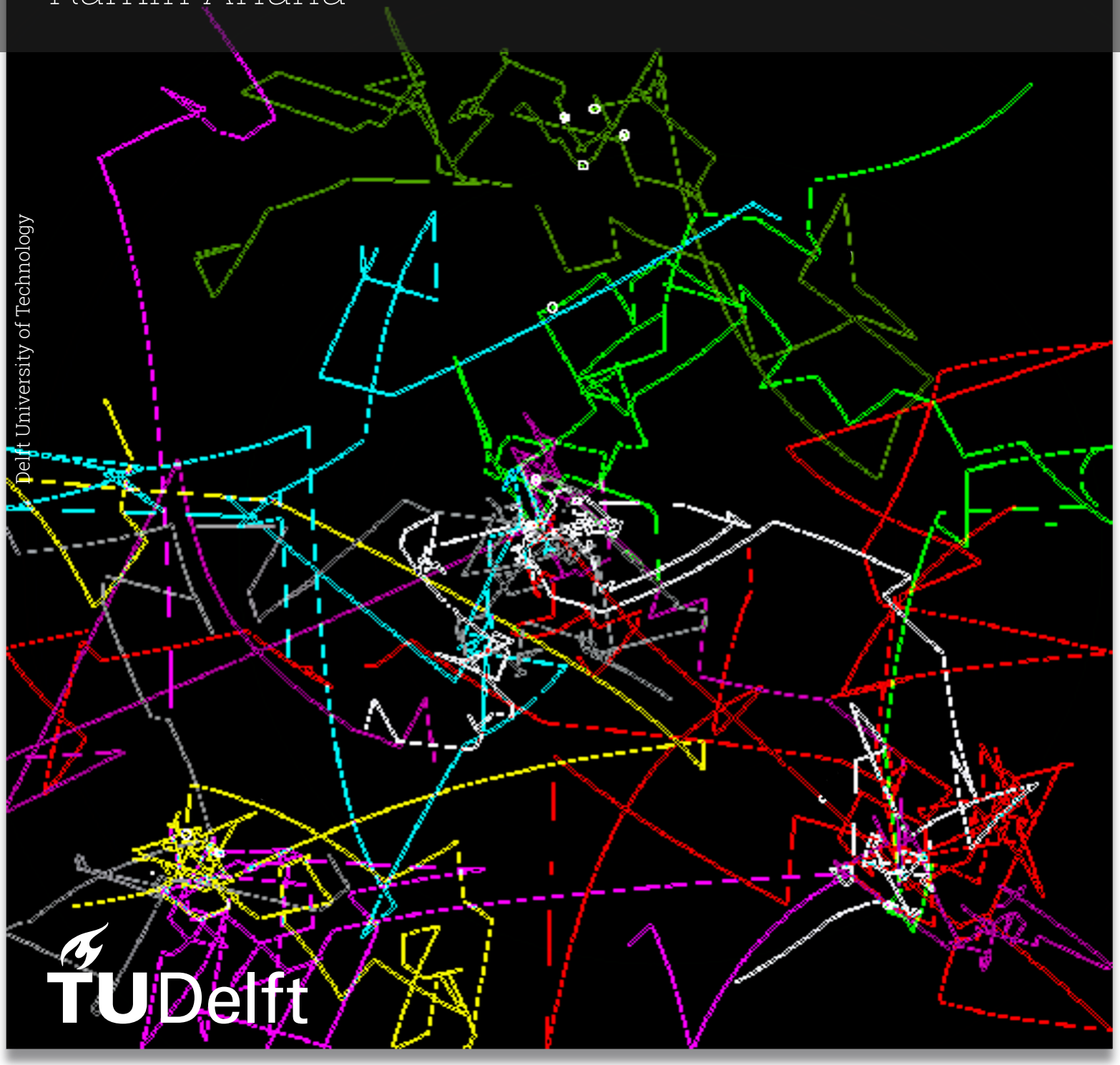
Master Thesis

Clustered Multi-Target Search in Unknown Large Environments Using Modified Bee Swarm Optimization

July 2023

Ramin Ariana



Delft University of Technology

TU Delft

Master Thesis

Clustered Multi-Target Search in Unknown Large Environments Using Modified Bee Swarm Optimization

by

Ramin Ariana

Student Name	Student Number
Ramin Ariana	5181585

Instructor:	Prof. dr. ir. J. Hellendoorn
Project Duration:	Feb 2023 - July 2023
Faculty:	Faculty of Mechanical Engineering, Delft

TU Delft

Preface

The Swan Song

A swan song is the final performance of an actor, poet, composer, or in this case, an engineer. Swans sing most beautifully before they pass, hence this phrase came to be used to describe someone who is leaving in style. Swans stand for grace and beauty. Aphrodite, the goddess of love, even found swans were sacred. In the same way, this is the final chapter of a long journey. A beautiful, intense, and above all memorable journey.

My journey at the TU Delft began with a Bachelor of Applied Science, resulting in a lot of catching up to do. The insecurities surrounding failing classes and asking for help during assignments were hard to overcome. Over time, I learned these are the landmarks of what makes a good engineer. Ask what you don't know and question what you do know. The only way to learn is to be curious. Doubt everything, except your own drive for success. Lastly, during my studies, I had to overcome some serious sickness. I have learned that your health, and the health of your dear ones, is the most important thing that exists. This is the one and only performance measure of leading a fruitful life. I am talking about physical health, but as much about mental health. These things are taken for granted until they are taken away from you. Cherish it, nourish it, and don't let it slip away.

Acknowledgements

I would like to thank my supervisor, Hans, for the subtle but efficient guidance throughout the thesis. The trust that you have had in me is something I acknowledge and appreciate. I would also like to thank Chris, as you sparked my interest in the topic of swarm robotics in the first place. The way you talk about swarming opens up endless possibilities in the minds that dare to dream.

Last but not least, I would like to thank my family for always guiding and supporting me. Especially my mother, who had driven me to Delft countless times when I needed to be there. The times I went to your home to recharge during the thesis were also invaluable to me. A special thanks to Loran, you were my partner in crime during the premaster. We studied together, which eased the distress. Lastly, I want to thank Koen and Korneel, who would always be available to discuss any topic I would throw against them. This was an individual journey, but you guys have helped me through it more than you think. Thank you for that.

Ramin Ariana
Delft, July 2023

Abstract

Swarm robotics (SR) is an emerging field of research that utilizes a swarm of robots that work together as a team to solve complex problems. One such challenge in SR is the exploration of unknown environments to find multiple targets. This problem finds applications in various domains, such as archaeology, underground exploration, signal source localization, and more. In line with this, the objective of this thesis is to develop a scalable SR algorithm for efficiently finding clustered targets in large unknown environments. Similar to how artefacts in archaeology are often found clustered around old settlements in large areas.

Inspired by Bee Swarm Optimization (BSO), the proposed algorithm leverages the strengths of BSO in balancing exploration and exploitation. However, modifications are made to adapt the proposed algorithm to incorporate limited communication range scenarios, common in large-scale environments. Also, the exploration-exploitation properties of BSO are redesigned. The proposed algorithm is named Modified Bee Swarm Optimization (MBSO). Here, robots assume different roles (scout, onlooker, experienced forager), similar to BSO, to optimize search and exploitation tasks. To address the issue of limited communication range, the robots establish an ad hoc network, truncating target information throughout the swarm. Additionally, an Artificial Potential Field (APF) is introduced to guide robots towards targets, and away from readily travelled clusters. To further aid the balancing of exploration and exploitation, a swarming architecture is introduced. This architecture is called the Architecture Multi-robot systems heterogeneous robots with Emergent Behaviour (AMEB) and aids with the decision-making of individual robots. The AMEB architecture is used to determine whether scouts should become onlookers, and the speed at which experienced foragers change back to scouts while considering real-robot physical limitations and individual performance levels. This architecture facilitates the continued advancement of the algorithm by allowing for the integration of additional sensory inputs, which in turn influences individual decision-making and, consequently, the emergence of the swarm. Lastly, cluster recognition is added to the algorithm, resulting in robots not transferring to readily travelled areas.

The characteristics of MBSO are evaluated. The target finding performance is benchmarked against a generic random walk method that stems from Lévy walking. Furthermore, this study investigates the effects of scaling the algorithm with an increasing number of robots and varying specific control parameters of MBSO on various aspects such as performance, redundancy, scalability, stability, and robustness. The results of this research have implications beyond archaeology, as the algorithm can be applied to various multi-target search problems in large unknown environments, such as minefield detection. By adjusting the proposed input variables, the algorithm can be optimized for these different scenarios. The developed SR algorithm shows promise in efficiently finding and truncating target information, leveraging the short communication range of the individual robots.

Overall, this thesis presents a novel approach to autonomous multi-target searching, in cases where targets are spread out in clusters, using scalable robotic swarming algorithms. In a field of 0.36 ha with 34 targets spread out over 5 clusters, robots that transfer with a speed of $6.4\,\mathrm{km\,h^{-1}}$ and optimal parameters for MBSO, scaling from 10 to 15 robots leads to $7.74\,\%$ more targets being found. Scaling from 15 to 20 robots resulted in $13.22\,\%$ more found targets. In comparison with random walking, MBSO with 10 robots performed the best, with an increase of $60.5\,\%$ more targets found. Increasing the number of robots exponentially increases the redundancy of the algorithm, resulting in the same targets being found by different robots more often. With 10 robots, it was shown to be more effective to transfer farther for targets and stay longer at cluster locations. Alternatively, for 15 and 20 robots, transferring to closer potential target locations increased the overall target-finding performance.

Contents

List of Figures

List of Tables

1

Introduction

In 2013, researchers showed the feasibility of a multi-swarm constructed of bacterial microbots to detect breast cancer [11]. In other work [2], researchers showed that it is possible to use a swarm of low-cost Unmanned Aerial Vehicles (UAVs) to search a wide range of areas in a short time. The swarm would still be efficient, even when multiple UAVs in the swarm fail or crash. The swarm also requires minimal operator input, which frees rescue workers from other tasks. These two examples show some important aspects of Swarm Robotics (SR). Firstly, individuals of a swarm can be as small as cells, or as large as planes. A swarm is, therefore, not defined by its size. What matters is the behaviour between robots and the emergence of swarm behaviour based on individual interactions. Secondly, the capabilities of such systems can be the difference between life and death scenarios. Hence, more research in this field is needed.

SR is an area of research that aims to solve complex problems by using a group of robots that work together as a team. Similar to animals and insects, swarm robots are characterised by their limited capabilities, little knowledge of the environment and lack of knowledge of target locations. These mobile robots are valuable in environments that are too dangerous, or inaccessible for humans. The reason for implementing multiple robots instead of one robot is that they are able to complete a task much faster than one single mobile robot, by actuating in several places in parallel [19]. One of the many challenging problems SR tries to solve is finding multiple targets in an unknown environment. This problem has various applications, one can think of exploration and searching for certain rocks in extraterrestrial domains like the moon, finding underground targets like gas pockets, an archaeologist looking for buried artefacts, the detection of mines [9, 37], signal source localisation [13, 24, 42, 75], finding caves on mars [33], leak detection [4], and more.

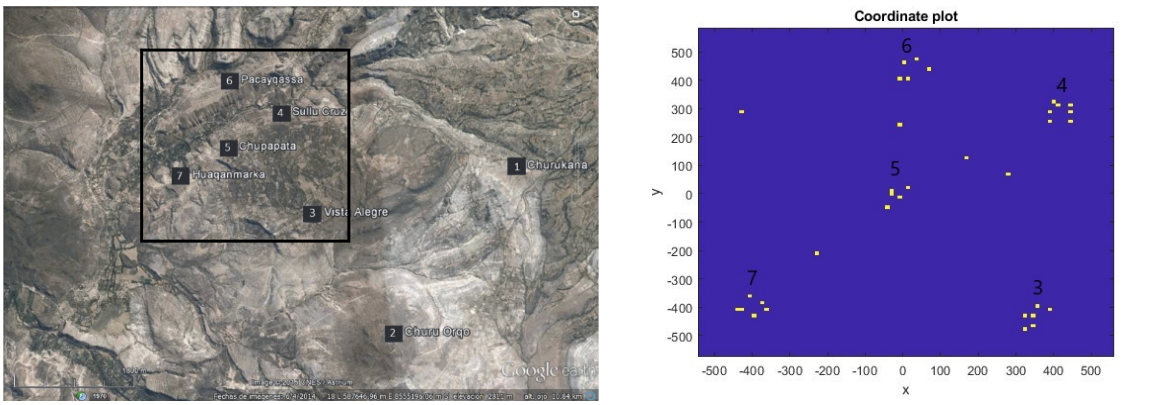
Swarming algorithms are generally used for (un)constrained optimization, multi-solution problems, multi-objective problems and dynamic optimization problems. These optimization problems can not always be directly translated into SR, as there are many limitations and constraints when transferring such optimization algorithms to robots. For example, in large-scale robotics applications, the necessary information for full-range swarm decision-making is often lacking. SR in the case of multi-target finding can be defined by that SR are a group of mobile robots that are using their sensory perception, while interacting with their neighbours, to navigate towards multiple target points or signals. The use of swarming algorithms in SR for multi-target search problems has been studied in the literature. Particle swarm optimization is one of the popular algorithms, as it is an effective tool for solving multi-target search problems [75]. It is inspired by the flocking of birds. However, other algorithms, such as ant colony optimization [43, 48, 54] and bee swarm optimization [1, 45] have also been proposed in the literature. These algorithms mostly work by optimising the search process by using a fitness function that drives the robots towards (local and global) optima. Communication range and other real-robot constraints are often not considered.

In this thesis, a robotic swarming algorithm has been developed to search targets that are scattered in a clustered fashion, e.g., in settlements, and with limited communication range. The problem arises from archaeology, where vast areas need to be explored, and targets are often gathered in settlements. By employing a swarm of mobile robots, this study aims to alleviate archaeologists from the laborious field survey phase by autonomously identifying and mapping artefacts in a large, unknown area, where local decentralized communication must be formed. The proposed algorithm guides the swarm towards areas of interest, allowing archaeologists to focus their efforts on these specific locations. Although the problem is phrased for archaeology, the proposed algorithm's benefit extends to diverse scenarios, such as minefield detection. A modified bee swarm optimization algorithm is introduced. The proposed algorithm minds the individual robots' limited communication range. In addition, exploration and exploitation are balanced by robots reasoning about environmental variables and individual performance levels.

1.1. Location

Our world remains largely unexplored, with increasing discoveries of traces from ancient civilizations, revealing glimpses of the past. The job of an archaeologist is to study past people and cultures by excavating and examining remains. However, not all areas are easily explored. For example, scholars believe the site of Rakigarhi in India ranges from 80 ha to 100 ha [53] to even 550 ha [46]. Only 5 % of the site has been discovered [67].

A swarm system is defined by its scalability, and it seems reasonable to make size simplifications to the simulated area to increase simulation speed. This has been realized by mimicking the placement of villages 3-7. The location of the originally 3750 m by 4810 m sized area visualized in Fig. 1.1a has been scaled down to a more approachable 1200 m by 1200 m field visualized in Fig. 1.1b. Also, the area is simulated as flat, and robots are assumed to be able to transverse throughout the space without obstacles like mountains or rivers being in their way. In the real world, this would form an entire problem in itself. For the purpose of this study, however, the importance of correctly simulating the environment is not high. Rather, the algorithm's relative and generalized performance in terms of cluster searching behaviour, and the balancing of exploration and exploitation is of importance. For the test environment, a total of 34 targets are spread out in the area. Village 3 has 6 targets, village 4 has 7 targets, village 6 has 5 targets, village 5 has 5 targets, and village 7 has 6 targets. To be able to test the robustness of the algorithm, 5 targets that do not adhere to any cluster are also introduced. Targets that do belong to a cluster are distanced from each other around 60 m. This simulated area is visualised in Fig. 1.1b.



(a) Map of settlements in Huari [64].

(b) Coordinate plot of simulated Huari of size 1200 m by 1200 m.

Figure 1.1: Real and simulated coordinate plot map of Huari.

1.2. Research goal and question

Swarming algorithms that are used in robotics, e.g. Swarm Robotics (SR), are usually designed to function in information-rich environments, as they function as optimisation algorithms. In a large and rural environment, however, it is not always possible to acquire the necessary information. This information deficiency in combination with the limited communication range between robots can drastically decrease the performance of these algorithms. The objective of this study is to develop a scalable SR algorithm capable of efficiently locating clustered targets in large-scale environments using a swarm of mobile robots operating through ad hoc networks with limited communication capabilities. The performance parameters of the proposed algorithm are analyzed such that others can determine efficient control parameters for other types of study. The research question becomes:

Research Question

How can an autonomous, scalable robotic swarming algorithm be developed to efficiently locate and map targets in an unknown environment with clustered target spread, leveraging limited communication capabilities, thereby alleviating archaeologists from the initial field survey phase?

By addressing this research question, novel insights will be gained into the development of SR algorithms for large-scale environments, with potential applications beyond archaeology. The study attempts to contribute to the advancement of SR technology and its practical implementation in multi-target search problems, ultimately enhancing the efficiency and effectiveness of exploration tasks in challenging and vast environments. In addition, some sub-questions are formed. These questions are based on literature findings, elaborated in Sect. 2, and are as follows:

Sub-questions

1. Do the physical range limitations improve the performance of the robots, due to robots being forced to explore?
2. Does the performance of the algorithm scale linearly when adding more robots?
3. Does migration between sub-swarms increase optimality?

1.3. Thesis outline

First, a literature study about Swarm Robotics (SR) is done in Ch. 2. To show the wide variety of algorithms that exist within swarming, a few examples are provided. In addition to these theoretical frameworks, some real-life applications of SR have also been covered in Sect. 2.2. Ch. 3 provides the proposed swarming algorithm. The origin of this new Modified Bee Swarm Optimization (MBSO) algorithm (Sect. 3.2) originates from bee swarm optimization, which is introduced first in Sect. 3.1. After introducing the basic foundation of MBSO, the ad hoc communication method between robots is elaborated on in Sect. 3.3. To help balance exploration and exploitation of the algorithm, Architecture Multi-robot systems heterogeneous robots with Emergent Behaviour (AMEB) is introduced in Sect. 3.4. AMEB has been applied to MBSO in Sect. 3.4.2. Then, an artificial potential field is introduced to help guide the robots towards targets and away from already visited areas. This can be found in Sect. 3.5. Next, the fundamentals of the global random walking method used in MBSO are explained in Sec. 3.6.2. For locally searching potential areas, a local walk method has also been introduced. This local walking method is named Dynamic Lévy Walk (DLW). More about DLW can be found in Sect. 3.7. Finally, the performance of the algorithm is measured by varying control variables of the algorithm, and measuring the response of the overall system. The methods used in this work are introduced in Ch. 4. Here, the benchmark (Sect. 4.6) of the algorithm is also given, which is solely based on the global random walk method introduced earlier. Finally, the results section is presented in Ch. 5. Lastly, the discussion (Ch. 6) and conclusion are given (Ch. 7). The appendix consists of independent research done to measure the target finding performance of MBSO with limited communication range, and limited target finding range Appendix A. Also, the controller code of MBSO is provided in Appendix B. Lastly, the full literature review has been added in Appendix C.

2

Literature Review

Swarm Robotics (SR) is not a new field, however, the definition of SR is still quite vague. Terms like collective robotics, distributed robotics, and robot colonies often overlap [56]. In this chapter, the definition of SR is clarified. This has been done by first differentiating between SR and other similar systems like multi-robot/multi-agent systems, sensor networks and SR. Also, the characteristics of an SR are briefly described. Then, multiple examples of SR are given. Particle swarm optimization, ant colony optimization, glowworm swarm optimization, fruit fly optimization, the bat algorithm, group explosion strategy and bee swarm optimization are examined if these algorithms seem fit to be used in a task where clustered targets must be sought out by the swarm. Then, some notable applications of SR at the current time of writing are shown.

2.1. Introduction of swarming algorithms

As a starting point, swarming is defined as: ”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 agents and between the agents and the environment [56]”. The main characteristics of SR are defined as [5]:

1. robots are autonomous;
2. robots are situated in the environment and can act to modify it;
3. robots sensing and communication capabilities are local;
4. robots do not have access to centralized control and/or to global knowledge;
5. robots cooperate to tackle a given task.

