi-agent systems* 28.1, pp. 101–125.

Buchanan, Edgar, Andrew Pomfret, and Jon Timmis (2016). "Dynamic Task Partitioning for Foraging Robot Swarms". In: *International Conference on Swarm Intelligence*. Springer, pp. 113–124.

Burns, Alan, John McDermid, and J Dobson (1992). "On the meaning of safety and security". In: *The Computer Journal* 35.1, pp. 3–15.

Camazine, Scott (2003). *Self-organization in biological systems*. Princeton University Press.

Campo, Alexandre and Marco Dorigo (2007). "Efficient multi-foraging in swarm robotics". In: *Advances in Artificial Life*, pp. 696–705.

Çelikkanat, Hande and Erol Şahin (2010). "Steering self-organized robot flocks through externally guided individuals". In: *Neural Computing and Applications* 19.6, pp. 849–865.

Christensen, Anders Lyhne, Rehan OGrady, and Marco Dorigo (2009). "From fireflies to fault-tolerant swarms of robots". In: *IEEE Transactions on Evolutionary Computation* 13.4, pp. 754–766.

Clarke, John, Jose Javier Dolado, Mark Harman, Rob Hierons, Bryan Jones, Mary Lumkin, Brian Mitchell, Spiros Mancoridis, Kearton Rees, Marc Roper, et al. (2003). "Reformulating software engineering as a search problem". In: *IEE Proceedings-software* 150.3, pp. 161–175.

Cliff, Dave, Phil Husbands, and Inman Harvey (1993). "Explorations in evolutionary robotics". In: *Adaptive behavior* 2.1, pp. 73–110.

Cortés, Jorge and Francesco Bullo (2005). "Coordination and geometric optimization via distributed dynamical systems". In: *SIAM Journal on Control and Optimization* 44.5, pp. 1543–1574.

Cortes, Jorge, Sonia Martinez, and Francesco Bullo (2005). "Spatially-distributed coverage optimization and control with limited-range interactions". In: *ESAIM: Control, Optimisation and Calculus of Variations* 11.4, pp. 691–719.

Couceiro, Micael S, Rui P Rocha, NM Fonseca Ferreira, and JA Tenreiro Machado (2012a). "Introducing the fractional-order Darwinian PSO". In: *Signal, Image and Video Processing* 6.3, pp. 343–350.

Couceiro, Micael S, Fernando ML Martins, Rui P Rocha, and Nuno MF Ferreira (2012b). "Introducing the fractional order robotic Darwinian PSO". In: *AIP Conference Proceedings*. Vol. 1493. 1. AIP, pp. 242–251.

Couceiro, Micael S, Patricia A Vargas, Rui P Rocha, and Nuno MF Ferreira (2013a). "Benchmark of swarm robotics distributed techniques in a search task". In: *Robotics and Autonomous Systems* 62.2, pp. 200–213.

Couceiro, Micael S, Rui P Rocha, Nuno MF Ferreira, and Patricia A Vargas (2013b). "Darwinian robotic swarms for exploration with minimal communication". In: *Evolutionary Computation (CEC), 2013 IEEE Congress on*. IEEE, pp. 127–134.

Crespi, Valentino, Aram Galstyan, and Kristina Lerman (2008). "Top-down vs bottom-up methodologies in multi-agent system design". In: *Autonomous Robots* 24.3, pp. 303–313.

Davis, Lawrence (1991). *Handbook of genetic algorithms*. Van Nostrand Reinhold.

De Jong, Kenneth Alan (1975). *Analysis of the behavior of a class of genetic adaptive systems*.

Derderian, Karnig, Robert M Hierons, Mark Harman, and Qiang Guo (2006). "Automated unique input output sequence generation for conformance testing of FSMs". In: *The Computer Journal* 49.3, pp. 331–344.

Dorigo, Marco, Mauro Birattari, and Thomas Stutzle (2006). "Ant colony optimization". In: *IEEE computational intelligence magazine* 1.4, pp. 28–39.

Dorigo, Marco, Gianni Di Caro, and Luca M Gambardella (1999). "Ant algorithms for discrete optimization". In: *Artificial life* 5.2, pp. 137–172.

Dorigo, Marco and Luca Maria Gambardella (1997). "Ant colony system: a cooperative learning approach to the traveling salesman problem". In: *IEEE Transactions on evolutionary computation* 1.1, pp. 53–66.