Similar frameworks are defined by others, adding that the swarm must be homogeneous or heterogeneous, that is, composed of a number of homogeneous groups [56]. An overview of the differences between SR and other similar systems is given in table 2.1.

	Swarm robotics	**Multi-robot system**	**Sensor network**	**Multi-agent system**
Population size	Variation in great range	Small	Fixed	In a small range
Control	Decentralized and autonomous	Centralized or remote	Centralized or remote	Centralized or hierarchical or network
Homogeneity	Homogeneous	Usually heterogeneous	Homogeneous	Homogeneous or heterogeneous
Flexibility	High	Low	Low	Medium
Scalability	High	Low	Medium	Medium
Environment	Unknown	Known or unknown	Known	Known
Motion	Yes	Yes	No	Rare
Typical Applications	Post-disaster relief Military application Dangerous application	Transportation Sensing Robot football	Surveillance Medical Care Environmental protection	Net resources management Distributed control

Table 2.1: Comparison of swarm robotics, multi-robot systems, sensor networks and m-agent systems [60].

SR is characterized by being: robust, flexible and scalable [3]. Robustness can be defined as the functionality of the system in the presence of failure of a part of the system, or other unexpected conditions. Flexibility can be defined as adaptability to changing requirements of the environment. Lastly, scalability can be defined as the ability to change the number of individuals in the system without drastically changing performance [3]. A lower bound for the group size of a swarm is accepted to be 10-20 agents [27, 56]. A swarm is not assumed to have an explicit leader.

The reader is invited to read the full literature review that has been attached in appendix C for a more comprehensive literature review. There, an examination of various swarming algorithms, robotic, swarming and communication architectures, obstacle and collision avoidance techniques and simulation methods is given. The literature review identified potential avenues for developing a scalable robotic swarming algorithm capable of conducting multi-target searches in unknown environments. It provides a strong foundation for the development of such an algorithm by considering the various approaches and techniques that have been proposed in the literature. This review is the backbone of this thesis. The most prominent swarming papers from the mentioned review have been visualized in Fig. 2.1. While some algorithms date back to 1987 (particle swarm optimization), others are relatively new (fruit fly optimization algorithm). The summary of this review is given in the following sections to point out the strengths and weaknesses of these different algorithms in different conditions, such that the choice of using bee swarm optimization for searching for clustered targets becomes apparent to the reader.

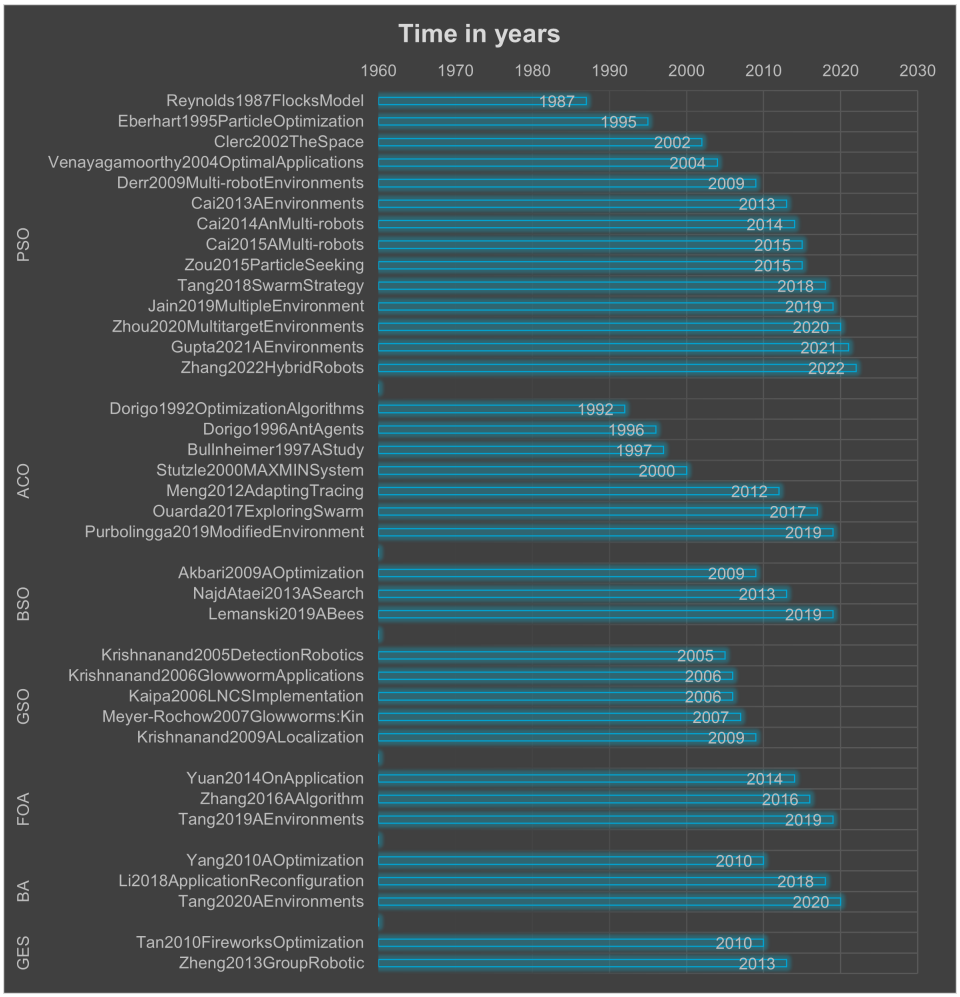


Figure 2.1: Most prominent studied robotic swarming algorithms, visualized against time.

2.1.1. Particle swarm optimization

Particle Swarm Optimization (PSO) is a widely used, simple and inexpensive algorithm for multi-target robotic swarming [6–8, 13, 18, 23, 63, 72, 75, 76]. However, to find multiple targets, the swarm must be divided into multiple sub-swarms. This is required because the original algorithm is not suitable for finding multiple targets. Because of its tenacity to optimize towards one particular solution, it is also prone to getting stuck in local optima. PSO guides all robots within the sub-swarm at the same time towards the target with the highest fitness within the group, by using a social component. This requires continuous communication. Robots in PSO share their personal best location with other swarm members. This value is compared within the sub-swarm and used to determine the best next position. PSO in Swarm Robotics (SR) should be used if robots are expected to move like a flock of birds, relatively close to each other, stochastically moving away from each other to then join each other again. An example could be the use of a swarm of unmanned areal vehicles to scan an area. PSO is thus not fit to be used in large areas where communication is limited, and targets are scattered, which is the case in this study.

2.1.2. Ant colony optimization

In Ant Colony Optimization (ACO), members continuously deposit pheromones while moving. These pheromones evaporate over time, and during this deposition and evaporation, prevalent pathways are strengthened. Traditional ACO is perfectly adequate to solve stochastic, discrete combinatorial optimization problems [14], like the travelling salesman problem. ACO in robotics is often applied in combination with other algorithms [43, 48, 54]. ACO requires pheromone concentrations of discretized nodes or 'locations', to be saved and iterated upon. This calls for centralized hierarchy, or at least full-swarm global updates of these pheromone concentrations for the algorithm to work optimally. Anti-Pheromone ACO [54], however, can be promising for SR. Here, robots are attracted to less visited notes, by inverting the consequence of pheromones. This way exploration is increased, and the dependency on these updates is reduced. Robots could use this Anti-Pheromone method to make sure they don't visit the same place twice or don't visit an already visited place by another robot. A better way to use ACO in Sr is as a communication method between robots in the form of an ad hoc communication method. Its strength lies in best-path exploitation, it could therefore find a better place in path optimization. As mentioned before, ACO also needs global updating of pheromone positions, making it unfit in decentralized environments.

2.1.3. Glowworm swarm optimization

Glowworm Swarm Optimization (GSO) is a truly simple algorithm basing robotic movement solely on one factor; luciferin value or 'glow'. GSO can be used for multi-robot searching, localisation and rendezvousing in environments where robots have a limited communication range [26, 34–36]. Unfortunately, GSO does not have good exploration properties, as agents do not explore. They can, therefore, easily get stuck in local optima. To use this algorithm to find multiple targets, robots must be spread out well initially. GSO is computationally inexpensive. This algorithm would come to good use in environments where next to no communication is available, and the robot can sense their neighbours. An example could be in caves, where robots use light to communicate findings. If there are enough robots spread out, the algorithm could suffice in finding multiple targets. However, in more complex, and larger environments, the algorithm would not be sufficient. Therefore, GSO is not deemed fit for the purpose of this study.

2.1.4. Fruit fly optimization algorithm

Fruit Fly Optimization Algorithm (FOA) is an all-or-nothing algorithm, directing all agents of the swarm towards the best position available at that moment. FOA is not naturally fit to find multiple targets. Multi-swarm robotic FOA exists [71, 73], however, using it in a hybrid form can improve on the limitations of just using FOA [61]. Instead of particle swarm optimization, where all particles are continuously influencing each other's motion by balancing their personal goal and global goal, the best global position within the (sub-)swarm is shared with all agents. This implies the algorithm requires long-range communication to function fully. Flies individually 'fly' towards this optimal target using 'sight'. This translates into them individually moving towards the truncated target. Additionally, the limited search space and swarm diversity weaken the global search ability. Unfortunately, this also makes it prone to getting stuck in local optima. The advantages are that it is an easy and

computationally inexpensive algorithm. FOA could be great for quick exploitation, as all flies fly towards the best position. Say there are only two targets in the search space, two sub-swarms can be created, and both of them can be directed to find one target. All robots in the corresponding sub-swarm are controlled to find their target, without having individual goals. The need for global updating, in combination with its weak exploration abilities, make FOA a bad fit for cluster searching in large environments.

2.1.5. Bat algorithm

The Bat Algorithm (BA) shows potential for solving optimization problems [41, 69]. However, only one use case for multi-target search in unknown environments, using swarm robotics, has been found [62]. To find multiple targets, the swarm must be divided into sub-swarms, similar to Particle Swarm Optimization (PSO). BA is very similar to PSO, the differences being that (i) exploration ”loudness” goes down in time, and the measuring pulse emissions go up if a target is approached (ii). Also, an additional frequency variable influences the speed update equation (randomly or by Doppler effect). Using varying measurement frequencies could be beneficial for some specific environments, like trying to find the deepest point of a cave. However, in the current study, it does not have any beneficial properties and all the negatives of PSO.

2.1.6. Group explosion strategy

Group Explosion Strategy (GES) can be interesting for robotic multi-target search [74]. The chaotic nature of the algorithm helps the initial exploration phases. Robots ’explode’ in a ring formation around the robot with the highest fitness. With enough robots in the field, these explosions can continuously happen, increasing exploration by exploding, and exploitation around interesting areas by the centre of the group moving towards the robots with the highest fitness. The fact that this algorithm doesn’t need synchronization, and is simple and easy to implement, can be interesting for swarm robotics. The exploration properties however are chaotic, and hard to control. An example of where this algorithm would be the algorithm of choice is when multiple targets are scattered around each other. For example, finding parts of a crashed aeroplane. Robots will converge towards this target, and around it, searching for nearby targets in a circular formation, and continuously exploding around each other. GES is hard to control and due to its chaotic nature unusable in large areas, it would be more useful to use as a local search method instead of a full swarming algorithm.

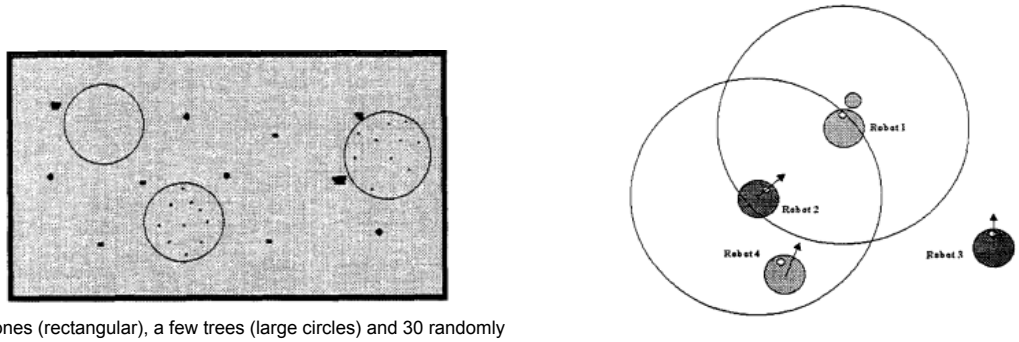
2.1.7. Bee swarm optimization

Bee Swarm Optimization (BSO) has been proposed for numerical function optimization [1], multi-target search, coverage in unknown areas [25] and multi-robot and multi-target search in an unknown environment [45]. The main advantage of BSO compared to particle swarm optimization is that in BSO, agents are categorized in three patterns of searching [45]. This automatically balances exploration and exploitation. It is therefore expected that using BSO provides better results in finding targets due to the balance of exploration and exploitation. BSO is a 3-state algorithm that uses intensive communication to balance exploration and exploitation, by sharing exploration and exploitation tasks between different kinds of agents. Scouts are tasked with exploring, foragers are exploiting, and onlookers help exploit interesting areas. BSO is somewhat similar to particle swarm optimisation in terms of the movement of single agents but has some overhanging framework to determine social behaviour. Initial placement of the robots should not influence the swarm as much, as exploration and exploitation are meant to be well balanced. This suggests this algorithm could be a good fit for searching multiple clustered targets. As exploitation at these clusters is as important as exploring between the clusters, the algorithm should be able to balance both. It seems that BSO, with some modifications to incorporate decentralized communication, should be a good fit to solve the main question asked in this thesis.

2.2. Application of swarm robotics

Domains of application associated with tasks that require finding targets in a region, covering a region that is too dangerous to visit by humans, tasks where the environment is large, where there is a need to scale-up or scale-down [56] are interesting topics for Swarm Robotics (SR). The proposed archaeological problem, however, does not exist in the current literature. Therefore, some applications of SR in similar domains as the one described in archaeology are given next. Behaviour cases that comply with collective exploration (as defined by [5]), collective localization, foraging, self-deployment and other similar techniques are also interesting subjects.

Cassinis, Bianco, Cavagnini et al. (1999) [9] provided a movement strategy based on vectorial combinations between robots. They simulated, defined and compared different movement methods and corresponding target-finding performances. The comparison was made between random movement, relay clustering, flocking, swarming, formation maintenance and comb movement during the search task. The goal was to find mines in a minefield, some trees and stones were also scattered in the area as seen in Fig. 2.2a. Relay clustering is when a robot that has found a target relays that information to neighbouring robots, which then move towards that robot. When a robot is out of range, it will keep randomly searching the area. This study shows that this physical range limitation can actually improve performance in the sense that there is more exploration, or 'wandering', by unknown robots.



(a) Stones (rectangular), a few trees (large circles) and 30 randomly placed mines of three different kinds (small circles). the mines of the same kind are gathered together in the field. This situation is very likely to occur in real minefields [9].

(b) Relay Clustering: in the first phase, Robot 2 and Robot 4 go towards Robot I, which has found a mine. Robot 3 still moves at random because it is out of the range of influence of Robot 1 [9].

Figure 2.2: Mine Searching task in swarm robotics [9].

Penders, Cervera, Witkowsi et al. (2007) [50] focuses on making a swarm of autonomous robots called "GUARDIANS" that can be applied for navigation and search in industrial warehouses that are filled with smoke. A time-consuming and dangerous job for humans. The research is still at its beginning, but it tries to realise that the robots provide and maintain mobile communication links and localization information can be inferred, enabling the detection and warning of toxic chemicals and assistance in the search for people. In their research, they use Adept's Pioneer 3-DX robot. This kind of robot is also used in the current research.

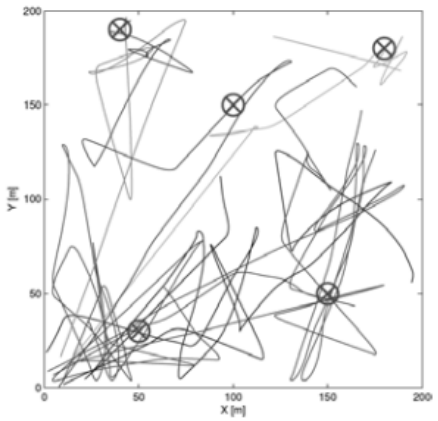


Figure 2.3: Search path of swarm using an electronic nose [50].

Duarte, Coste, Gomes et al. (2016) [15] explored Aquatic environmental monitoring of a large environment of size $10\,000\,\mathrm{m}^2$. In this paper, researchers tried to solve this search problem by training a controller that was scored based on the amount of area covered per robot. Scalability was tested by starting with eight robots, removing 4 robots after 5 minutes, and adding two robots after 10 minutes. In their study, the performance of the controller scaled linearly with the number of robots used.

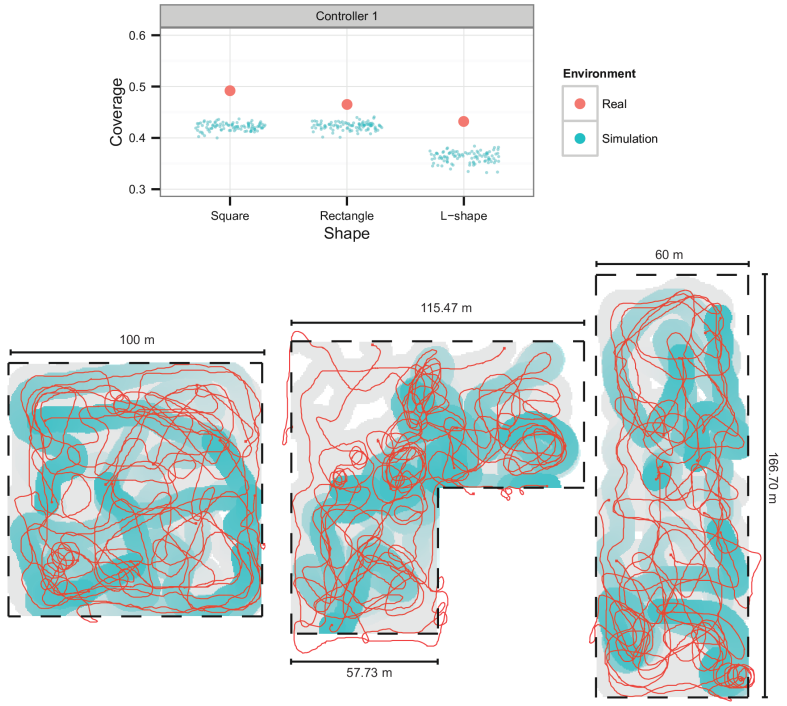


Figure 2.4: Coverage of different shaped maps with a real swarm presented in blue with a decay of $100\,\mathrm{s}$. Trajectories for the full duration of the task are presented in red, and all the areas visited by the robots are filled in grey [15].

Hayes, Martinoli, Goodman et al. (2003) [21] presented a swarm of odour localization robots that move using a spiral surge algorithm, as visualized in Fig. 2.5. They found that solving this odour localization task in a group outperforms the performance of a single robot. They also found that when completion time becomes more valued over total energy used, large group sizes become optimal. Also, larger movement parameter values can result in more efficient search performance for larger group sizes using their proposed algorithm.

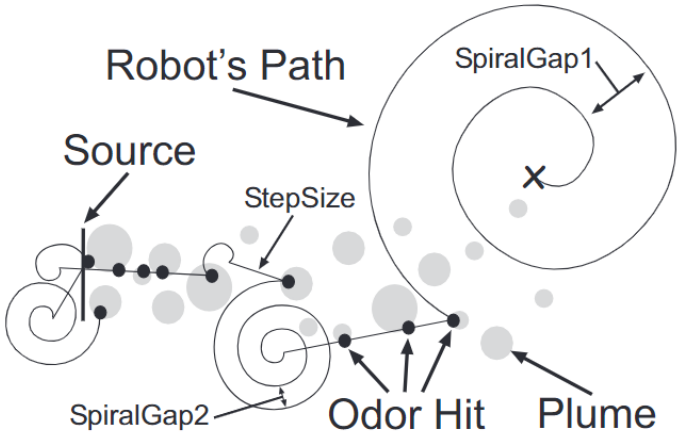


Figure 2.5: Swarm odor localization using the spiral surge algorithm in swarm robotics [21].

Zhou, Chen, Zhang et al. (2020) [75] issued to solve the multi-target search problem in an unknown complex environment by putting the emphasis on task allocation. After sub-swarms were created, individuals would sort their target type, target response intensity and communication distance to enable lower-level members to withdraw from the sub-swarm and participate in the search with other sub-swarm. This enables robots to migrate between sub-swarm. They set an upper limit to the maximum size of a sub-swarm.

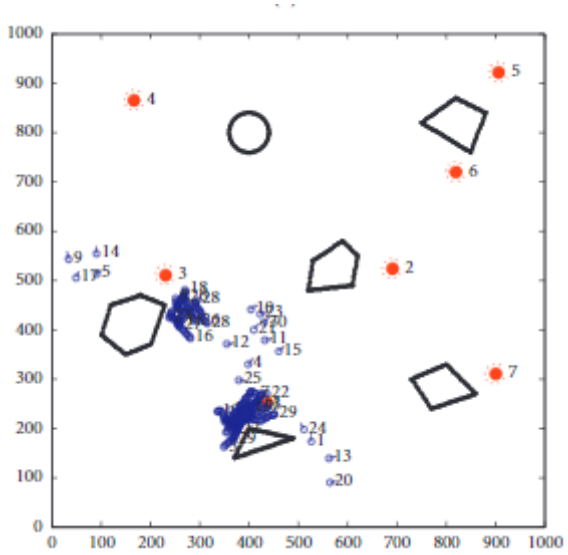


Figure 2.6: Swarm robots for multi-target search in unknown environments with convex obstacles [75].

2.3. Concluding remarks

In conclusion, the field of Swarm Robotics (SR) research contains the study of large numbers (minimum of 10-20) of relatively simple agents that interact with each other and their environment to achieve desired collective behaviours. The characteristics of SR include autonomy, scalability, local sensing and communication, lack of centralized control or global knowledge, and cooperation among robots to accomplish complex tasks.

Various algorithms have been proposed for SR, each with its strengths and weaknesses. Particle Swarm Optimization (PSO) is a widely used algorithm but requires sub-swarm forming and continuous communication, making it more suitable for smaller areas. Ant colony optimization can be effective but relies on centralized or global updates of pheromone concentrations, limiting its use in decentralized communication scenarios. Glowworm swarm optimization is simple but lacks exploration properties, making it more suitable for smaller environments with limited communication and sensing capabilities. The fruit fly optimization algorithm offers quick exploitation but can get stuck in local optima and requires global updating. The bat algorithm shares similarities with PSO. The group explosion strategy is chaotic and therefore not useful as a global search algorithm, however, it could be useful if used as a local search method. Bee swarm optimization provides a good balance between exploration and exploitation by using three types of agents, where some are focused on exploration, and others on exploitation, making it promising for clustered multi-target search in large environments.

Some examinations from the literature are also considered for the proposed algorithm. Firstly, [9] suggests that the physical range limitations of robots can improve performance in such searching tasks, as wandering naturally improves exploration. Secondly, the performance of individual controllers in large environments scales linearly with the number of robots [15]. Lastly, the migration between sub-swarms can be optimal for the performance of the swarm [75]. The sub-questions that are formed from these observations from the literature are listed next:

1. Do the physical range limitations improve the performance of the robots, due to robots being forced to explore?
2. Does the performance of the algorithm scale linearly?
3. Does migration between sub-swarms increase optimality?

Overall, SR should offer robustness, flexibility, and scalability, making it a promising approach for various applications such as post-disaster relief, military operations, operating dangerous environments, and more. By understanding the strengths and weaknesses of different SR algorithms, researchers can select the most appropriate approach for their specific application and contribute to the advancement of this rapidly evolving field. In the case of large environments, scalability, and the lack of communication can potentially be used as leverage to increase exploration.

3

Proposed Swarming Algorithm

The proposed algorithm in this thesis is inspired by Bee Swarm Optimisation (BSO). BSO has good balancing properties between exploration and exploitation. BSO manages this by dividing exploration and exploitation tasks between three types of robots. Some are tasked with searching the entire area, whereas others exploit high-potential areas. The three roles are scout, onlooker and experienced forager. The original BSO algorithm, however, requires continuous communication capabilities to allow the algorithm to function fully. In the case of exploring an unknown, harsh, and especially large environment, this is not always feasible. Besides, BSO has been optimized for finding optima in use cases where comprehensive data is present, this information is used to guide the algorithm to an optimal solution. When an underwhelming amount of information is presented to the robots, the original BSO algorithm will not suffice.

To enable robots to reason in an information-lacking environment with limited communication range constraints, a few adjustments are necessary to be made to the original BSO algorithm presented in Sect. 3.1. First, the basic framework of the proposed Modified Bee Swarm Optimization (MBSO) algorithm is set in Sect. 3.2. The algorithm is split up into a scout (Sect. 3.2.2), onlooker (Sect. 3.2.3) and experienced forager (Sect. 3.2.4), similar to BSO. Communication is performed locally using an ad hoc network. This decentralized communication method used in MSBO is elaborated on in Sect. 3.3. Next, the balancing of exploration and exploitation is managed using the introduced swarming architecture, which is explained in Sect. 3.4. In Sect. 3.5, the goal-orientated movement of onlookers using an artificial potential field is defined. Then the global random walk method used by scouts in MBSO is elaborated on in Sect. 3.6. Experienced foragers use a local search method, which is explained in Sect. 3.7. Lastly, the algorithm is improved with cluster recognition in Sect. 3.8. The reason why this is added on lastly is that it is one of many ways to make sure robots don't transverse to already visited areas. Also, cluster recognition only kicks in after much time is passed. In this study, simulation times are relatively short, resulting in the performance of cluster recognition being minimal. The simulations in this thesis only are $2\,$h and $15\,$min long.

3.1. Bee swarm optimization

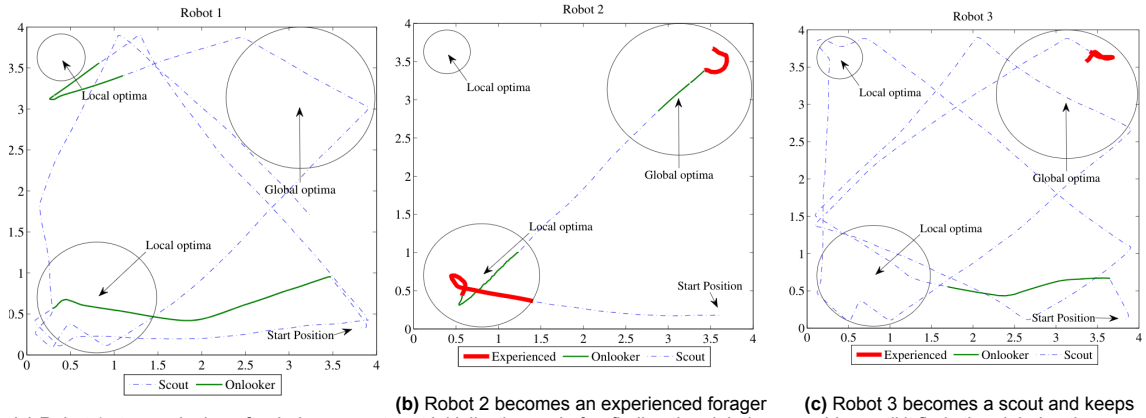
Optimization algorithms are often inspired by behaviour that has evolved from nature. One of the most recent algorithms is called BSO and is based on the foraging behaviour of honey bees. [45] showed, using a set of random benchmarks, that it outperformed PSO by 50.6%. They claim the following: "The main advantage of BSO compared to PSO is the fact that BSO categorizes agents into three different types with three different patterns of searching. We expect that having three different patterns in our search system provide better results in finding the target with best fitness due to balancing exploration and exploitation [45]". PSO is one of the most popular swarming algorithms that has been used for such multi-target robotic swarming purposes [6–8, 13, 18, 23, 63, 72, 75, 76]. In BSO, however, bees get assigned different tasks. The search space is explored by a scout bee. On the other hand, patterns are formed by exploiting potential areas by foragers and onlooker bees. This automatically introduces a good contrast between exploration and exploitation.

3.1.1. Biological foundation

Honey bees (*Apis mellifera*) are capable of performing complex tasks like collecting, processing, and advertising nectar. Also, tasks like brood care, hive construction and foraging are part of their task set. They do all this only using relatively simple decentralized rules. In foraging, learning plays an important role. Honey bees need to be able to identify suitable resources, return them repeatedly, and communicate the location of the quality of the resources [39].

3.1.2. Bee swarm optimization algorithm

There are three kinds of agents in the BSO algorithm, namely: scouts, onlookers and experienced foragers. Scouts are searching for new food sources, while experienced foragers and onlooker bees have the task of exploring the search space. Onlooker bees choose an experienced forager with high fitness, using a roulette wheel, receiving their position. This selected experienced forager with a higher fitness now becomes the new target location for this particular onlooker bee. Only when a scout finds a target with higher fitness than the experienced foragers, the scout becomes an experienced forager, and the other robots are again divided into scouts and onlookers. In Fig. 3.1, an example of how a bee algorithm performs is given. Here, bees change their type, based on their proximity to the target, and interactions with each other [45].



(a) Robot 1 stays onlooker after being a scout because it has not found the global optimum.

(b) Robot 2 becomes an experienced forager at initialization and after finding the global optimum

(c) Robot 3 becomes a scout and keeps searching until it finds the global optimum, and becomes an experienced forager.

Figure 3.1: Visualisation of the three types of robots and their trajectories in BSO by [45].

Basic BSO uses three different search patterns, which can help balance exploitation and exploration. In the BSO algorithm, bees are first initialised. Then, some fitness function, like the Euclidean distance between the bee and the target, or the intensity of the target signal, is calculated. A certain amount of bees with the poorest fitness values are selected as *scouts*. This selection makes sense, as these particular bees are not in the best place, and should keep exploring. The rest of the bees are halved, and the best half become *experienced foragers*, while the other bees are selected as *onlooker bees*. The scout bees i, random walk (RW) a region in dimension $j = 1.., n_x$, with radius τ and in time t. This can be seen in (3.1). Often, this radius decreases throughout iterations [1]. This helps with the exploitation of the search area.

$$x_{ij}(t+1) = x_{ij}(t) + RW(\tau, x_{ij}(t)) \tag{3.1}$$

The trajectory of the onlooker bee i is calculated by using (3.2). c_2 is an acceleration constant, and r_2 is a random variable in the range [0,1], similar to PSO. $elite_j(t)$ is the position of the chosen experienced forager (elite bee), by onlooker bee i, using a roulette wheel.

$$x_{ij}(t+1) = x_{ij}(t) + c_2 * r_2 * (elite_j(t) - x_{ij}(t)) \tag{3.2}$$

Lastly, the experienced foragers update their position by using (3.3). c_1, c_2 are acceleration constants, and r_1, r_2 are random variables in the range [0,1], similar to PSO. $pbest_{ij}$ is the best position of experienced forager i in dimension $j = 1.., n_x$. $gbest_j$ is the position found between all experienced

foragers. The relative importance and stochasticity of cognitive ($pbest_{ij}(t)$) and social ($gbest_j(t)$) components are determined by the acceleration constants and random variables defined before.

$$x_{ij}(t+1) = x_{ij}(t) + c_1 * r_1 * (pbest_{ij}(t) - x_{ij}(t)) + c_2 * r_2 * (gbest_j(t) - x_{ij}(t)) \tag{3.3}$$

It is clear that this is an optimization algorithm that needs a continuous fitness function to be evaluated to enable the swarm to find the global optimum. This is the case with many algorithms, however, the scout bees introduce some stochasticity in the search process, enabling the swarm to find targets more efficiently.

3.1.3. Advanced

A problem in BSO is that onlooker bees only move based on the past experience of the forager bees. Because of this, the population could quickly ignore a part of the search space. [1] solves this by introducing a repulsion factor. This encourages the bees to randomly ignore the direction of the rest of the swarm. The onlookers and experienced foragers change their position update formula into 3.4) and (3.5, respectively.

$$x_{ij}(t+1) = x_{ij}(t) + sign(r) * c_2 * r_2 * (elite_j(t) - x_{ij}(t)) \tag{3.4}$$

$$x_{ij}(t+1) = x_{ij}(t) + sign(r) * [c_1 * r_1 * (pbest_{ij}(t) - x_{ij}(t)) + c_2 * r_2 * (gbest_j(t) - x_{ij}(t))] \tag{3.5}$$

with $sign(r)$ equal to (3.6), P_{rf} equal to some predefined probability, and r as some random number. This helps increase the diversity of the populations and introduces some additional stochastic influence.

$$sign(r) = \begin{cases} 1, & \text{if } (r \leq P_{rf}) \\ -1, & \text{if } (r > P_{rf}) \end{cases} \tag{3.6}$$

[1] gave the experienced foragers memory, enabling them to use historical information about previously visited high-fitness locations. This makes the algorithm stronger, and less prone to be stuck in local minima.

3.1.4. Summary

BSO has been proposed for numerical function optimization [1], multi-target search and coverage in an unknown area [25] and multi-robot and multi-target search in an unknown environment [45]. The main advantage of BSO, compared to, for example, PSO, is that BSO agents are classified into three search patterns [45]. This automatically balances exploration and exploitation. BSO uses intensive communication to balance exploration and exploitation by agents sharing their fitness with each other. BSO is somewhat similar to PSO in terms of the movement of single agents but has some overhanging framework to determine social behaviour. Original BSO requires communication between experienced foragers and onlookers, and must therefore be modified for tasks where communication (range) is limited. Scouts start by randomly searching the space. Therefore, the initial placement does not influence the performance of the algorithm as much as with other algorithms where exploration is less prominent. Another important feature of BSO is the recruitment of help within the swarm. The ability of a robot to decide whether it should help a peer exploit an area has great potential in cases where areas of interest are distant in a clustered fashion, as this requires a good balancing of the properties of the algorithm.

3.2. Modified bee swarm optimization

The original Bee Swarm Optimization (BSO) algorithm is designed for finding global optima in information-rich environments, as experienced foragers and onlookers have to continuously communicate fitness levels for the algorithm to work wholly. In this case, where agents try to converge to a single global optimal target while continuously measuring an ever-changing fitness function, this function optimization-like movement method would be suitable. This could, for example, be helpful for finding the source of a gas cloud or the deepest part of a cave. However, if targets are dispersed and rare, and there are communication range limitations, agents will lack information to coordinate their movement. Without continuous input, the performance of BSO would quickly stagnate in local optima. Limited communication would result in the experience foragers not being able to communicate with onlookers, which is one of the important elements of BSO. To solve these issues, the BSO algorithm is modified to handle these newly imposed conditions effectively.

1. **Increase exploration through full-swarm scout initialization:** In BSO, robots are classified as scouts, onlookers or experienced foragers based on their fitness values. However, in this study, robots have no knowledge of their fitness level during initialization, making this classification impossible. This is solved by initializing all robots as scouts. This means all robots are exploring at the beginning stages of the algorithm, where exploration is also desired the most. The goal of a scout is explained in Sect. 3.2.2.
2. **Exchange of target information instead of individual robot fitness levels:** Agents in BSO are driven by the higher fitness value of other bees and share only this information. However, with limited communication capabilities in combination with sparsely distributed targets, this fitness value of neighbouring robots may not be so relevant. To address this issue, Modified Bee Swarm Optimization (MBSO) uses knowledge sharing of found and received target locations between robots in the neighbourhood. Each robot keeps a target list that is compared and truncated with other neighbouring robots. This enables robots to keep an up-to-date target list that evolves throughout the swarm. More details are provided in Sect. 3.3.
3. **Environmental stimuli-based scout-to-onlooker transition:** In BSO, onlookers choose an experienced forager to move towards based on a roulette wheel. In MBSO, the onlooker variable O is introduced to help with this decision-making. Scouts can individually reason about aspects of their environment by using O and their current target knowledge, and decide whether to become an onlooker or not. This enables scouts to determine if, when, and for which target to become an onlooker. This decision-making process is based on an emotional model, extensively described in Sect. 3.4. For example, a scout may choose to become an onlooker for a target if it is in close proximity to the target, has few other robots within its communication range, and has sufficient battery capacity to reach the target.
4. **Environmental stimuli-based experienced forager-to-scout transition:** The experienced forager engages in local exploration through Dynamic Lévy Walk (DLW), scaling down the original step size associated with global search with a factor of size b. To incrementally go back from exploitation to exploration, b gradually increases from b_{MIN} to b_{MAX} to favour exploration as the step size returns to normal. The speed of this change is also managed by O. The experienced forager dynamically adjusts the scale factor b based on environmental stimuli, such as the presence of other scouts in its local exploitation area, to encourage exploration or exploitation as needed. This dynamic swarm regulation aims to prevent robots from piling up in the same regions and drive robots away from each other during DLW. Further details can be found in Sect. 3.7.

3.2.1. General aspects

To avoid repeating the same information for every type of robot, e.g., scouts, onlookers and experienced foragers in MBSO, some general characteristics that apply to all types are described. Then, more in-depth properties of these different types of robots in MBSO are described in the subsequent sections. All types of robots truncate target information with other robots in range. It does not matter which state a robot is in. This allows swarm intelligence to be formed. This ad hoc communication method will be further explained in Sect. 3.3. Additionally, the emitter range of all robots is set to $100\,\mathrm{m}$.

All robots use the same controller and can switch between the categories of agents. The robots that are used in this research are Adept's Pioneer 3-At, which are equipped with 16 distance sensors that are placed around the robot. This enables the robots to use simple BraitenBerg avoidance to avoid each other, obstacles, and walls. These robots are then also equipped with a simplified model of a camera that is capable of target recognition. Targets can be recognized at a maximum distance of $15\,\text{m}$. The camera has an FoV of $1.05\,\text{rad}$, which is equal to around $60°$. The code associated with Braitenberg obstacle avoidance, target recognition, switching between agents and more is given in appendix B.

3.2.2. Scout

In Modified Bee Swarm Optimisation (MBSO), all robots are initiated as scouts. Scouts are tasked with finding targets through random walking. The process involves using a Lévy step generator to generate a step size drawn from a Lévy distribution. This step size is then set as the duration of the forward movement. The Lévy distribution is known for its heavy tail, which means it allows for occasional long steps that help exploration in a large search space, as is true in this case. More about this method of random walking can be found in Sect. 3.6. At the initialization stage, robots have no prior knowledge about the environment or the number of robots present. During random walking, scouts continue taking steps until a target has been encountered. Once a scout finds a target, it saves the target location information in an individual target array, and shares this target array with other robots within its communication range, as will be explained in Sect. 3.3. Scout bee i, Lévy Walks (LW), in a region with dimension $j = 1.., n_x$, scale factor with size $b = \text{b}_{\text{MAX}}$, in time t. The formula for this movement can be seen in (3.7).

$$x_{\text{scout}_{ij}}(t+1) = x_{\text{scout}_{ij}}(t) + b \cdot \text{LW}(x_{\text{scout}_{ij}}(t)) \tag{3.7}$$

Newly acquired targets, received by inter-robot communication, are added to the individual targets array. For the scout to determine whether to become an onlooker, it must check two things. Firstly, it checks if the closest target in its targets array is closer than the Target Recognition Threshold (TRT). This variable is introduced so that robots do not chase targets that are too far away, wasting energy. If this condition is satisfied, the scout checks if the onlooker variable O is set to favour exploitation or exploration. O enables the robot to reason about certain environmental variables. O is based on a framework introduced by [51, 52]. This framework is called the Architecture of Multi-robot Systems Heterogeneous Robots with Emergent Behaviour (AMEB) and will be described in Sect. 3.4. In essence, O determines if scouts should, or should not, become an onlooker, by making sense of their sensory information. This sensory information can, for example, be the number of robots in the vicinity, the battery level of the robot, or the number of obstacles in the neighbourhood. The flowchart of a scout, and the transition to an onlooker, are visualised in Fig. 3.2. Cluster recognition will be added to the algorithm later in Sect. 3.8.