Dorigo, Marco, Vito Trianni, Erol Şahin, Roderich Groß, Thomas H Labella, Gianluca Baldassarre, Stefano Nolfi, Jean-Louis Deneubourg, Francesco Mondada, Dario Floreano, et al. (2004). "Evolving self-organizing behaviors for a swarm-bot". In: *Autonomous Robots* 17.2, pp. 223–245.

Dorigo, Marco, Elio Tuci, Vito Trianni, Roderich Groß, Shervin Nouyan, Christos Ampatzis, Thomas Halva Labella, Michael Bonani, F Mondada, et al. (2006). *SWARM-BOT: Design and implementation of colonies of self-assembling robots*. Tech. rep. IEEE Computational Intelligence Society.

Drogoul, Alexis and Jacques Ferber (1993). "Some experiments with foraging robots". In: *From Animals to Animats* 2, p. 451.

Dudek, Gregory, Michael RM Jenkin, Evangelos Milios, and David Wilkes (1996). "A taxonomy for multi-agent robotics". In: *Autonomous Robots* 3.4, pp. 375–397.

Eberhart, Russell and James Kennedy (1995). "A new optimizer using particle swarm theory". In: *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on*. IEEE, pp. 39–43.

Eglese, RW (1990). "Simulated annealing: a tool for operational research". In: *European journal of operational research* 46.3, pp. 271–281.

Ferber, Jacques (1999). *Multi-agent systems: an introduction to distributed artificial intelligence*. Vol. 1. Addison-Wesley Reading.

Ferrante, Eliseo, Ali Emre Turgut, Cristián Huepe, Alessandro Stranieri, Carlo Pinciroli, and Marco Dorigo (2012). "Self-organized flocking with a mobile robot swarm: a novel motion control method". In: *Adaptive Behavior* 20.6, pp. 460–477.

Forrester, Justin E and Barton P Miller (2000). "An empirical study of the robustness of Windows NT applications using random testing". In: *Proceedings of the 4th USENIX Windows System Symposium*. Seattle, pp. 59–68.

FOWLER, HAROLD G and SW Robinson (1979). "Foraging by Atta sexdens (Formicidae: Attini): seasonal patterns, caste and efficiency". In: *Ecological Entomology* 4.3, pp. 239–247.

Franklin, Stan and Art Graesser (1996). "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents". In: *International Workshop on Agent Theories, Architectures, and Languages*. Springer, pp. 21–35.

Friedmann, Martin (2010). "Simulation of autonomous robot teams with adaptable levels of abstraction". PhD thesis. Technische Universität.

Gerkey, Brian, Richard T Vaughan, and Andrew Howard (2003). "The player/stage project: Tools for multi-robot and distributed sensor systems". In: *Proceedings of the 11th international conference on advanced robotics*. Vol. 1, pp. 317–323.

Glover, Fred (1990). "Tabu search: A tutorial". In: *Interfaces* 20.4, pp. 74–94.

Godefroid, Patrice, Nils Klarlund, and Koushik Sen (2005). "DART: directed automated random testing". In: *ACM Sigplan Notices*. Vol. 40. 6. ACM, pp. 213–223.

Goldberg, David E, Kalyanmoy Deb, and James H Clark (1991). "Genetic algorithms, noise, and the sizing of populations". In: *Urbana* 51, p. 61801.

Goss, Simon and Jean-Louis Deneubourg (1992). “Harvesting by a group of robots”. In: *Proceedings of the First European Conference on Artificial Life*, pp. 195–204.

Grefenstette, John J (1986). “Optimization of control parameters for genetic algorithms”. In: *IEEE Transactions on systems, man, and cybernetics* 16.1, pp. 122–128.

Gruau, Frédéric (1994). “Automatic definition of modular neural networks”. In: *Adaptive behavior* 3.2, pp. 151–183.

Gu, Dongbing and Huosheng Hu (2008). “Using fuzzy logic to design separation function in flocking algorithms”. In: *IEEE Transactions on fuzzy Systems* 16.4, pp. 826–838.

Gu, Yaoyao, Doruk Bozdağ, Robert W Brewer, and Eylem Ekici (2006). “Data harvesting with mobile elements in wireless sensor networks”. In: *Computer Networks* 50.17, pp. 3449–3465.

Hart, Adam G, Carl Anderson, and Francis L Ratnieks (2002). “Task partitioning in leafcutting ants”. In: *Acta ethologica* 5.1, pp. 1–11.