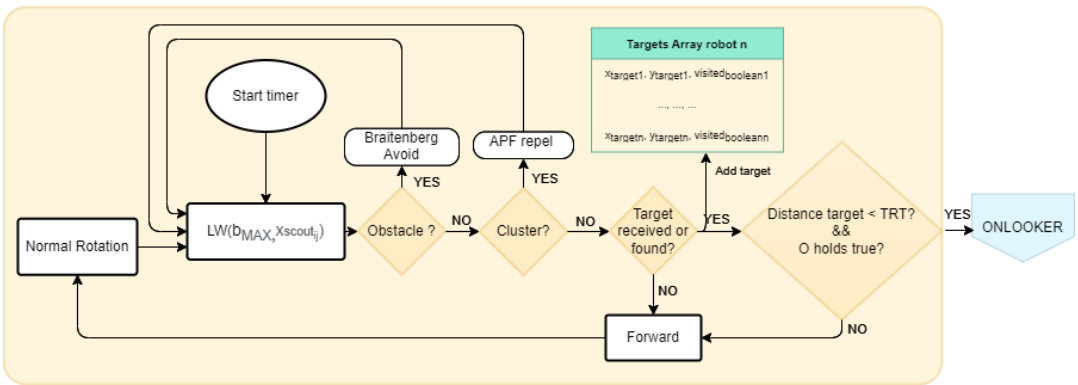


Figure 3.2: Flowchart of the behaviour of a scout in MBSO.

3.2.3. Onlooker

A scout inspects if any target in its target array fits the set requirements. If so, it decides to become an onlooker, as explained in Sect. 3.2.2. This choice is at the core of balancing the exploration and exploitation of the algorithm. The trajectory of the onlooker bee i in dimension $j = 1.., n_x$ is calculated by (3.8). More about how a scout chooses to become an onlooker by using the onlooker variable O can be found in Sect. 3.4.2. Onlookers move towards targets using an artificial potential field U, which will be elaborated on in Sect. 3.5.

$$x_{\text{onlooker}_{ij}}(t+1) = \begin{cases} x_{\text{onlooker}ij}(t) + \nabla U(x_{\text{onlooker}_{ij}}(t), x_{\text{closest\_target}_j}(t)), & \text{if } \vec{Z} < \text{TRT and } O \text{ holds } true \\ x_{\text{forager}_{ij}}(t+1), & \text{if } \vec{Z} \leq 5 \end{cases} \tag{3.8}$$

with $\vec{Z}$ as the Euclidean distance between the closest target in the target list and the position of the robot. When the onlooker reaches the target $\vec{Z} \leq 5$, it becomes an experienced forager. The target $x_{\text{closest\_target}_j}(t)$ from all targets l in the target array, that is closest to robot $i = 1.., n_x$, in time t, is the target with the highest fitness in dimension $j = 1.., n_x$, as shown in (3.9).

$$f(i, j, l, t) = \vec{Z} = min(||x\_robot_{ij}(t) - x\_target_{lj})(t)|| = closest\_target_j(t) \tag{3.9}$$

Target Recognition Threshold (TRT) in (3.8) is the minimum distance between the closest target and the robot to even take the target into account. In addition, the robot uses an emotional model to determine whether it should or should not become an onlooker. This is handled using variable O. More about this in Sect. 3.4. The architecture enables one to add multiple other variables that could influence the exploration and exploitation properties of the algorithm, by reasoning about environmental variables. Some examples of variables that could be added to the architecture are, for example, the performance of the robot (OS), amount of obstacles in its neighbourhood (SS), Battery State (BS), and more [51, 52]. In this study, however, only the Interaction State (IS) is considered. IS corresponds to the number of robots in the emitter range of the current robot. The onlooker variable O is defined as shown in (3.10), but is unrestricted to multiple inputs.

$$O = \begin{cases} true \text{ (exploitation)}, & \text{if } IS < \text{IS}_{\text{threshold}}, (OS < \text{OS}_{\text{threshold}}, BS > \text{BS}_{\text{threshold}}, \ldots) \\ false \text{ (exploration)}, & \text{otherwise} \end{cases} \tag{3.10}$$

When a scout receives an interesting target position and decides to become an onlooker, it moves towards the chosen target using an artificial potential function, as will be described in Sect. 3.5. During this movement, robots are continuously attracted by the target location with the highest fitness, based on (3.9). If a robot finds a new target or receives a target with higher fitness, it will re-orientate itself towards the new target. When an onlooker reaches the target location, it marks the target as visited in the targets array, and becomes an experienced forager. Targets that are found by the current robot are also marked as visited. The handling of targets and communication methods will be further explained in Sect. 3.3.

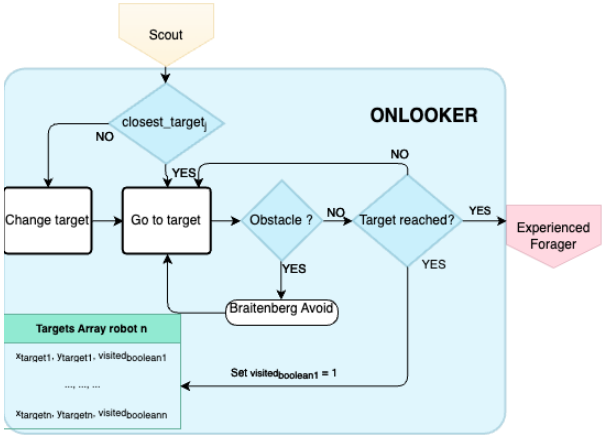


Figure 3.3: Flowchart of the behaviour of an onlooker in MBSO.

3.2.4. Experienced forager

Experienced foragers are tasked with exploiting areas of interest. After an onlooker has reached a target position, the robot becomes an experienced forager. Foragers update their position using (3.11). When onlookers turn into experienced foragers, they start making very small steps by setting $b \leftarrow \mathrm{b_{MIN}}$. This sets the Lévy step generator to favour maximum exploitation. From there on, the foragers will incrementally increase scale factor b with increments of $\gamma_{\text{Lévy\_exploit}}$ until $b = \mathrm{b_{MAX}}$. It could happen that after turning into a forager, the environment changes and exploration is favoured again. If the experienced forager senses that it should be exploring instead of exploiting, based on the same environmental variable O, it changes scale factor b such that it now takes larger increments with size $\gamma_{\text{Lévy\_explore}}$. This ensures the robot changes back to a scout more quickly. This whole process is called Dynamic Lévy Walk (DLW) and is elaborated on in Sect. 3.2.4.

$$x_{\text{forager}_{ij}}(t+1) = \begin{cases} x_{\text{forager}_{ij}}(t) + b(\gamma_{\text{Lévy\_exploit}}) \cdot \text{DLW}(x_{\text{forager}_{ij}}(t)) & \text{if the condition } O \text{ holds } true \\ x_{\text{forager}_{ij}}(t) + b(\gamma_{\text{Lévy\_explore}}) \cdot \text{DLW}(x_{\text{forager}_{ij}}(t)), & \text{if the condition } O \text{ holds } false \\ x_{\text{scout}_{ij}}(t+1), & \text{if } (b = \mathrm{b_{MAX}}) \end{cases} \tag{3.11}$$

with $\gamma_{\text{Lévy\_exploit}} \ll \gamma_{\text{Lévy\_explore}}$. O influences both scouts and experienced foragers. The difference is that O influences the **speed** of an experienced forager changing back to a scout. Alternatively, it influences the **decision** of a scout on whether it should become an onlooker. Visualisation of the different influences of O on a scout and experienced forager are visualized in Fig. 3.9. A visualisation of a forager's behaviour is shown in Fig. 3.4.

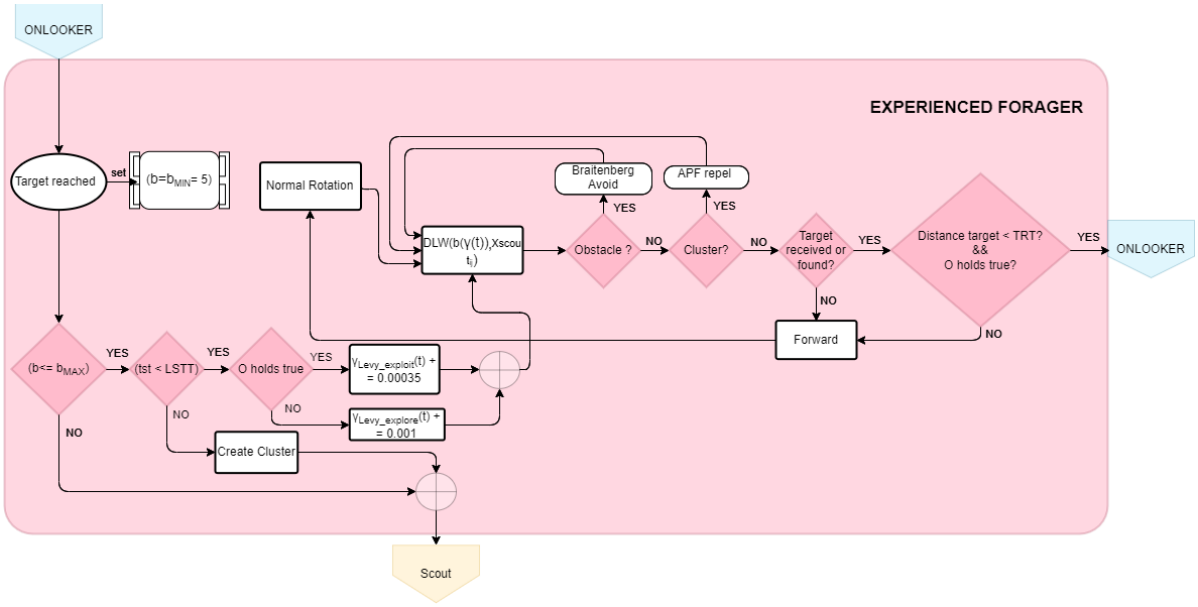


Figure 3.4: Flowchart of the behaviour of an experienced forager in MBSO.

3.3. Decentralized ad hoc communication of target arrays

In Swarm Robotics (SR), it is often desired to introduce a so-called 'neighbourhood', this is the range of a group of robots where individuals can sense or communicate with each other. To enable communication within a robot's neighbourhood, often a Mobile Ad hoc NETwork (MANET) is set up [12]. Once a robot detects another robot, communication is initiated. MANETs consist of special kinds of wireless mobile nodes that form a temporary network without using any infrastructure or centralized administration. Robots can join or leave these networks at any time, enabling a swarm to scale up and down as needed. There is no fixed infrastructure, nodes are equal and there is no centralized overview or control in a MANET. The differences between communicating using a cellular network and a MANET are displayed in table 3.1. Aspects that make a MANET valuable for a swarm system are its rapid deployment capabilities, dynamic network topologies, ability to function in hostile environments with noise and losses, adaptability of the network and the cost-effectiveness of the system.

Cellular network	**MANET**
Infrastructure networks	Infrastrucureless networks
Fixed, prelocated cell sites and base station	No base station, and rapid deployment
Static backbone network topology	Highly dynamic network topologies with multihop
Relatively caring environment and stable connectivity	Hostile environment (noise, losses) and irregular connectivity
Detailed planning before base station can be installed	Ad hoc network automatically forms and adapts to changes
High setup costs and large setup time	Cost-effective and short setup time

Table 3.1: MANET network compared to a cellular network in SR [58].

The Modified Bee Swarm Optimization (MBSO) algorithm takes into consideration the emitter range of the robots. Using a MANET for the proposed swarm would facilitate a more practical approach to the development of such a system in rural and inaccessible settings like extraterrestrial exploration, subsurface investigation, and hazardous environments. In the current study, it is assumed that robots use this ad hoc method of communicating, with a maximum emitter range of $100\,\mathrm{m}$, as the connection range of a MANET is often limited to this range [65]. However, different communication speeds and bandwidths require different communication protocols, effectively changing the possible communication range of a MANET.

Instead of sharing fitness levels between robots, like in bee swarm optimization, members of MBSO share, compare and store target information. This enables individual knowledge to truncate and develop throughout the swarm, forming swarm intelligence. The process starts with robots keeping individual target arrays that are increasingly being filled with target positions. Target positions are estimated by using a camera, global navigation satellite system and inertial unit of the robot. These arrays are compared with other robots in the neighbourhood and any unfamiliar targets are added to the individual target arrays. In addition to the target positions, hereby defined as $(x_{\text{target}_n}, y_{\text{target}_n})$, these arrays also keep track of whether the robot has visited or received the target. This is tracked by a Boolean called; $visited_{\text{boolean}_n}$. Received targets always get marked as unvisited. Robots use this target array to determine the closest unvisited target (the target with the highest fitness, as seen in (3.9)), which then becomes a potential target for onlookers. Comparing target arrays enables robots to determine the Interaction State (IS), which drives variable O, as previously explained in Sect. 3.2.2, Sect. 3.2.3 and Sect. 3.4.

In addition to onlookers using their target array to determine potential targets, this array is also used to recognize clusters. The functionality of cluster recognition will be explained in Sect. 3.8. Briefly explained, targets that are believed to be in a cluster (within a range of $75\,\mathrm{m}$), are recognized as a cluster. The average position of the cluster is calculated and saved in a repulsive list. The repulsive list is also kept individually per robot. This repulsive list is used as input for the artificial potential field, repelling robots from readily visited areas. The information that robots send only has to be updated every few iterations, as target information changes infrequently. It is assumed these target arrays can be sent through an ad hoc network with low communication speed. The need for continuous

communication is removed using this communication method, possibly increasing the robustness of MBSO against BSO, and other techniques requiring continuous communication. A flowchart of this information-sharing method is shown in Fig. 3.5.

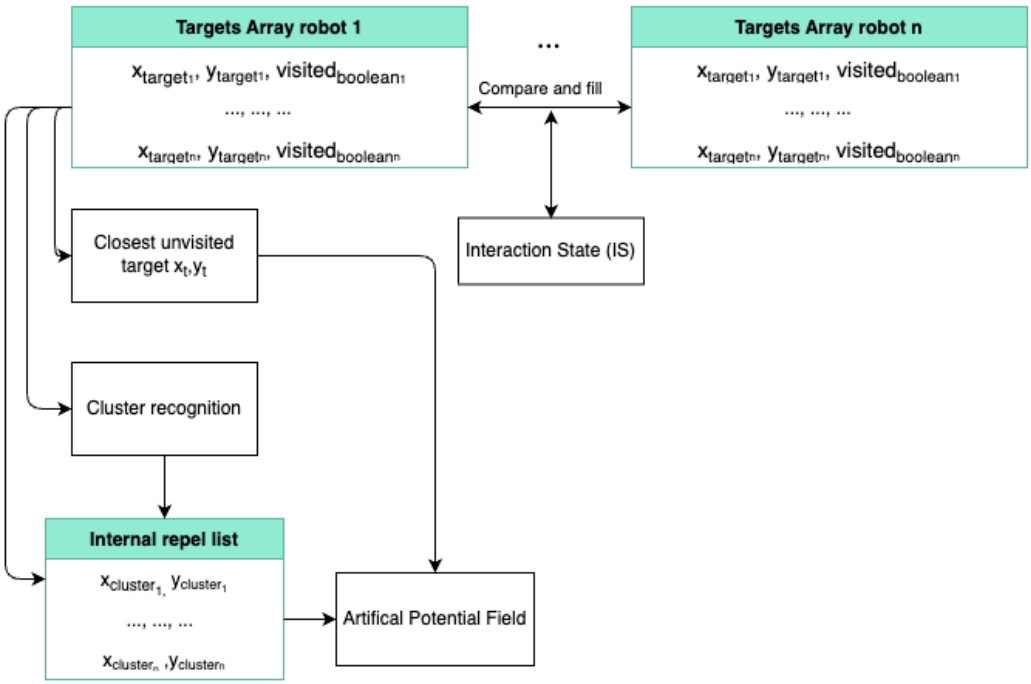


Figure 3.5: Information sharing between robots results in the interaction state, closest unvisited target, and repel list.

3.4. Architecture for multi-robot systems with emergent behaviour

To be able to call a system a swarm system instead of a multi-robot system, the architecture must allow decentralised control and collective robot intelligence to emerge. Therefore, it is necessary to define an architecture that enables (i) decentralised control, (ii) building individual and collective knowledge, and (iii) collective behaviour to emerge from individual robotic behaviour [17].

3.4.1. AMEB

Architecture Multi-robot systems heterogeneous robots with Emergent Behaviour (AMEB) is an architecture specifically designed for handling swarm robots, allowing the emergence of self-organising behaviour. The general structure of this architecture is displayed in Fig. 3.6. Robots use environmental stimuli to calculate a satisfaction index. The emotional state of the robots is tied to a satisfaction index, as shown in Fig. 3.7, and results in individual robot behaviour. The combined actions of singular robots emerge into collective behaviour. The knowledge of individual robots and collective behaviour then results in collective knowledge being created.

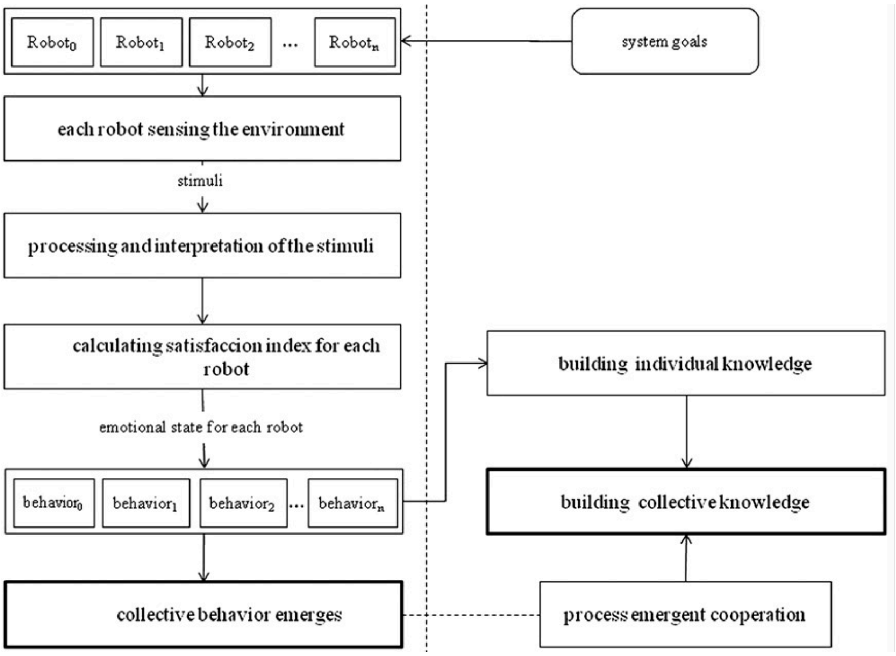


Figure 3.6: General operation of AMEB, inspired by [17].

The affective layer of AMEB has originally been proposed by [51, 52]. The model considers positive or negative emotions that affect the level of self-organisation and emergence of the system. They also affect the individual and collective behaviour of robots. This affective component is transverse to the cognitive and social components of an algorithm (see $c_1 \cdot r_1$, $c_2 \cdot r_2$ in (3.3)). Joy, for example, drives the robot to be social and imitate other robots, whereas anger stimulates reactive and individual behaviour. The emotional model is affected by environmental stimuli and the current state of the robot. A visualisation of the emotional model can be found in Fig. 3.7.

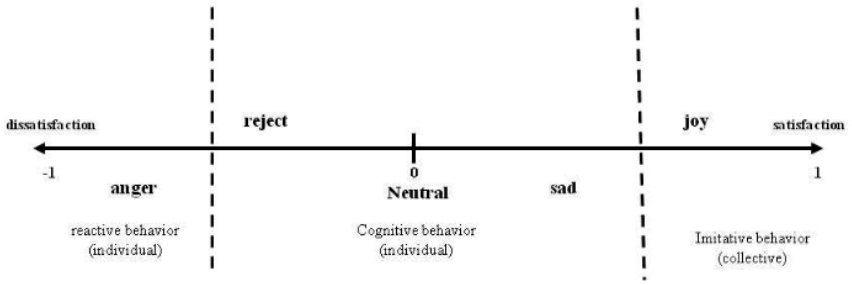


Figure 3.7: Emotional model in AMEB [51, 52].