Hart, Adam G and Francis LW Ratnieks (2001). “Task partitioning, division of labour and nest compartmentalisation collectively isolate hazardous waste in the leafcutting ant Atta cephalotes”. In: *Behavioral Ecology and Sociobiology* 49.5, pp. 387–392.

Hassan, Abdelrahman M, Catherine M Elias, Omar M Shehata, and Elsayed I Morgan (2017). *A Global Integrated Artificial Potential Field/Virtual Obstacles Path Planning Algorithm for Multi-Robot System Applications*.

Haupt, Randy L and Sue Ellen Haupt (2004). *Practical genetic algorithms*. John Wiley & Sons.

Heddaya, Abdelsalam, Abdelsalam Helal, et al. (1997). *Reliability, availability, dependability and performability: A user-centered view*. Tech. rep. Boston University Computer Science Department.

Hinchey, Michael G, Roy Sterritt, and Chris Rouff (2007). “Swarms and swarm intelligence”. In: *Computer* 40.4.

Hoff, Nicholas R, Amelia Sagoff, Robert J Wood, and Radhika Nagpal (2010). “Two foraging algorithms for robot swarms using only local communication”. In: *Robotics and Biomimetics (ROBIO), 2010 IEEE International Conference on*. IEEE, pp. 123–130.

Holland, John H (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

Holland, Owen, John Woods, Renzo De Nardi, and Adrian Clark (2005). “Beyond swarm intelligence: the ultraswarm”. In: *Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE*. IEEE, pp. 217–224.

Houhamdi, Zina and Belkacem Athamena (2011). "Structured system test suite generation process for multi-agent system". In: *International Journal on Computer Science and Engineering* 3.4, pp. 1681–1688.

Jacobson, Ivar, Grady Booch, James Rumbaugh, James Rumbaugh, and Grady Booch (1999). *The unified software development process*. Vol. 1. Addison-wesley Reading.

Jager, Markus and Bernhard Nebel (2001). "Decentralized collision avoidance, deadlock detection, and deadlock resolution for multiple mobile robots". In: *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*. Vol. 3. IEEE, pp. 1213–1219.

Jakobi, Nick (1997). "Evolutionary robotics and the radical envelope-of-noise hypothesis". In: *Adaptive behavior* 6.2, pp. 325–368.

Jakobi, Nick, Phil Husbands, and Inman Harvey (1995). "Noise and the reality gap: The use of simulation in evolutionary robotics". In: *Advances in artificial life*, pp. 704–720.

Kaelbling, Leslie Pack, Michael L Littman, and Andrew W Moore (1996). "Reinforcement learning: A survey". In: *Journal of artificial intelligence research* 4, pp. 237–285.

Karaboga, Dervis (2005). *An idea based on honey bee swarm for numerical optimization*. Tech. rep. Technical report-tr06, Erciyes university.

Kelly, ID and DA Keating (1996). "Flocking by the fusion of sonar and active infrared sensors on physical autonomous mobile robots". In: *Proceedings of The Third Int. Conf. on Mechatronics and Machine Vision in Practice*. Vol. 1, pp. 1–4.

Khatib, Oussama (1986). "Real-time obstacle avoidance for manipulators and mobile robots". In: *The international journal of robotics research* 5.1, pp. 90–98.

Koren, Yoram and Johann Borenstein (1991). "Potential field methods and their inherent limitations for mobile robot navigation". In: *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*. IEEE, pp. 1398–1404.

Kube, C Ronald and Hong Zhang (1993). "Collective robotics: From social insects to robots". In: *Adaptive behavior* 2.2, pp. 189–218.

Kwong, Henry and Christian Jacob (2003). "Evolutionary exploration of dynamic swarm behaviour". In: *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*. Vol. 1. IEEE, pp. 367–374.

Labella, Thomas H, Marco Dorigo, and Jean-Louis Deneubourg (2006). "Division of labor in a group of robots inspired by ants' foraging behavior". In: *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 1.1, pp. 4–25.

Lambrinos, Dimitrios, Ralf Möller, Thomas Labhart, Rolf Pfeifer, and Rüdiger Wehner (2000). "A mobile robot employing insect strategies for navigation". In: *Robotics and Autonomous systems* 30.1, pp. 39–64.

Langdon, William B and Riccardo Poli (2013). *Foundations of genetic programming*. Springer Science & Business Media.

Laprie, J-C (1995). "Dependability of computer systems: concepts, limits, improvements". In: *Software Reliability Engineering, 1995. Proceedings., Sixth International Symposium on*. IEEE, pp. 2–11.

Laprie, Jean-Claude (1985). "Dependable computing and fault-tolerance". In: *Digest of Papers FTCS-15*, pp. 2–11.

Laprie, Jean-Claude (1993). "Dependability: from concepts to limits". In: *Safecomp'93: 12th International Conference on Computer Safety, Reliability, and Security*. Springer, pp. 157–168.

Lerman, Kristina and Aram Galstyan (2002). "Mathematical model of foraging in a group of robots: Effect of interference". In: *Autonomous Robots* 13.2, pp. 127–141.

Lerman, Kristina, Alcherio Martinoli, and Aram Galstyan (2004). "A review of probabilistic macroscopic models for swarm robotic systems". In: *International Workshop on Swarm Robotics*. Springer, pp. 143–152.

Leveson, Nancy G and Clark S Turner (1993). "An investigation of the Therac-25 accidents". In: *Computer* 26.7, pp. 18–41.

Lindhé, Magnus, Petter Ogren, and Karl Henrik Johansson (2005). "Flocking with obstacle avoidance: A new distributed coordination algorithm based on voronoi partitions". In: *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE, pp. 1785–1790.

Liu, Wenguo and Alan FT Winfield (2010). "Modeling and optimization of adaptive foraging in swarm robotic systems". In: *The International Journal of Robotics Research* 29.14, pp. 1743–1760.

Lumelsky, V and Alexander Stepanov (1986). "Dynamic path planning for a mobile automaton with limited information on the environment". In: *IEEE transactions on Automatic control* 31.11, pp. 1058–1063.

Martinez, Sonia, Jorge Cortes, and Francesco Bullo (2007). "Motion coordination with distributed information". In: *IEEE Control Systems* 27.4, pp. 75–88.

Martinoli, Alcherio, Auke Jan Ijspeert, and Luca Maria Gambardella (1999). "A probabilistic model for understanding and comparing collective aggregation mechanisms". In: *European Conference on Artificial Life*. Springer, pp. 575–584.

Marusteri, Marius and Vladimir Bacarea (2010). "Comparing groups for statistical differences: how to choose the right statistical test?" In: *Biochemia medica: Biochemia medica* 20.1, pp. 15–32.

Massink, Mieke, Manuele Brambilla, Diego Latella, Marco Dorigo, and Mauro Birattari (2012). "Analysing Robot Swarm Decision-Making with Bio-PEPA." In: *ANTS*. Springer, pp. 25–36.

Matarić, Maja J (1995). "Designing and understanding adaptive group behavior". In: *Adaptive Behavior* 4.1, pp. 51–80.

Matarić, Maja J (1997). "Reinforcement learning in the multi-robot domain". In: *Autonomous Robots* 4.1, pp. 73–83.

Mataric, Maja J, Martin Nilsson, and Kristian T Simsarin (1995). "Cooperative multi-robot box-pushing". In: *iros*. IEEE, p. 3556.

McMinn, Phil and Mike Holcombe (2003). "The state problem for evolutionary testing". In: *Genetic and Evolutionary Computation—GECCO 2003*. Springer, pp. 214–214.

Meyer, Jean-Arcady, Phil Husbands, and Inman Harvey (1998). "Evolutionary robotics: A survey of applications and problems". In: *Evolutionary Robotics*. Springer, pp. 1–21.

Mitchell, Melanie (1998). *An introduction to genetic algorithms*. MIT Press.

Moeslinger, Christoph, Thomas Schmickl, and Karl Crailsheim (2011). "A minimalist flocking algorithm for swarm robots". In: *Advances in Artificial Life. Darwin Meets von Neumann*, pp. 375–382.

Mogilner, Alex and Leah Edelstein-Keshet (1996). "Spatio-angular order in populations of self-aligning objects: formation of oriented patches". In: *Physica D: Nonlinear Phenomena* 89.3-4, pp. 346–367.

Nguyen, Cu D, Simon Miles, Anna Perini, Paolo Tonella, Mark Harman, and Michael Luck (2012). "Evolutionary testing of autonomous software agents". In: *Autonomous Agents and Multi-Agent Systems* 25.2, pp. 260–283.