The interpretation of environmental stimuli results in states, which in term influence the emotional state of the robot [17]. Such states are:

1. Battery State (BS): energy level.
2. Operation State (OS): performance level.
3. Security State (SS): collision average.
4. Interaction State (IS): the number of sent or received messages.

3.4.2. AMEB for modified bee swarm optimization

Robots receive different sensory stimuli. In this study, the inputs are mainly from the camera, distance sensors, receiver, battery, inertial unit and wheel sensors. These values are interpreted and processed. Camera recognition in combination with GNSS enables the robot to determine target locations, building an individual target array and thus individual knowledge. The interactions between receivers and emitters enable robots to make an Interaction State (IS), the battery sensor results in the Battery State (BS), and the distance sensors enable the robot to create a security state (SS). From the targets that are found by that particular robot, the current robot's performance can be determined. This is also known as the Operational State (OS). These states are used to determine the current emotion of the robot. Collective behaviour emerges from these inter-robot interactions. Additionally, collective knowledge is built by sharing and updating individual target arrays, creating swarm intelligence on a macro level. The interpretation of Fig. 3.6 for MBSO is shown in Fig. 3.8. For the sake of simplicity, the current study only uses the Interaction State (IS). This interaction state results in the robots being satisfied and thus performing exploitation, or dissatisfaction, resulting in exploration. This choice is reflected in the onlooker variable O. Robots in MSBO reason about their environment (the number of robots), by using sensory input (receiver stimuli) and use it to determine whether they want to exploit (are satisfied) or explore (are dissatisfied) using O. The architecture enables more environmental variables to be added to the decision-making process in a structural matter, as seen in Fig. 3.8.

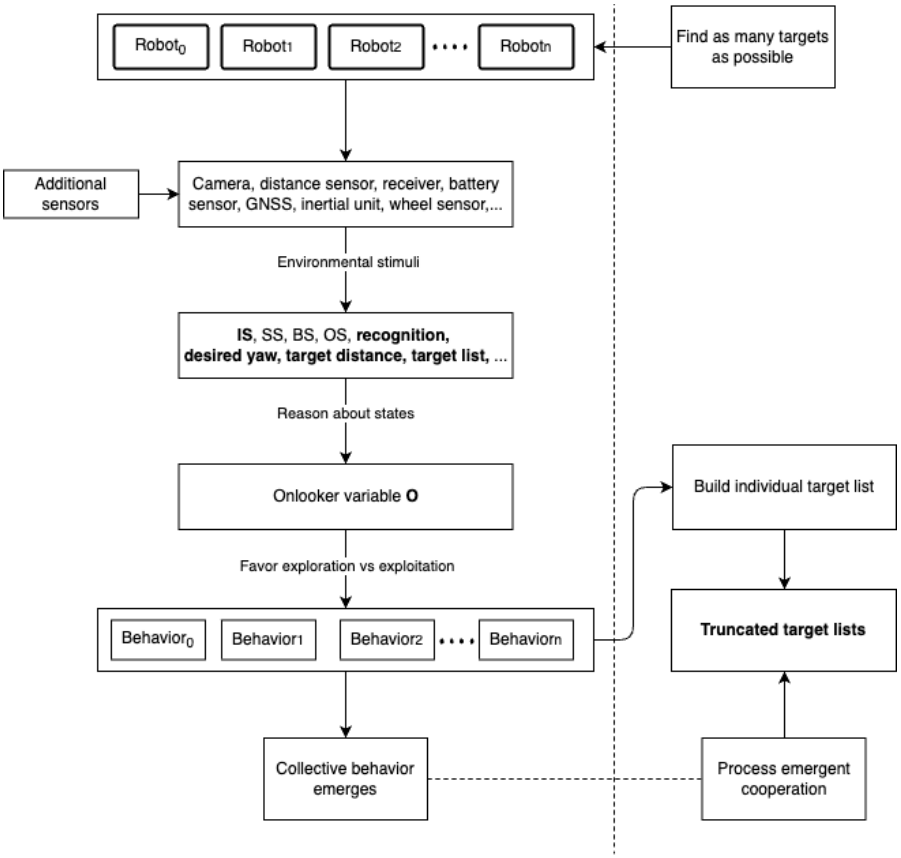


Figure 3.8: General operation of AMEB in MBSO, inspired by [17].

The onlooker variable O is used in two ways. Firstly, to decide whether a scout becomes an onlooker and starts exploiting. Secondly, to decide how quickly an experienced forager turns back into a scout. Effectively, the structure enables the balancing of the algorithm by influencing the two agents that decide the exploration and exploitation properties of the MBSO algorithm. In Fig. 3.9, the influence of AMEB in MBSO is visualized.

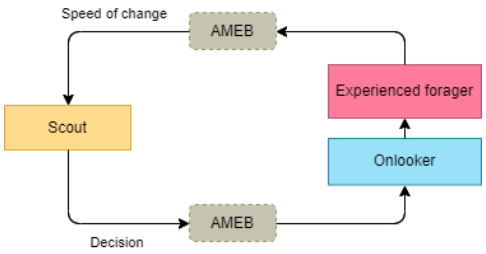
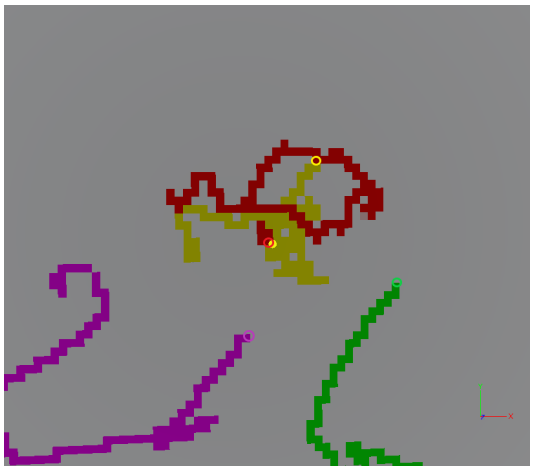


Figure 3.9: Influence of AMEB in decision making in MBSO.

O makes sure that there is a limit on the number of robots that are in each other's neighbourhood that can become an onlooker. Robots can make this choice individually. This puts a maximum on the number of robots that may converge on the same solutions. This essentially creates sub-swarms with a cap on the number of robots, similar to new research done by Zhou, Chen, Zhang, et al. (2020) [75]. The mentioned study found that this sub-swarm formation method in combination with an artificial potential field results in good performance of collision avoidance, effectively reducing the collision conflicts among robots, environments and individuals.

In the following example, 4 robots are placed in a small environment with one target in the middle. The robots check whether the number of interactions is below the set threshold $\text{IS}_{\text{threshold}}$, which is set to 1 for Fig. 3.10a and is set to 2 for Fig. 3.10b. (a maximum of 2 and 3 robots may become onlookers, respectively). The influence of O is demonstrated in these cases. On the left, where there can only be 2 robots, the green and purple robots completely ignore the received target information. In the figure on the right, the green robot ignores the target position, as there are too many robots in the neighbourhood. Additionally, if the robots that are still global searching come into the neighbourhood of the experienced foragers, performing a local search, the experienced foragers increase their step size according to (3.11).



(a) Four robots are initialized at the coloured circles. The red robot is initialized at the location of the only target situated at the white point and recognizes the target. Target information is truncated for all robots. The purple and green robots ignore the target, and the yellow and red are performing local searches. $\text{IS}_{\text{threshold}} = 1$.

(b) Four robots are initialized at the coloured circles. The red robot is initialized at the location of the only target situated at the white point and recognizes the target. Target information is truncated for all robots. The green robot ignores the target, and the yellow, red and purple robots are performing local searches. $\text{IS}_{\text{threshold}} = 2$.

Figure 3.10: A visualization of the use of O with $\text{IS}_{\text{threshold}}$ set at 1 and 2.

3.5. Artificial potential field

To enable onlookers to move towards targets, an Artificial Potential Field (APF)-guided movement method is introduced to the algorithm. APF in this study is also used to repel robots from clusters and walls in the simulated area. This chapter explains how APF is used to guide robots towards targets, and away from unattractive positions. Consider the Cartesian coordinate of a robot $q = (x, y)^T$. The total potential field can then be calculated as seen in (3.12).

$$U(q) = U_{att} + U_{rep} \tag{3.12}$$

where $U(q) =$ artificial potential field, $U_{rep}(q) =$ repulsive field, $U_{att}(q) =$ attractive field. The total force on the robot is the negative gradient of the attractive and repulsive potential and is calculated as follows in (3.13) [44].

$$\begin{aligned} F(q) &= -\nabla U(q) \\ &= -\nabla U_{rep}(q) - \nabla U_{att}(q) \\ &= F_{att}(q) + F_{rep}(q) \end{aligned} \tag{3.13}$$

where $F(q) =$ artificial force, $F_{att}(q) =$ attractive force and $F_{rep}(q) =$ repulsive force. The attractive field between the robot and the goal helps drag the robot to the goal area and is defined as seen in (3.14):

$$\begin{aligned} U_{att} &= \frac{1}{2} \cdot k_a \cdot (q - q_d)^2 \\ &= \frac{1}{2} \cdot k_a \cdot \rho_{goal}^2(q) \end{aligned} \tag{3.14}$$

where $k_a =$ attractive gain coefficient, $q =$ current position vector of the robot, q_d as the current position vector of the target. In addition, $\rho_{goal} = ||q - q_d||$ is the Euclidean distance from the robot's position to the goal position. The attractive force is then simply calculated by (3.15) [59].

$$F_{att} - x(q) = -k_a \cdot (q - q_d) \tag{3.15}$$

The repulsive artificial field repels robots away from obstacles, however, when the robot is at some Repulsion Distance (RD) from the obstacle, the robot motion must not be influenced by this repulsive force anymore [59]. This field is therefore range limited, as seen in (3.17).

$$U_{rep}(q) = \begin{cases} \frac{1}{2} \cdot k_b \cdot (\frac{1}{d(q)} - \frac{1}{d_o})^2, & \text{if } d(q) \leq \text{RD} \\ 0, & \text{if } d(q) \geq \text{RD} \end{cases} \tag{3.16}$$

where $k_b =$ repulsion gain coefficient, $d =$ distance between the robot and the obstacle. The force declines quickly when the distance between the obstacle and the robot increases and goes to infinity when this distance goes to zero. The corresponding repulsion force function is calculated as follows [40]:

$$F_{rep}(q) = \begin{cases} k_b(\frac{1}{d(q)} - \frac{1}{d_0}) \cdot (\frac{1}{d^2(q)}) \cdot (\frac{q - q_c}{||q - q_c||}), & \text{if } d(q) \leq \text{RD} \\ 0, & \text{if } d(q) \geq \text{RD} \end{cases} \tag{3.17}$$

The total forces from obstacles that are detected are summed up. This sum is then used to calculate one total force vector [44, 55]. The force vector is used to determine the desired angle of the robot, which in turn moves the robot towards the right position. APF gives the possibility to be used as a more complex obstacle avoidance method [23, 74, 75].

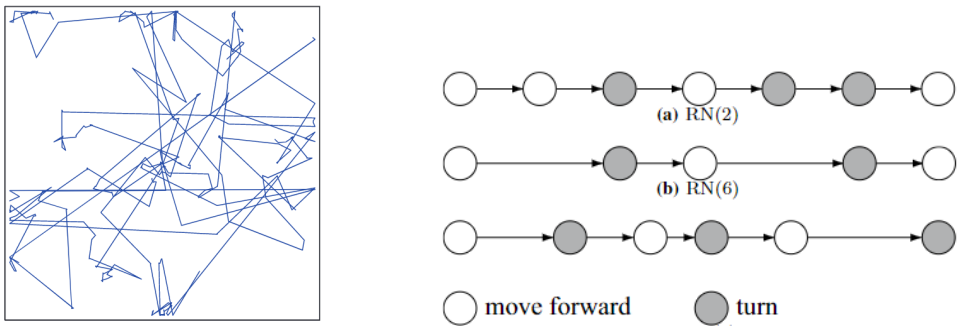
3.6. Lévy walk generator

Lévy walk (LW) is widely used as a random exploration method in robotics. Its movement has been derived from chaos theory and is particularly effective for simulating random or pseudo-random natural phenomena. Among these natural collective systems, LW, inspired by Lévy flight, is one of the most efficient random walk patterns [28, 32]. In real robot experiments, LW has also been proven to outperform traditional random walks for exploration purposes [27]. LW has even been recognized as one of the most efficient search strategies methods for unknown environments [16, 30, 47, 49, 57]. Multi-robot swarming algorithms often start off in some manner of random walking before iterating towards (partial) solutions.

3.6.1. Introduction of Lévy walk

Random walking in this thesis is used for both the exploration of targets and the exploitation of target locations. Striking a balance between exploration and exploitation has been an essential part of random searching algorithm design [48]. Too much exploitation can result in agents getting stuck in local optima and result in poor performance, whereas excessive exploration can be inefficient in cases where targets are clustered, or energy is limited. A key task of autonomous mobile robots is to explore unknown environments with limited energy capacity, and other constricting conditions. The exploration method must therefore be efficient, and ensure the maximum area is covered while minimizing overlap.

LW generates smaller step sizes in high frequency with an occasional larger step size. This behaviour is visualised in Fig. 3.11a. Khaluf, Van Havermaet and Simoens (2018) [31] gave an example of a multi-robot collective LW. Robots move with a fixed linear speed and sample the duration of their next step from a Lévy distribution. After the linear step, a rotating state is initialised. The rotation is performed over a constant angular velocity, sampled from a uniform distribution. The transition between the movement and rotation phase has been visualized in Fig. 3.11b. The rotational step is always taken after the Lévy step has been completed.



(a) Lévy walk with 10 robots [49].

(b) Transition between forward movement and rotation [28].

Figure 3.11: Visualisation of Lévy walking.

3.6.2. Mathematical foundation of Lévy walk

A random variable is stable (also defined as α-stable or Lévy stable) if a linear combination of two independent copies of the random variables has the same distribution as the random variable itself. Stable distributions, are generally characterised by [22]:

1. Stability index $\alpha \in (0, 2]$, which is also called the tail index, or characteristic exponent. It determines the rate at which the tails of the distribution taper off. The smaller α is, the greater the size of extreme events. When $\alpha > 1$, μ exists and is the mean of the distribution.
2. Skewness parameter $\beta \in [-1, 1]$. When positive, the distribution is skewed to the right, meaning the right tail is thicker. When $\beta = 0$, this distribution is symmetric about μ.
3. Scale parameter $\sigma > 0$, which determines the width of the probability density distribution.
4. Shift parameter $\mu \in R$, which is the shift mode, or peak, of the distribution. It is similar to the mean in a normal distribution. The stable random variable, therefore, is expressed as $X \approx S(\alpha, \beta, \sigma, \mu)$.

Lévy walk (LW) is a random search method that is based on selecting a step size from a heavy-tailed probability distribution. This distribution maximizes the search efficiency when the distribution of the targets is sparse, and the searchable area is large [31]. The Lévy distribution, in general, can be interpreted as a distribution of a sum of numbers N identically and independently distributed. The Fourier transform takes the form shown in (3.18) [70]:

$$F_N(k) = \exp[-|k|^{\beta}] \tag{3.18}$$

For this general case, the inverse integral (stable probability distribution) can be described in the following form [38]:

$$P(s) = \frac{1}{\pi} \cdot \int_0^{\infty} e^{-\gamma \cdot q^{\alpha}} \cdot \cos(qs)\, dq \tag{3.19}$$

(3.19) is symmetrical with respect to $s = 0$. γ is a scaling factor with $\gamma > 0$, and α controls the shape of the distribution, requiring $0 < \alpha < 2$. Getting the actual distribution is not as easy as just taking the inverse of (3.18). The mathematical description of the Probability Density Function (PDF) and Cumulative Density Function (CDF) can only be described in a few special cases. When β in (3.18) is $\beta = 1$, the corresponding integral corresponds to the Cauchy distribution. When $\beta = 2$, it corresponds to the Gaussian distribution. The functions PDF and CDF do not have closed expressions, but in the special case where $\alpha = 0.5$ and $\beta = 1$, (3.19) can be described using its characteristics function in (3.20). [10, 20, 22, 70]:

$$P(s, \gamma\mu) = \begin{cases} \sqrt{\frac{\sigma}{2\pi}} \cdot \frac{1}{(s-\mu)^{3/2}} \cdot \exp(-\frac{\sigma}{2(s-\mu)}) \frac{1}{(s-\mu)^{3/2}}, & \text{if } 0 < \mu < s < \infty \\ 0, & \text{if } s \leq 0 \end{cases} \tag{3.20}$$

The scaling factor γ in (3.19) can, without loss of generality, be set to $\gamma = 1$. When this value is fixed, for large values of s, (3.19) can be approximated by the simple power law:

$$L(s) \sim |s|^{-1-\beta}, \qquad |s| >> 1 \tag{3.21}$$

where $0 < \beta \leq 2$. When $\beta \approx 1$, LW optimizes random search over a wide range of target distributions, including sparse, heterogeneous, patchy, scale-free and fractal [47]. The beauty of working with the Lévy distribution is that it has a scale-free characteristic, meaning that (3.21) is invariant when scaling $s \to b \cdot s$, with $b \in R$. $\beta \approx 1$ is the value that is deemed optimal when there is an absence of knowledge about the target distribution [31]. Other recommendations mention setting $\beta = 0.2$ [27, 28]. The different probability functions $P(s)$ associated with this varying β in the simplified power distribution representation of LW are visualized in Fig. 3.12.

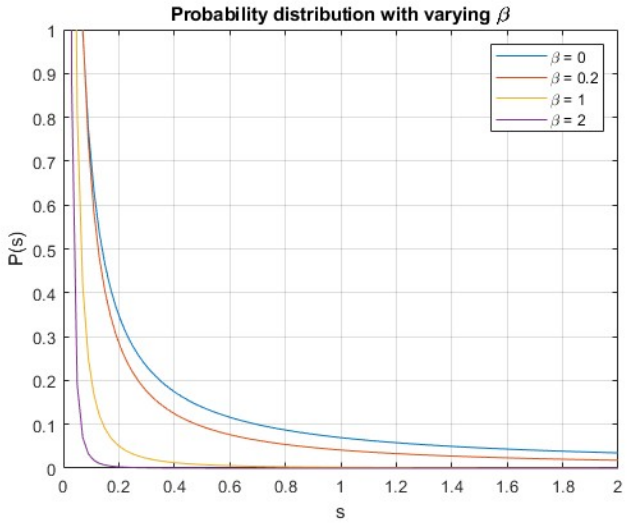


Figure 3.12: Probability distribution of (3.21) with varying β.

3.6.3. Optimizing Lévy parameters for modified bee swarm optimization

Viswanathan, Afanasyev, Buldyrev et al. (2000) [66] derived equations in order to determine skewness parameter β, based on the relative distance between target sites. The mean number of flights, or movements N, between successive target sites is approximated by:

$$N \approx (\frac{\lambda}{r_v})^{\frac{\beta}{2}} \tag{3.22}$$

where λ is the average distance between successive target sites, and r_v is the sensing range of the robot in question. From this formula, the optimal choice of β in (3.21) can be approximated by

$$\beta = 1 - \zeta \tag{3.23}$$

where

$$\zeta \approx \frac{1}{(ln(\lambda/r_v)^2} \tag{3.24}$$

This means the optimal β value between clusters can be calculated, knowing that clusters are spaced around $\lambda = 500\,\text{m}$ from each other. The sensing range of a robot is $r_v = 15\,\text{m}$. This results in the following values: $\zeta = 0.081$, and $\beta = 0.919$. Which indeed sets $\beta \approx 1$, the value that is deemed optimal when there is an absence of knowledge about the target distribution [31].