Nguyen, Cu Duy, Anna Perini, and Paolo Tonella (2009). "Goal-oriented testing for MASs". In: *International Journal of Agent-Oriented Software Engineering* 4.1, pp. 79–109.

Nguyen, Duy Cu (2009). "Testing techniques for software agents". PhD thesis. University of Trento.

Nolfi, Stefano, Josh Bongard, Phil Husbands, and Dario Floreano (2016). "Evolutionary robotics". In: *Springer Handbook of Robotics*. Springer, pp. 2035–2068.

Nouyan, Shervin and Marco Dorigo (2006). "Chain based path formation in swarms of robots". In: *ANTS Workshop*. Springer, pp. 120–131.

O'Dowd, Paul, Alan FT Winfield, and Matthew Studley (2010). *Towards accelerated distributed evolution for adaptive behaviours in swarm robotics*.

O'Dowd, Paul J, Alan FT Winfield, and Matthew Studley (2011). "The distributed co-evolution of an embodied simulator and controller for swarm robot behaviours". In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, pp. 4995–5000.

Olfati-Saber, Reza (2006). "Flocking for multi-agent dynamic systems: Algorithms and theory". In: *IEEE Transactions on automatic control* 51.3, pp. 401–420.

Ostergaard, Esben H, Gaurav S Sukhatme, and Maja J Matari (2001). "Emergent bucket brigading: a simple mechanisms for improving performance in multi-robot constrained-space foraging tasks". In: *Proceedings of the fifth international conference on Autonomous agents*. ACM, pp. 29–30.

Pan, Jiantao (1999). "Software testing". In: *Dependable Embedded Systems* 5, p. 2006.

Panait, Liviu and Sean Luke (2005). "Cooperative multi-agent learning: The state of the art". In: *Autonomous agents and multi-agent systems* 11.3, pp. 387–434.

Pargas, Roy P, Mary Jean Harrold, and Robert R Peck (1999). "Test-data generation using genetic algorithms". In: *Software Testing Verification and Reliability* 9.4, pp. 263–282.

Parker, Lynne E (1994). "ALLIANCE: An architecture for fault tolerant, cooperative control of heterogeneous mobile robots". In: *Intelligent Robots and Systems' 94.'Advanced Robotic Systems and the Real World', IROS'94. Proceedings of the IEEE/RSJ/GI International Conference on*. Vol. 2. IEEE, pp. 776–783.

Pasteels, Jacques M and Jean-Louis Deneubourg (1987). *From individual to collective behavior in social insects*. Birkhauser.

Payton, David, Mike Daily, Regina Estowski, Mike Howard, and Craig Lee (2001). "Pheromone robotics". In: *Autonomous Robots* 11.3, pp. 319–324.

Penders, Jacques, Lyuba Alboul, Ulf Witkowski, Amir Naghsh, Joan Saez-Pons, Stefan Herbrechtsmeier, and Mohamed El-Habbal (2011). "A robot swarm assisting a human fire-fighter". In: *Advanced Robotics* 25.1-2, pp. 93–117.

Pham, Duc Truong and Michele Castellani (2009). "The bees algorithm: modelling foraging behaviour to solve continuous optimization problems". In: *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science* 223.12, pp. 2919–2938.

Pinciroli, Carlo, Vito Trianni, Rehan O'Grady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni Di Caro, Frederick Ducatelle, et al. (2011). "ARGoS: a modular, multi-engine simulator for heterogeneous swarm robotics". In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, pp. 5027–5034.

Pinciroli, Carlo, Vito Trianni, Rehan O'Grady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni Di Caro, Frederick Ducatelle, et al. (2012). "ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems". In: *Swarm intelligence* 6.4, pp. 271–295.

Pini, Giovanni, Arne Brutschy, Marco Frison, Andrea Roli, Marco Dorigo, and Mauro Birattari (2011a). "Task partitioning in swarms of robots: An adaptive method for strategy selection". In: *Swarm Intelligence* 5.3, pp. 283–304.

Pini, Giovanni, Arne Brutschy, Mauro Birattari, and Marco Dorigo (2011b). "Task partitioning in swarms of robots: reducing performance losses due to interference at shared resources". In: *Informatics in Control Automation and Robotics*. Springer, pp. 217–228.

Pini, Giovanni, Arne Brutschy, Alexander Scheidler, Marco Dorigo, and Mauro Birattari (2014). "Task partitioning in a robot swarm: Object retrieval as a sequence of subtasks with direct object transfer". In: *Artificial life* 20.3, pp. 291–317.