An approximation when calculating N results in robots reaching from one to another cluster within $N = 5$ moves. After testing this requirement, this number was not accurate. Because the Lévy distribution is scale-free, meaning that (3.21) is invariant when scaling $s \rightarrow b \cdot s$, with $b \in R$. The Lévy parameter can be scaled to fit (3.22) of moving from one to another cluster in any preferred number of moves. This enables the adjustment of the scale parameter based on relative cluster distances.

$$L(s) \sim b \cdot |s|^{-1.919}, \qquad \text{with } b(t) < \frac{\gamma(t)}{\gamma\_MAX} * 800 \tag{3.25}$$

The scaling parameter of the Lévy step generator was determined based on the average amount of moves until a distance of $\lambda = 500\,\text{m}$ was reached. This is the average distance between clusters. 8 Robots are initiated in the middle of a circle, and Lévy is scaled with some arbitrary number, here 25. The scale factor b is updated so that the average of the eight robots takes 11 moves to reach outside the circle's radius. This value is chosen based on trial and error and seems to be a good fit for random walking, showing signs of good circle coverage, with minimal overlap.

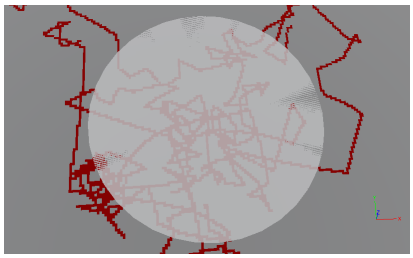


Figure 3.13: Counting the number of moves for 8 robots to move out of the circle with radius $500\,\text{m}$.

After testing different configurations it seems that $b_{MAX} = 33$ is the right scale parameter to move outside the circle in $N = 12$ moves, different than the approximated $N = 5$ moves in (3.22), returning an average step size of 175 'units', and reaching the mark of $500\,\text{m}$ using on average around 12 moves. The step size in the generator is used as time in seconds for the forward step to be taken. This means the robots, on average, move for 175 seconds forwardly within a step. Scale parameter b is also used to do local walk, where it is set to $b_{MIN} = 5$. Here, it returns, as expected, an average step size of 25 'units', or around $75\,\text{m}$. This seemed to be a good value to find targets within a cluster.

3.6.4. Normal rotation generator for Lévy walking

The rotation generator is a function that returns a random number from a normal (Gaussian) distribution with a randomized mean μ between [0, 2π]. The standard deviation is set to $\sigma = \pi$. The function generates and stores two static variables, $X1$ and $X2$. To generate each pair of random numbers, the function uses a variation of the Box-Muller transformation. This method involves generating two independent random variables $U1$ and $U2$ that are uniformly distributed between $[-1, 1]$. These variables are then transformed into two independent normal variables $X1$ and $X2$ using the following equations:

$$X_1 = U_1\sqrt{-2\ln W/W}, \quad X_2 = U_2\sqrt{-2\ln W/W}$$

where $W = U_1^2 + U_2^2$. Finally, the generator retrieves the final normally distributed number by implementing (3.26). The returned number is the angle the robot will take. The robots uses skid-steering to reach the desired angle. The normal generator uses the Box-Muller transform algorithm to generate random numbers from the standard normal distribution. The distribution is cut off between $[0, 2\pi]$. The function takes an individual custom seed value to ensure randomization between robots. Fig. 3.14 shows the distribution of the normal generator.

$$\text{result} = \mu + \sigma \cdot (X_1 \text{ or } X_2) \tag{3.26}$$

This results in a distribution that is made of multiple smaller distributions, with their tops distributed between $[0, 2\pi]$ as can be seen in Fig. 3.14.

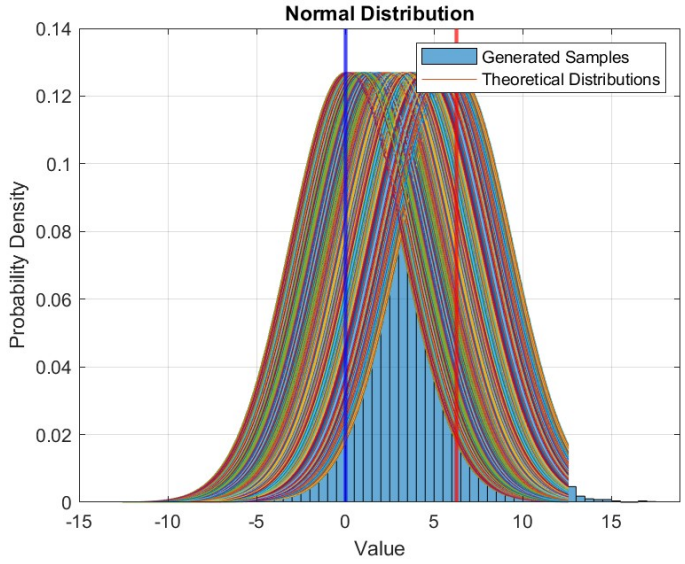
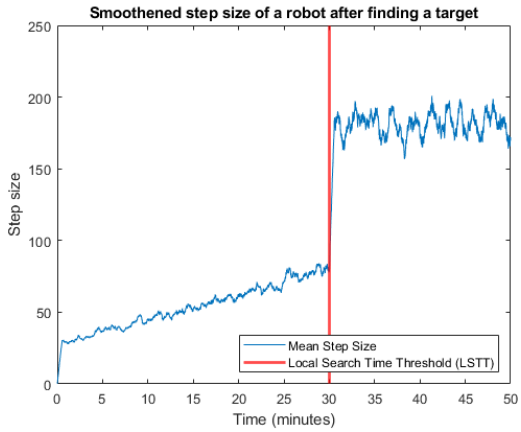


Figure 3.14: Normal distribution used for the rotating step in random walking.

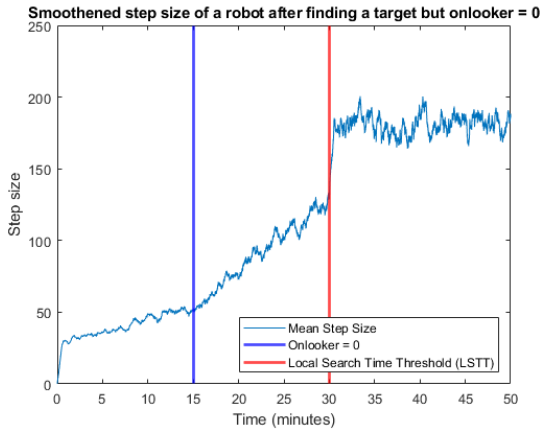
3.7. Dynamic Lévy walk

Lévy walk (LW) has been used as a random walking method for the initial search for targets. The step sizes for this random walking method are generated from a Lévy distribution, which is heavy-tailed. This means it favours small steps, with an occasional large step. After a Lévy step, the robot turns using the normal generator in the previous section. It then takes another Lévy step, and so on. However, whenever a robot finds and moves towards a target, it starts Dynamic Lévy walk (DLW) at the target location. DLW is a process where the scale parameter b is considerably reduced from b_{MAX} to b_{MIN}. This results in exploitation being favoured by the robot by taking much smaller steps. This scale parameter b gradually changes back to the initial Lévy global search value if no other targets are found. The rate of going back from local exploitation to global exploration is called Dynamic Lévy walking (DLW) and is equal to $\gamma_{\text{Lévy\_explore}}$ or $\gamma_{\text{Lévy\_exploit}}$, based on onlooker variable O. The introduction of the use of scale parameter b was given in Sect. 3.2.3, and was elaborated on in Sect. 3.6.

Onlooker variable O changes the speed at which an experienced forager increases scale parameter b. Fig. 3.15a visualizes the difference in the step size generation when onlooker variable O is true (favouring exploitation), and where it switches to false (favouring exploration) at $t = 15$ min in Fig. 3.15b. This could happen when $IS >$ IS$_{\text{threshold}}$, meaning too many robots are exploiting the same area. In this first example, the forager finds a target and variable O stays true, meaning it should keep exploiting the neighbourhood. In the second example, however, O becomes (and stays) false at $t = 15$ min. This means the experienced forager, tasked with exploitation, should rather favour exploration. It therefore quickly increases its scale factor b to make larger steps. When Local Search Time Threshold (LSTT) is reached at $t = 30$ min, the robots set $b \leftarrow$ b$_{\text{MAX}}$. More about this threshold in the next chapter, Sect. 3.8. This is an important variable as it determines the maximum time that may pass between finding a target and searching for new targets until the area is deemed as fully explored by the experienced forager.



(a) Average Lévy step while O is true (favouring exploitation).



(b) Average Lévy Step while O changes and stays false at $t = 15$ min (favouring exploration).

Figure 3.15: Difference between dynamic Lévy walking after a target has been found for $O =$ true and $O =$ false at $t = 15$ min.

3.8. Modified bee swarm optimization with cluster recognition

The Modified Bee Swarm Optimization (MBSO) algorithm uses Architecture Multi-robot systems heterogeneous robots with Emergent Behaviour (AMEB) to balance exploration and exploitation. Because this study focuses on finding targets that are gathered in clusters, the algorithm is improved by adding basic cluster recognition to the algorithm. Since this is an aspect that is particular to such cases, it should only be considered within these clustered target conditions. To minimize stagnation of the algorithm in local optima, increase inter-robot communication and teamwork, and give the algorithm more possibility to go to extremes, robots are equipped with the capability of recognizing and moving away from clusters. This way robots can go away from these explored regions after no additional targets are found.

Whenever a new target is found or received, this target is added to a robot's targets array, as explained in Sect. 3.3. If a target is found, the timer tst is started. The timer tracks the time that has passed between the last target that was found by the current robot. When this timer hits Local Search Time Threshold (LSTT$_{\text{threshold}}$), it combines all targets that are within the recognizable range with each other and adds the average position of the cluster to a repel list. The repel list is an individual list and keeps average cluster locations where robots will be repelled from, using the Artificial Potential Function (APF) described in Sect. 3.5. Robots that have communicated with each other, containing the same target arrays, will not have the same repel list if they have not visited the same targets, as tst is only started after a robot marks the target as visited (see Sect. 3.3). At the beginning of Sect. 3.2, a list of points, defining MBSO where given. The following point can be added to that list:

Cluster recognition removes the chance of robots revisiting the same clusters, and constrains the maximum search time within a cluster: To solve the issue of getting stuck in a local optimum, timer: '*time since target*', or tst, is introduced. This timer resets after every found target. When no additional targets are found by the robot after tst reaches threshold $\text{LSTT}_{\text{threshold}}$, the robot will recognise all targets that have been found within a certain distance from each other as a cluster. The average position of this cluster will be saved in a repel list. This is an individually kept list of clusters, recognized from visited targets, that robots will avoid. The repelling from the targets is done by using the previously defined APF.

The introduction of the repel list affects scouts and experienced foragers. Scouts are able to repel themselves from these average cluster coordinates. Scout bee i, now randomly walks in a region with dimension $j = 1.., n_x$, scale factor with size $b = \text{b}_{\text{MAX}}$, in time t. When the closest distance between any position $k = 1.., last\_cluster$ in the repel list, and the robot i, is closer than the Repulsion Distance (RD) (which has been defined in Sect. 3.5), the robot repels itself from this position until it is out of this repelling range. The new scout's movement equation becomes (3.27).

$$x_{\text{scout}_{ij}}(t+1) = \begin{cases} \text{Repel from Cluster}, & \text{if } (||\min(\text{repel\_list}_{kj} - x_{\text{scout}_{ij}}(t))|| \leq \text{RD}) \\ x_{\text{scout}_{ij}}(t) + b \cdot \text{LW}(x_{\text{scout}_{ij}}(t)), & \text{else} \end{cases} \tag{3.27}$$

The movement equation of experienced foragers is changed from (3.3) to (3.28). An additional condition to check is whether $tst \leq \text{LSTT}_{\text{threshold}}$. When $tst > \text{LSTT}_{\text{threshold}}$, foragers add the average coordinates within a recognizable range to their repel list and repel themselves a given distance RD from the cluster. Foragers then become scouts and keep the repel list such that scouts are repelled from these positions even during random walking.

$$x_{\text{forager}_{ij}}(t+1) = \begin{cases} x_{\text{forager}_{ij}}(t) + b(\gamma_{\text{Lévy\_exploit}}) \cdot \text{DLW}(x_{\text{forager}_{ij}}(t)) & \text{if } (tst \leq \text{LSTT}_{\text{threshold}} \text{ and } O \text{ holds } true) \\ x_{\text{forager}_{ij}}(t) + b(\gamma_{\text{Lévy\_explore}}) \cdot \text{DLW}(x_{\text{forager}_{ij}}(t)), & \text{if } (tst \leq \text{LSTT}_{\text{threshold}} \text{ and } O \text{ holds } false) \\ b \leftarrow \text{b}_{\text{MAX}} \text{ and Repel from Cluster}, & \text{if } (tst > \text{LSTT}_{\text{threshold}}) \\ x_{\text{scout}_{ij}}(t+1), & \text{if } (b = \text{b}_{\text{MAX}}) \end{cases} \tag{3.28}$$

Robots use an APF to repel themselves from average cluster positions in the target list. Because the repulsion force vector used in APF quickly becomes zero when the distance between the robot and cluster grows larger, an attractive angle is also added to move away from the clusters. The direction of the angle is set at $45°$ away from the cluster, relative to the robot's position. Meaning, if the cluster is at the top left of the robot, the robot will go $45°$ in the down-right direction. This means the robot always goes away from the cluster until RD has been reached. When the robot reaches a wall, it uses the same logic to move away from the wall. APF allows for more complicated movement, avoidance and guidance. However, for the purpose of this study, this method seems sufficient.

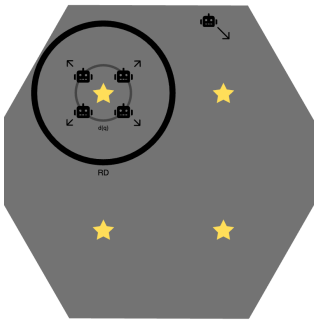


Figure 3.16: Visualisation of simple APF-guided movement away from clusters and/or walls.

3.9. Modified bee swarm optimization summary

The final flowchart of MBSO with cluster recognition is shown in Fig. 3.17. The algorithm contains three different kinds of agents, scouts, onlookers and experienced foragers. In Modified Bee Swarm Optimization (MBSO), all robots are initialized as scouts. This enhances the exploration properties of the algorithm in the beginning stages of MBSO. This change is necessary for unknown and large environments where communication is limited, as there is no information to act on, yet. Communication between robots is accomplished by truncating their individual target array. These are filled with information regarding found targets. These arrays between robots are compared and any differences are added to the individual's lists. This way, information can truncate throughout the swarm using an ad hoc communication method. In addition to target positions, information about if the target has been visited is also saved in these arrays. This information, however, is kept private and can be used to determine the next potential target position. Exploration is done by using Lévy Walking, which is an efficient category of random walking in unknown and large environments. When scouts find or receive targets, they reason whether to move towards these targets and start locally searching that area. This is decided based on Target Recognition Threshold (TRT) and Onlooker variable O. TRT looks at all targets in the target array and checks whether the target with the highest fitness, e.g., the closest target, is closer than a certain set threshold. This makes sure robots don't transfer to far-away targets. Alternatively, the onlooker variable O manages the emotions of the robot. If the robot is happy and also confines to TRT, it could decide to exploit that area. O is determined by, for example, the number of robots in its neighbourhood. If there are too many robots close by, the scout should probably keep exploring. There are many other factors that can be added to variable O. These determine the emotional state of the robot, and whether a scout should start exploitation or stay exploring.

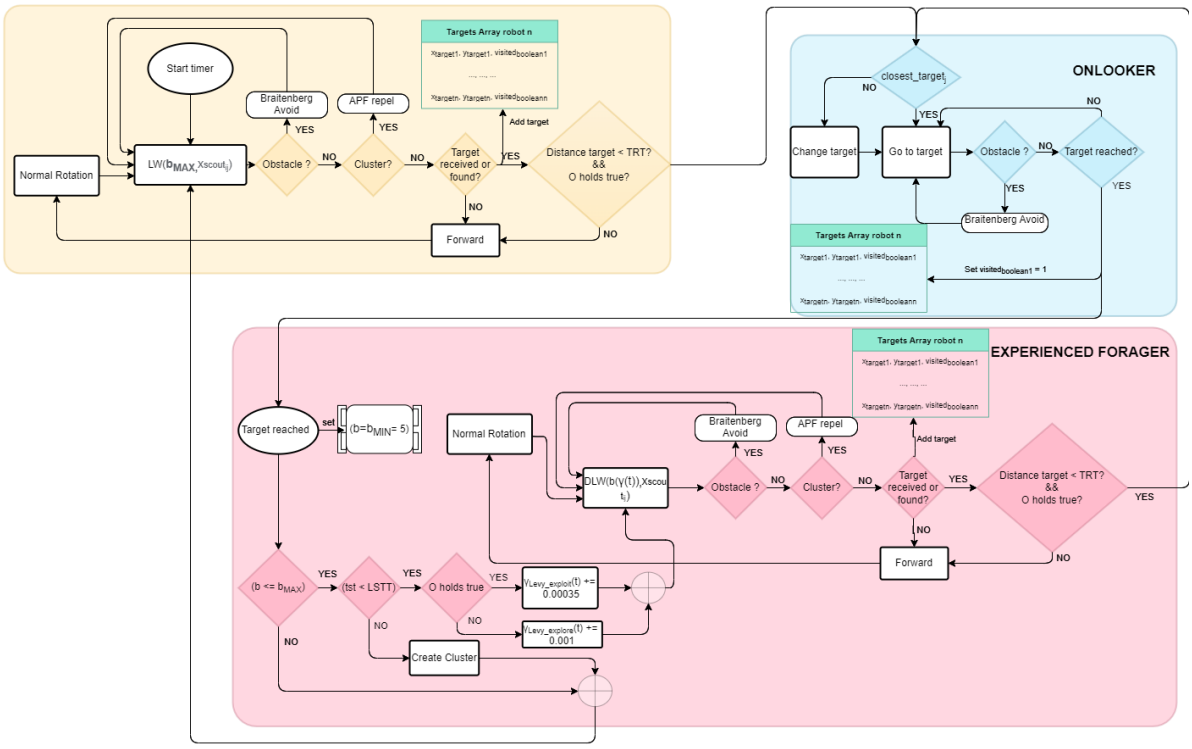


Figure 3.17: Flowchart algorithm with cluster recognition.

Whenever a scout has found a target to its liking, it decides to move towards that target using an artificial potential field. When the onlooker has reached the position of the target, the onlooker becomes an experienced forager and starts dynamic Lévy walking. This is a scaled-down version of the same Lévy walking method that scouts use. Experienced foragers make small steps, randomly searching around and inside the cluster for targets. The step size b increases back to the initial value that has been set for scouts, slowly favouring exploration again. However, when the first target has

been found, a timer is started. When this timer tst reaches the Local Search Time Threshold $\text{LSTT}_{\text{threshold}}$, foragers should no longer search in that area, as no other targets are found. Hence, the target is probably not inside a cluster, or there are no targets left to search for. Experienced foragers then become scouts again. When a new target is found before the threshold is reached, the timer resets. When $tst \geq \text{LSTT}_{\text{threshold}}$, experienced foragers save the average cluster position to a repel list. This is a list of average cluster positions where experienced foragers have been to. Also, the onlooker variable O determines the speed at which experienced foragers change back into scouts. This is controlled by changing the incremental speed of scale factor b. When the threshold signals exploration, it scales more quickly with $\gamma_{\text{Levy\_Explore}}$, than in the case where exploitation is required. When O signals exploitation, b scales with $\gamma_{\text{Levy\_Exploit}}$. The influential variables within MBSO are shown below. In this study, only the bold variables are optimized. However, optimizing other variables could increase the performance of the algorithm.

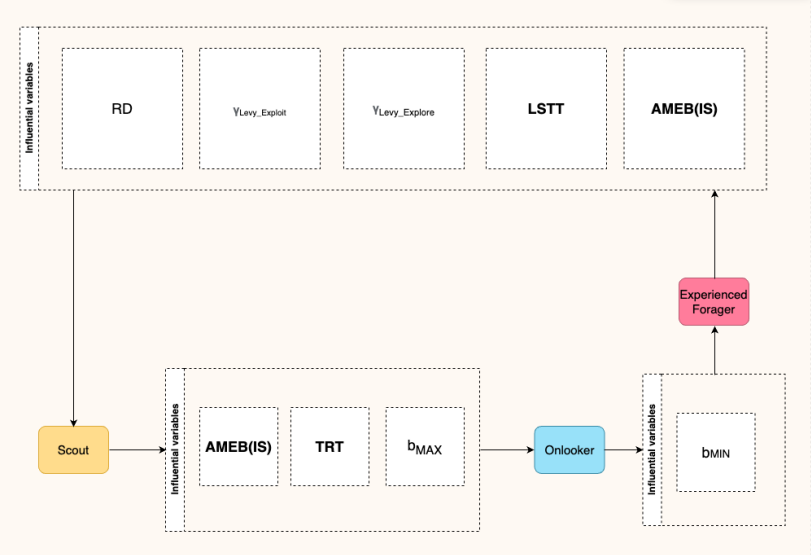


Figure 3.18: Influential variables in MBSO.

4

Methodology

Swarm systems with few robots that are studied in a limited time window are often hard to evaluate. The problem with defining swarm performance is that the performance of an individual member influences the swarm performance differently with a varying number of robots used [29]. The mentioned study also concludes that at the swarm level, adding more robots increases performance as long as the benefit of parallelizing the work is more significant than escaping the interference between robots. To be able to reason about the parameters of the swarming algorithm, the following research design is proposed.

4.1. Research design

The research aims to evaluate the parameters of Modified Bee Swarm Optimization (MBSO) within the defined environment in Sect.1.1. An experimental design is performed. This method involves manipulating independent variables and measuring the influence on the output parameters of the system. 8 Sets were constructed (table 4.1) with varying parameters evaluated over $N = 10, 15, 20$ robots. Every set consists of 30 trials. Every trial is a simulation of around 2 h and 15 min. The following characteristics of the system are considered: performance and redundancy, robustness, scalability and stability. These are evaluated by looking at the target arrays, received by the robots. In addition to evaluating the overall performance of MBSO, the individual characteristics of MBSO for varying N are also evaluated.

Some considerations are taken from the concluding remarks section in the literature review (Sect. 2) and are also evaluated for MBSO. These are the following:

1. Do the physical range limitations improve the performance of the robots, due to robots being forced to explore?
2. Does the performance of the algorithm scale linearly when adding more robots?
3. Does migration between sub-swarms increase optimality?

The first item can be answered by analyzing the Target Recognition Threshold (TRT), which is a physical range limitation between robots and targets. Another way to analyze this is to look at the Emitter Range (ER), which is the physical range limitation between robots. Both can influence the performance of the robots. The second question can be answered by looking at the scalability of the system. The last sub-question can be answered by measuring the influence of O, or in this case IS, on the optimality of the swarm. O is responsible for capping the maximum number of robots in a sub-swarm, and is, therefore, immediately responsible for inter-swarm migration. Optimality is interpreted as any type of performance gain in the system.

4.1.1. Overall performance

The performance of the algorithm can not simply be taken as the average percentage of visited targets per set. One of the main reasons is that, with increasing numbers of robots, the redundancy of visits for the same targets increases, essentially decreasing target-finding efficiency. To solve this, another way to quantify performance is introduced. This is called redundancy and is calculated by analyzing how target-finding performance is spread out between robots. In particular: ”How often are targets visited repeatedly?”. If some targets are visited several times by multiple robots, there is redundancy in the system, and the system could potentially get the same result with fewer robots. Target searching redundancy should, therefore, also be taken into account for determining the overall performance measure. The redundancy rating provides valuable insights into the efficiency of target visiting in a given scenario. Redundancy is calculated as follows:

$$\text{redundancy rating} = \frac{\text{total redundancy count}}{\text{number of unique targets}} \tag{4.1}$$

where ’total redundancy count’ is a counter that increments every time a target is visited by a robot, see Sec. 3.5. A lower redundancy rating indicates more efficient exploration with fewer redundant visits of the same target. This can also be correlated to improved energy efficiency, and increased productivity. The final performance measure is modelled as the average number of targets found, divided by the average redundancy rating over a set. The performance of the set is always averaged over 30 trials.

4.1.2. Robustness analysis

The robustness of the system is quantified by looking into the performance change of the system when perturbations are present. The simulated environment has 5 targets spread throughout the environment that do not adhere to any cluster. These targets are seen as a perturbation. There are a maximum of 5 ’perturbations’ that interfere with the normal cluster searching behaviour. It is expected that finding such a target lowers the overall performance when robots locally search for a long time around these perturbations.

4.1.3. Stability analysis

The standard deviation or ”stability” of MBSO over the target finding performance over 30 trials is measured. A lower standard deviation indicates that the system consistently performs close to the mean value, implying greater stability. Conversely, higher standard deviations suggest volatile and unpredictable behaviour, often resulting from an imbalance of exploration and exploitation.

4.1.4. Scalability analysis

The scalability of MBSO is gauged by examining the performance change when the number of robots increases. One of the sub-questions to be answered is whether the performance of the algorithm scales linearly when more robots are added.

4.2. Data collection

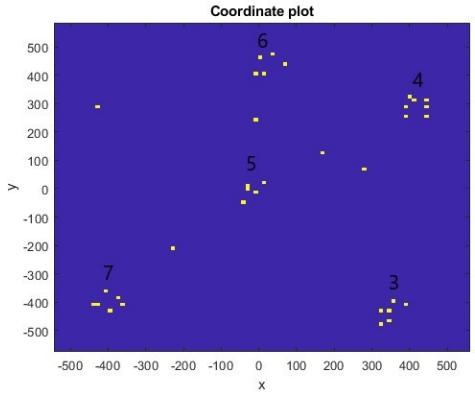
Simulations are performed on a desktop computer with an AMD Ryzen 7 37000X 8-Core processor with 16.0 GB of RAM and an NVIDIA GTX 1060 6GB graphics card. Every time a robot reaches the time limit of $2\,\text{h}$ and $15\,\text{min}$, or it runs out of battery, it saves its target array. When all robots have finished saving their targets array, the simulation resets. Data is collected by running the simulation 30 consecutive times. Robots are initiated randomly in the field by setting the initial position parameters of every robot using a supervisor script. The actual supervisor script can be found in the appendix appendix B.7. The data is saved in .txt files with time stamps. A file could for example be called: ’*targets_data_20230621_222856.txt*’, and is shortly visualized below. These files are filled with targets and information regarding whether the current robot has visited or received the target, as explained in Sect. 3.3.

```
targets.txt
[-5,410,0
 ...
 0,0,0]
```

4.3. Experimental setup

This section is intended to summarise the parameters used in this simulation. The performance of the swarm is tested within the location and simulation framework defined earlier in Sect. 1.1. The average step size of the robots is 175 sec with $b_{MAX} = 33$, equal to around 311 m, and is combined with a normal generator that generates random rotations between $[0, 2\pi]$. More about the step generator can be found in Sect. 3.6. Robots use MBSO (Sec. 3.2), which is a modified version of BSO (Sect. 3.2), that encapsulates local communication (Sect. 3.3).

General simulation parameters can be found in 4.1b, and robot parameters can be found in 4.2b. The time step used in the simulation is equal to 64 ms. The simulation time is based on the battery of the robot, which lasts around 2 h and 15 min tops. There are 5 settlements in the simulation that have a total of 29 clustered targets divided somewhat evenly. These targets are spaced between each other averaging 60 m between targets. The cluster centres and locations are also tabulated in figure 4.1b. Additionally, there are 5 singular targets added in the field, these are seen as perturbations in the system. The coordinates given in the table are relative to the centre of the field, which is at position $\{x, y\} = \{0, 0\}$.



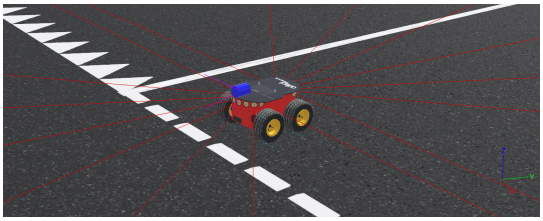
(a) Simulation of size 1200 x 1200 in Webots [68] with targets.

Global parameters	Amount	Unit
Time step	64	[ms]
Simulation time	2:15	[hours]
Number of targets	34	[-]
Number of robots	10,15,20	[-]
Size of field	1200 x 1200	$[m^2]$
Cluster 6 - Targets	5	targets
Cluster 6 - Centre	[25,440]	[x,y]
Cluster 4 - Targets	7	targets
Cluster 4 - Centre	[411,290]	[x,y]
Cluster 5 - Targets	5	targets
Cluster 5 - Centre	[-20,0]	[x,y]
Cluster 7 - Targets	6	targets
Cluster 7 - Centre	[-400,400]	[x,y]
Cluster 3 - Targets	6	targets
Cluster 3 - Centre	[-355, 430]	[x,y]

(b) Global simulation parameters.

Figure 4.1: Simulated map and real map of Huari alongside simulation parameters.

The robot used for simulating the swarm is the Pioneer Adept A-3. The max speed of the robot is 6.4 km h^{-1} and the robot has 3 batteries, each with 7.2 A h capacity. The robot weighs 12 kg and can carry an additional payload up till 12 kg. It uses skid steering to rotate and thus has a turning radius of 0. The rotational speed is $140\,°/s$. Skid steering makes it easy to implement the normal rotations of the robot during Lévy walking. The Adept A-3 is also able to traverse steps with size 10 cm, gaps of size 15 cm and a slope of $35°$. It can traverse terrains like sand and dirt, thus suitable for rugged outdoor use. The Adept is equipped with a 20-MHz Siemens 88C166-based micro-chip with a high-performance 16-bit CPU and many I/O capabilities. It also has 16 sonar sensors, which are split up into two rows of 8 forward-sensing and 8 backwards- sensing sonar sensors. In addition, in this simulation, the robots are also equipped with a receiver, emitter, inertial unit, GNSS, and camera with object recognition. More about how camera recognition and obstacle avoidance are set up can be found in the appendix in Sect. B.



(a) Modified Adept Pioneer 3-AT.

Robot parameters	Amount	Unit
Max speed	6.4	[km/h]
Recognition distance	15	[m]
Max battery	933120	[J]
Amount of distance sensors	16	[-]
b_{MIN}	5	[-]
b_{MAX}	33	[-]
Emitter range	100	[m]
Average step size	311	[m]
Repulsion distance	75	[m]
$\gamma_{explore}$	0.00035	[iteration]
$\gamma_{exploit}$	0.003	[iteration]
Average step size	311	[m]

(b) Simulation parameters for the robot.

Figure 4.2: Adept Pioneer 3-AT alongside robot simulation parameters.

4.4. Variables and measurements

Control variables that are varied to determine the performance of the algorithm are the following:

1. TRT: Target Recognition Threshold. This is the maximum distance for a robot to determine if it will take the target into account. If the closest unvisited target in the target array is further than TRT, the robot will not consider the target viable. By varying TRT, one can assess the influence of robots going for distant targets and determine if they should rather stay exploring.
2. IS: Interactions state. This determines the sensitivity of the onlooker variable O. It sets the maximum number of robots that may search a potentially interesting area in parallel. IS also determines whether scale factor b increases with increments of $\gamma_{explore}$ or $\gamma_{exploit}$. By varying IS, one can analyze how the maximum number of robots in a sub-cluster and their interactions affect the exploration and exploitation of the overall algorithm.
3. $LSTT$: Local Search Time Threshold. LSTT represents the threshold for how long an area of interest can be exploited. By varying LSTT, the influence of different time thresholds for local searching can be examined.
4. N: The number of robots used in the simulations. By varying N, the impact of different densities of robots on the overall performance of the algorithm can be investigated. It allows for analyzing the scalability and effectiveness of the algorithm in different conditions.

Varying these variables results in the sets tabulated in table 4.1. These sets are examined for $N = 10, 15, 20$, as swarm performance is expected to be different for a different number of robots.

	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6	Set 7	Set 8
TRT	150	150	150	150	300	300	300	300
IS	1	2	1	2	1	2	1	2
LSTT	15	15	30	30	15	15	30	30

Table 4.1: All combinations of variables grouped in sets. All sets are examined over N = 10,15,20.

The way the measurements are structured are shown in figure 4.3.

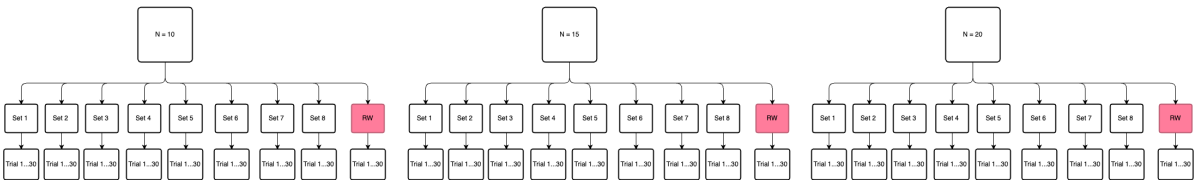


Figure 4.3: Structure of simulations.

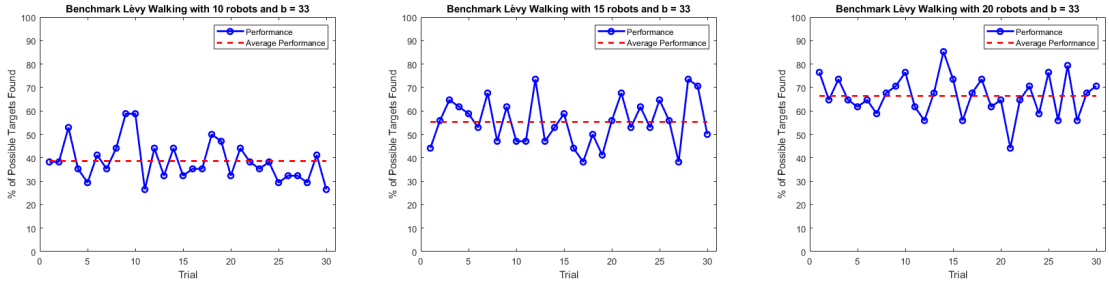
4.5. Data analysis

The presence of outliers provides insight into the algorithm's properties, and are, therefore, not removed. To assess statistical significance, a linear regression analysis is used. Linear regression allows the investigation of the relationship between variables. Statistical significance is determined by assessing the p-values associated with the tested control variables. In this study, a significance level of $p < 0.05$ is used. Any value below this threshold shows a relationship between variables. For every regression, the associated ANOVA table is given. An ANOVA table is a statistical tool used to assess the significance of group differences. The SumSq variable in the ANOVA table represents the sum of squares. This is the variability within each variable and indicates how much the data deviates from the mean. The larger the sum of squares, the more variability. DF represents the degrees of freedom associated with each variable. If there are two groups that are being compared, DF is equal to 1. MeanSq corresponds with the mean square. This is an estimate of the variance within each variable and is calculated by dividing the sum of squares by the degrees of freedom. The F-value measures the ratio of variability between groups to the variability within the group. A high F-value indicates large differences between group means, which corresponds to stronger statistical significance between the groups. Lastly, the p-value shows the probability of observing the data if the hypothesis is true.

4.6. Random walking as benchmark

To benchmark the relative performance of the proposed MBSO model, it is compared with the case where all robots solely use random walking to explore the environment. This is done by setting $b \leftarrow \mathrm{b_{MAX}}$ with $\mathrm{b_{MAX}} = 33$ resulting in an average step size of 311 m.

The simulation is run with $N = 10, 15, 20$ robots that are placed randomly in the field. The simulation is reset when all batteries are empty, or the limit of 2 h and 15 min is reached. The percentage of targets found with just random walking with a varying number of robots $N = 10, 15, 20$, for a duration of 2 h and 15 min are 38.63 %, 55.29 % and 66.37 % respectively.



(a) 10 robots Lévy Walking the environment results in an average of 38.63 % targets found over 30 trials.

(b) 15 robots Lévy Walking the environment results in an average of 55.29 % targets found over 30 trials.

(c) 20 robots Lévy Walking the environment results in an average of 66.37 % targets found over 30 trials.

Figure 4.4: Random Lévy Walking benchmark with a varying number of robots and scale parameter $b = \gamma_{\mathrm{max}}$.

The percentage of targets found and standard deviations of robots performing random walks with $\mathrm{b} = b_{\mathrm{MAX}}$ are shown in Fig. 4.5. As the number of robots increases, the number of targets found also increases. Going from $N = 10$ to $N = 15$, increases the percentage of targets found with 16.64 %, where the increase from $N = 15$ to $N = 20$ is equal to 11.08 % more targets found.

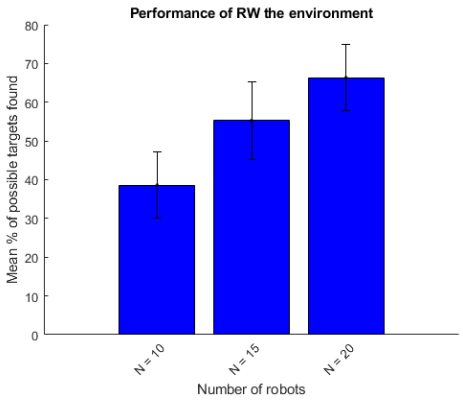


Figure 4.5: Benchmark of N=10,15,20 robots.

4.7. Methodology summary

Swarm systems with few robots are hard to evaluate in a limited time. Individual members influence the swarm performance differently with various numbers of robots [29]. To solve this, swarm characteristics should also be evaluated for different numbers of robots. Therefore, not only the total swarm performance is analyzed, but also the individual characteristics over fixed $N = 10, 15, 20$. From the literature review, the following considerations are also analyzed:

1. Do the physical range limitations improve the performance of the robots, due to robots being forced to explore?
2. Does the performance of the algorithm scale linearly when adding more robots?
3. Does migration between sub-swarms increase optimality?

The characteristics of the swarm are evaluated by looking at the performance, redundancy, robustness, stability and scalability of the swarm. The performance is the percentage of targets that the swarm has found. However, with increasing numbers of robots, targets are also visited more redundantly. Therefore, the overall performance is taken as the percentage of targets found divided by the redundancy rating. This should give a measure of how efficiently the targets are found, in addition to the number of targets found, creating a balanced performance measure. The scalability of the swarm is estimated by looking at the characteristics of the swarm, with increasing numbers of robots. The robustness of the swarm is measured by looking at the number of targets found, in comparison to the number of perturbations that are found. If many perturbations are found, the system should perform worse, as the robots would start searching these unimportant areas. Measuring how the swarm handles these perturbations will indicate the robustness of the system. The stability of the system is determined by looking at the standard deviation of the trials. Data is collected by target arrays of robots, containing received and found targets. The simulated area used for determining the characteristics of the swarm is an area where targets are spread out in clusters. Additionally, some targets are spread out as anomalies in the system, not belonging to any cluster. These targets should show the robustness of the system.

The analyzed variables of modified bee swarm optimization are the Target Recognition Threshold TRT, Onlooker variable O or Interaction State IS, Local Search Time Threshold $LSTT$ and the Number of Robots N. Changing these control variables should influence the swarm system in such a way that the characteristics of the swarm become apparent. A total of 8 sets with varying the mentioned control variables are created. Every set is tested over varying N, with $N = 10, 15, 20$. For every N, one random walk benchmark is done. Every set is averaged over 30 trials, where every trial is a maximum of 2 h and 15 min long. In total, there are $8 * 30 * 3 = 720$ trials for the 8 sets and $1 * 30 * 3 = 90$ sets for the random walk trials. Together, 810 trials, or 1822 h and 30 min worth of real-world time simulations are performed to gather the necessary data.

The statistical significance of the variables is assessed by looking at the associated ANOVA table. These are then compared with the case where robots only use random walking. Random walking with $N = 10, 15, 20$ robots could find 38.63 %, 55.29 % and 66.37 % of targets, respectively.

5 Results

In this chapter, the characteristics of the proposed modified bee swarm optimization algorithm are analyzed, and compared with the random walk method performed in Sect. 4.6. First, a comparison is made of the number of targets found between random walking and the average percentage over all sets, for varying N. In addition, the most optimal variables in all 8 sets for the corresponding N are also visualized. Then, the overall scalability is determined by looking at the performance increase, with increasing N, in Sect. 5.2. The variables that influence the performance and redundancy are then determined in Sect. 5.3. Robustness is analyzed in Sect.5.4. Stability is analyzed next, in Sect. 5.5. Finally, the performance is individually analyzed for $N = 10$, $N = 15$, and $N = 20$ because individual members influence the swarm performance differently with a varying number of robots [29].