Pohl, Klaus (2010). *Requirements engineering: fundamentals, principles, and techniques*. Springer Publishing Company, Incorporated.

Pugh, Jim and Alcherio Martinoli (2008). "Distributed adaptation in multi-robot search using particle swarm optimization". In: *From Animals to Animats 10*, pp. 393–402.

Radatz, Jane, Anne Geraci, and Freny Katki (1990). "IEEE standard glossary of software engineering terminology". In: *IEEE Std* 610121990.121990, p. 3.

Ratnieks, FLW and C Anderson (1999). "Task partitioning in insect societies". In: *Insectes sociaux* 46.2, pp. 95–108.

Reynolds, Craig W (1987). "Flocks, herds and schools: A distributed behavioral model". In: *ACM SIGGRAPH computer graphics* 21.4, pp. 25–34.

Roache, Patrick J (1998). *Verification and validation in computational science and engineering*. Vol. 895. Hermosa Albuquerque, NM.

Rohmer, Eric, Surya PN Singh, and Marc Freese (2013). "V-REP: A versatile and scalable robot simulation framework". In: *Intelligent Robots* *and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, pp. 1321–1326.

Rouff, Christopher (2002). "A test agent for testing agents and their communities". In: *Aerospace Conference Proceedings, 2002. IEEE*. Vol. 5. IEEE, pp. 5–2638.

Şahin, Erol (2004). "Swarm robotics: From sources of inspiration to domains of application". In: *International workshop on swarm robotics*. Springer, pp. 10–20.

Şahin, Erol and Alan Winfield (2008). "Special issue on swarm robotics". In: *Swarm Intelligence* 2.2, pp. 69–72.

Sànchez, Nicolàs D Griffiths, Patricia A Vargas, and Micael S Couceiro (2018). "A Darwinian Swarm Robotics Strategy Applied to Underwater Exploration". In: *2018 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, pp. 1–6.

Schaffer, J David, Richard A Caruana, Larry J Eshelman, and Rajarshi Das (1989). "A study of control parameters affecting online performance of genetic algorithms for function optimization". In: *Proceedings of the third international conference on Genetic algorithms*. Morgan Kaufmann Publishers Inc., pp. 51–60.

Schneider-Fontán, Miguel and Maja J Mataric (1996). "A study of territoriality: The role of critical mass in adaptive task division". In: *From animals to animats IV*. Citeseer.

Seo, Heui-Seok, Tadashi Araragi, and Yong Kwon (2004). "Modeling and testing agent systems based on statecharts". In: *Applying Formal Methods: Testing, Performance, and M/E-Commerce*, pp. 308–321.

Shannon, Claude Elwood (2001). "A mathematical theory of communication". In: *ACM SIGMOBILE Mobile Computing and Communications Review* 5.1, pp. 3–55.

Sharkey, Amanda JC (2007). "Swarm robotics and minimalism". In: *Connection Science* 19.3, pp. 245–260.

Sharpe, Titus and Barbara Webb (1998). "Simulated and situated models of chemical trail following in ants". In: *Proc. 5th Int. Conf. Simulation of Adaptive Behavior*, pp. 195–204.

Shell, Dylan A and Maja J Mataric (2006). "On foraging strategies for large-scale multi-robot systems". In: *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. IEEE, pp. 2717–2723.

Siciliano, Bruno and Oussama Khatib (2016). *Springer handbook of robotics*. Springer.

Soysal, Onur and Erol Sahin (2005). "Probabilistic aggregation strategies in swarm robotic systems". In: *Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE*. IEEE, pp. 325–332.

Spears, William M, Diana F Spears, Jerry C Hamann, and Rodney Heil (2004). "Distributed, physics-based control of swarms of vehicles". In: *Autonomous Robots* 17.2, pp. 137–162.

Srivastava, Praveen Ranjan and Tai-hoon Kim (2009). "Application of genetic algorithm in software testing". In: *International Journal of software Engineering and its Applications* 3.4, pp. 87–96.

Stranieri, Alessandro, Eliseo Ferrante, Ali Emre Turgut, Vito Trianni, Carlo Pinciroli, Mauro Birattari, and Marco Dorigo (2011). "Self-organized flocking with an heterogeneous mobile robot swarm." In: *ECAL*, pp. 789–796.

Svennebring, Jonas and Sven Koenig (2004). "Building terrain-covering ant robots: A feasibility study". In: *Autonomous Robots* 16.3, pp. 313–332.

Trianni, Vito (2008). *Evolutionary swarm robotics: evolving self-organising behaviours in groups of autonomous robots*. Vol. 108. Springer.

Turgut, Ali E, Hande Çelikkanat, Fatih Gökçe, and Erol Şahin (2008). "Self-organized flocking with a mobile robot swarm". In: *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*. International Foundation for Autonomous Agents and Multiagent Systems, pp. 39–46.

Varela, Francisco J and Paul Bourgine (1992). *Toward a practice of autonomous systems: Proceedings of the First European Conference on Artificial Life*. MIT press.

Vargas, Patricia A, Ezequiel A Di Paolo, Inman Harvey, and Phil Husbands (2014). *The horizons of evolutionary robotics*. MIT Press.

Vargha, András and Harold D Delaney (2000). "A critique and improvement of the CL common language effect size statistics of McGraw and Wong". In: *Journal of Educational and Behavioral Statistics* 25.2, pp. 101–132.

Vaughan, Adam Lein1 Richard T (2008). "Adaptive multi-robot bucket brigade foraging". In: *Artificial Life* 11, p. 337.

Vicsek, Tamás, András Czirók, Eshel Ben-Jacob, Inon Cohen, and Ofer Shochet (1995). "Novel type of phase transition in a system of self-driven particles". In: *Physical review letters* 75.6, p. 1226.

Vincent, Patrick and Izhak Rubin (2004). "A framework and analysis for cooperative search using UAV swarms". In: *Proceedings of the 2004 ACM symposium on Applied computing*. ACM, pp. 79–86.

Watson, Richard A, Sevan G Ficici, and Jordan B Pollack (2002). “Embodied evolution: Distributing an evolutionary algorithm in a population of robots”. In: *Robotics and Autonomous Systems* 39.1, pp. 1–18.

Wegener, Joachim, André Baresel, and Harmen Sthamer (2001). “Evolutionary test environment for automatic structural testing”. In: *Information and Software Technology* 43.14, pp. 841–854.

Wei, Hao, Jonathan Ian Timmis, and Robert David Alexander (2017). “Evolving Test Environments to Identify Faults in Swarm Robotics Algorithms”. In: *IEEE Congress on Evolutionary Computation 2017*. York.

Werger, Barry Brian and Maja J Mataric (1996). “Robotic" food" chains: Externalization of state and program for minimal-agent foraging”. In: *Maes et al*. Citeseer.

Whitley, Darrell (1994). “A genetic algorithm tutorial”. In: *Statistics and computing* 4.2, pp. 65–85.

Winfield, Alan FT (2009). “Foraging robots”. In: Springer.

Winfield, Alan FT, CF Harper, and Julien Nembrini (2006). “Towards the application of swarm intelligence in safety critical systems”. In: IET.

Winfield, Alan FT, Christopher J Harper, and Julien Nembrini (2004). “Towards dependable swarms and a new discipline of swarm engineering”. In: *International Workshop on Swarm Robotics*. Springer, pp. 126–142.

Winfield, Alan FT and Julien Nembrini (2006). “Safety in numbers: fault-tolerance in robot swarms”. In: *International Journal of Modelling, Identification and Control* 1.1, pp. 30–37.

Winfield, Alan FT, Wenguo Liu, Julien Nembrini, and Alcherio Martinoli (2008). “Modelling a wireless connected swarm of mobile robots”. In: *Swarm Intelligence* 2.2, pp. 241–266.

Wooldridge, Michael and Nicholas R Jennings (1994). “Agent theories, architectures, and languages: a survey”. In: *International Workshop on Agent Theories, Architectures, and Languages*. Springer, pp. 1–39.

Xiong, Naixue, Jing He, Yan Yang, Yanxiang He, Tai-hoon Kim, and Chuan Lin (2010). “A survey on decentralized flocking schemes for a set of autonomous mobile robots”. In: *JCM* 5.1, pp. 31–38.

Zhang, Zhiyong, John Thangarajah, and Lin Padgham (2007). “Automated Unit Testing for Agent Systems.” In: *ENASE* 7, pp. 10–18.

Zheng, Mao and Vangalur S Alagar (2005). “Conformance testing of BDI properties in agent-based software”. In: *Software Engineering Conference, 2005. APSEC’05. 12th Asia-Pacific*. IEEE, 8–pp.