5.1. Overall results

Variation of variables TRT, IS, and $LSTT$ provide 8 total sets which were tested with varying numbers of robots N, making a total of 24 variations. In Sect. 4.6, a benchmark statistic using Random Walking (RW) was provided. The average percentage of targets found (MBSO Average) over the 8 total sets and the most optimal values (MBSO Optimal) of these sets are compared with each other for varying N in Fig. 5.1. With RW in red, MBSO Average in green, and MBSO Optimal in blue. The highest performing sets in MBSO Optimal are set 4 for $N = 10$, set 3 for $N = 15$ and set 2 for $N = 20$. The associated variables with these sets can be found in table 4.1.

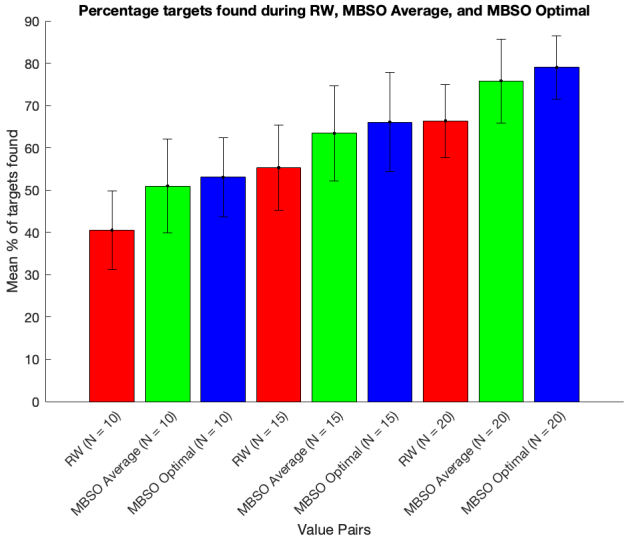


Figure 5.1: Percentage of targets found in RW (red) versus the combined average of targets found over all sets in MBSO (green) and the set with the highest percentage of targets found in MBSO (blue) plotted for $N = 10, 15, 20$.

5.2. Scalability

Scaling from $N = 10$ to $N = 15$, and to $N = 20$, using RW resulted in finding $16.66\,\%$ and $11.08\,\%$ more targets respectively. This suggests that increasing the number of robots from $N = 10$ to $N = 15$, even without using an algorithm, results in significant performance increases. However, there exists a non-linear recline in performance during scaling. Scaling the average values of MBSO from $N = 10$ to $N = 15$, and to $N = 20$ resulted in the algorithm finding $12.45\,\%$ and $12.40\,\%$ more targets. When optimized parameters for MBSO were used, scaling MBSO from $N = 10$ to $N = 15$, and to $N = 20$ resulted in finding $7.74\,\%$ and $13.22\,\%$ more targets, respectively. This signals a non-linear increase. This has been visualized in Fig. 5.2. It should be clear that the percentages visualized are the difference between the percentage of values found per N and not the percentual increases. For example, RW 10-15 is the difference between $N = 10$ with $38.63\,\%$ and $N = 15$ with $55.29\,\%$. And MBSO AVG10-15 is the difference between $N = 10$ with $50.97\,\%$ and $N = 15$ with $63.42\,\%$.

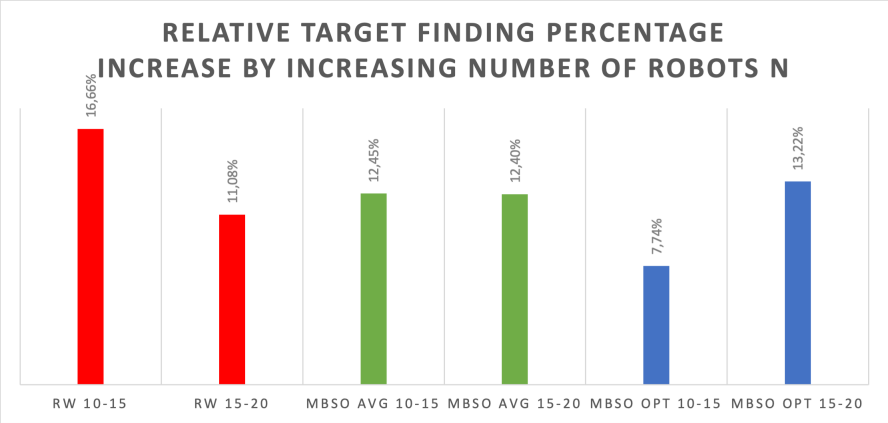


Figure 5.2: Increases of target finding percentages with increasing N.

Above, a figure of the relative increase in the percentage of targets found from scaling the number of robots was shown. In the following plot, the percentual difference between using RW and MBSO Average, and RW and MBSO Optimal are shown. This can be understood as: 'How much percentage is MSBO better than RW?'. The largest improvement was noticed by using MBSO with a small number of robots. This makes sense as there is more need for communication between robots to optimize cluster locations. It was shown that choosing the right parameters for a specific number of robots increases the performance radically.

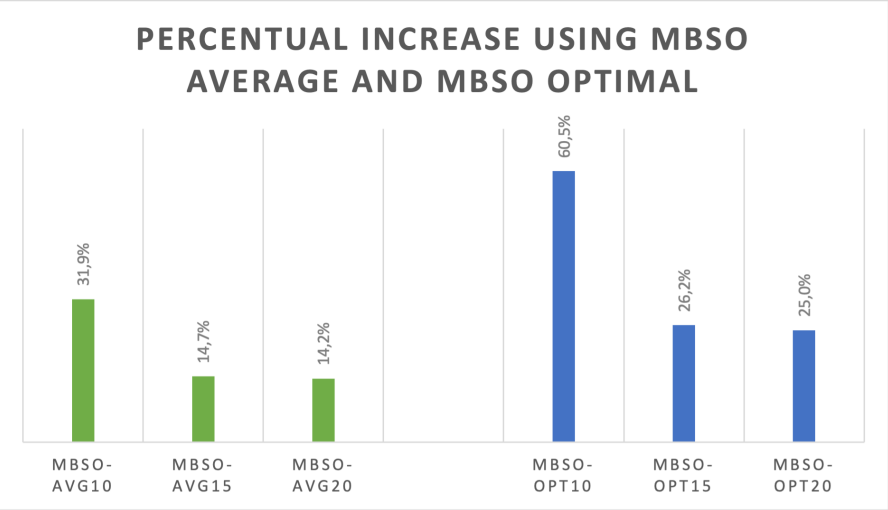
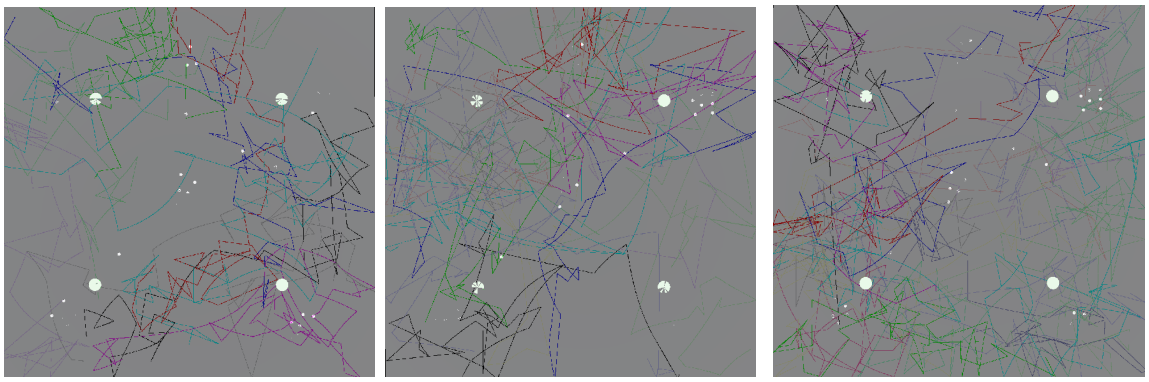


Figure 5.3: Percentual increase of target finding capabilities between RW and MBSO Average, and RW and MBSO Optimized for different N.

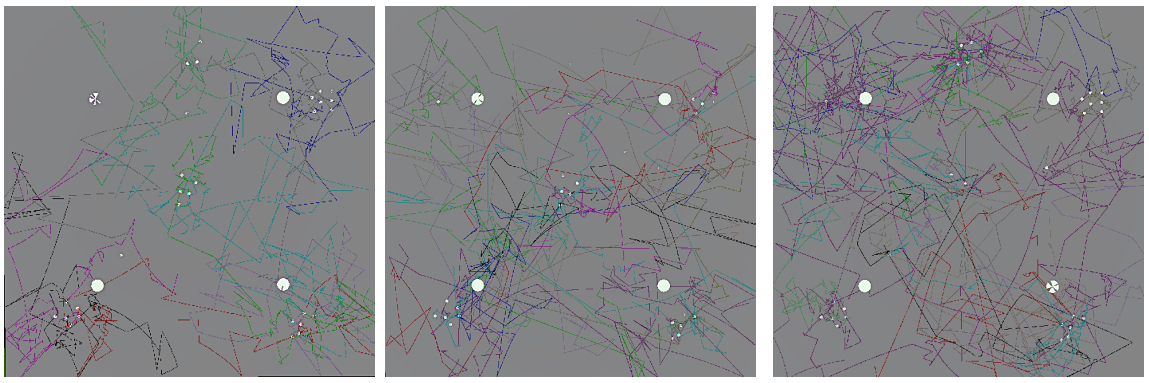
A visualization of one trial using RW, for varying numbers of robots N is shown in Fig. 5.4. The same is shown for MBSO, with parameters from set 1, in Fig. 5.5. When the number of robots increases, the covered area in RW also increases. The wall avoidance technique introduced in Fig. 3.16 can also be noticed in these plots.



(a) RW with $N = 10$ robots. **(b)** RW with $N = 15$ robots. **(c)** RW with $N = 20$ robots.

Figure 5.4: RW with varying number of robots $N = 10, 15, 20$ in the simulated environment for 2 h and 15 min. The small white circles represent targets and the different coloured lines are robots.

The following figures show visualisations of MBSO with parameters from set 1, which can be found in table 4.1. From the following plots, it is clear that MBSO is able to guide robots efficiently to target locations. However, with increasing numbers of robots, the redundancy of target visits also grows larger.



(a) An above average trial of set 1 with $N = 10$ robots. **(b)** An above average trial of set 1 with $N = 15$ robots. **(c)** An above average trial of set 1 with $N = 20$ robots.

Figure 5.5: Above average MBSO performance with variables from set 1 with a varying number of robots $N = 10, 15, 20$ in the simulated environment for 2 h and 15 min. The small white circles represent targets and the different colored lines are robots.

5.3. Performance and redundancy

The percentage of targets found and the number of redundant target visits should both be taken as indicators for the overall performance of MBSO. The overall performance is, therefore, a combination of both, as explained in Ch. 4. Using linear regression, the overall performance is analyzed as a combination of weighted control variables $N, TRT, IS, LSTT$. The following ANOVA table shows the significance of these variables for the performance of the algorithm. As a reminder, the performance is equal to the percentage of targets found, divided by the redundancy rating.

Variable	SumSq	DF	MeanSq	F	pValue
N	1733.20	1	1733.20	459.70	8.96e-15
TRT	20.84	1	20.84	5.53	2.97e-2
IS	0.81	1	0.81	0.21	0.65
LSTT	11.98	1	11.98	3.18	9.06e-2

Table 5.1: ANOVA table of the performance for set 1-8 with $N = 10, 15, 20$.

N shows a highly significant effect on the target finding performance. The large F-value $F = 459.70$ and extremely small p-value $p = 8.96e - 15$ support this motion. TRT also shows to be significant, as $p = 2.97e - 2$ with $F = 5.53$. On the contrary, there is not enough evidence to prove that IS and $LSTT$ have any effect on the performance of the swarm system. The linear regression model is, therefore, changed to only incorporate N and TRT. These non-significant variables are removed from the regression model, and the linear regression model is simplified to:

$$\text{Performance} = \beta_0 + \beta_1 \cdot N + \beta_2 * TRT \tag{5.1}$$

with $\beta_0 = 40.67$, $\beta_1 = 2.08$ and $\beta_2 = -1.24e - 2$. The linear model for the performance of the algorithm is visualized in Fig. 5.6.

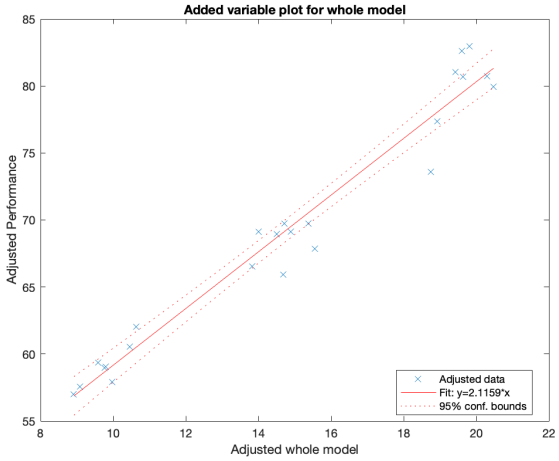


Figure 5.6: Linear regression model based on N and TRT of the performance of MBSO.

Increasing TRT has a negative effect on the overall performance of MBSO. On the contrary, increasing N drastically increases performance. Lastly, changing IS from 1 to 2 (a maximum of 2 or 3 robots per sub-swarm) does not show to have a significant effect on the performance of the algorithm, however, it is expected that removing this maximum size limitation would decrease performance, as many robots would search the same area.

The number of redundant target visits seems to increase with an increasing number of robots, as seen in Fig. 5.5a, Fig. 5.5b and Fig. 5.5c. A polynomial fit between these two is performed in Fig. 5.7. It shows that increasing from $N = 10$ to $N = 15$ robots gives the largest increase in redundancy. The polynomial fit is given as follows:

$$\text{Redundancy} = \beta_0 + \beta_1 * N - \beta_2 * N^2 \tag{5.2}$$

with $\beta_0 = 0.61$,$\beta_1 = 3.38e - 2$ and $\beta_2 = -8.41e - 4$.

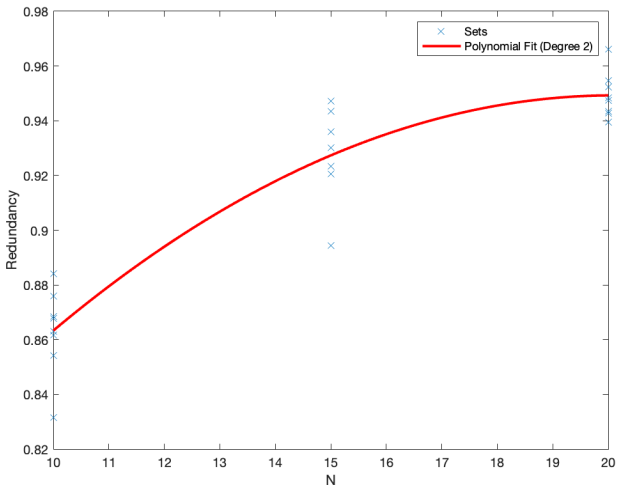


Figure 5.7: Redundancy of target visits plotted against an increasing number of robots.

5.4. Robustness

The robustness of the swarm is analyzed by looking at the target finding percentage, relative to the number of times the perturbations (5 outliers that are placed in the area) have been visited. First, the number of outlier visits averaged over 30 trials per set is calculated. This will result in a data table with the percentage of outlier visits:

Outlier coordinates	**Visit Percentage**
(174, 131, 1)	10.67%
(280, 80, 1)	7.00%
(-223, -202, 1)	13.33%
(-8, 252, 1)	9.33%
(-426, 290, 1)	4.67%

Table 5.2: Average outlier visit percentages for a random set.

Calculating the mean percentage over this table, an average of 9% is computed from this example. To get the final robustness of the system, the target finding percentage is subtracted from the number of outlier visits. This measure indicates how robots deal with these perturbations. The following table shown in Table 5.3, shows the statistical importance of the control variables for the robustness of the system.

Variable	SumSq	DF	MeanSq	F	pValue
N	2059	1	2059	350.67	1.0498e-13
TRT	120.49	1	120.49	20.52	2.29e-4
IS	8.08	1	8.08	1.38	0.26
LSTT	35.46	1	35.46	6.04	2.38e-2

Table 5.3: ANOVA table of the robustness for set 1-8 with $N = 10, 15, 20$.

It seems that N and TRT and $LSTT$ are all influencing the robustness of the algorithm. Also, IS is not significant. The regression model changes to:

$$\text{Robustness} = \beta_0 + \beta_1 \cdot N + \beta_2 \cdot TRT + \beta_3 \cdot LSTT \tag{5.3}$$

with $\beta_0 = 20.23$ and $\beta_1 = 2.27$, $\beta_2 = -0.03$ and $\beta_3 = 0.16$. Increasing N and $LSTT$ increases the robustness, whereas increasing TRT lowers it. The linear fit to calculate the robustness is visualized in Fig. 5.8.

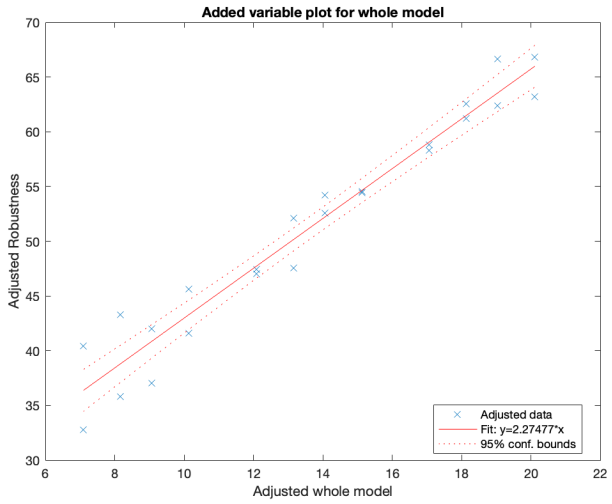


Figure 5.8: Linear regression model for correlating robustness against N, $LSTT$ and TRT.

5.5. Stability

The stability of the system is determined by examining the standard deviation σ of the different sets with varying N. Using linear regression, with a chosen significance level of $p < 0.05$, the following table shows the significance of the variables:

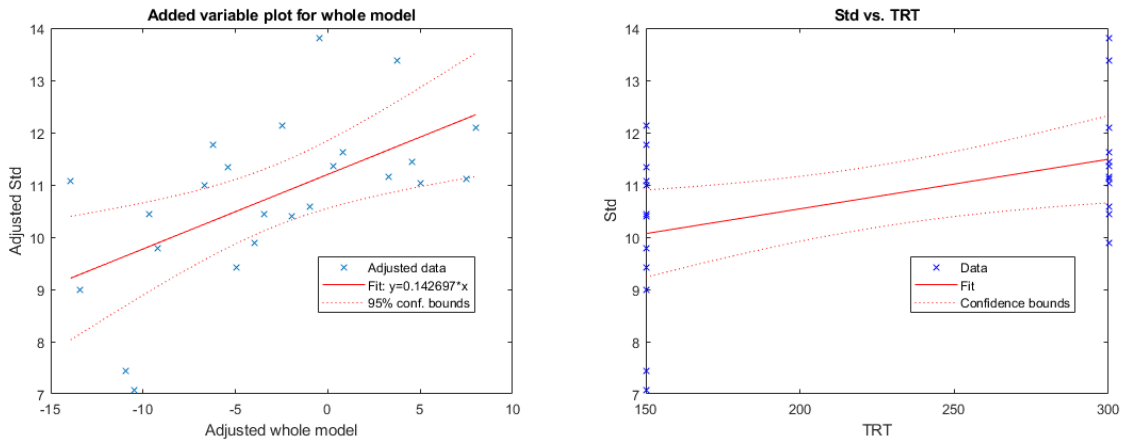
Factor	SumSq	DF	MeanSq	F	pValue
N	5.86	1	5.86	3.11	9.38e-2
TRT	12.15	1	12.15	6.45	2e-2
IS	2.89e-2	1	2.89e-2	1.53e-2	0.90
LSTT	1.08	1	1.08	0.57	0.46

Table 5.4: ANOVA table of the stability for set 1-8 with $N = 10, 15, 20$.

It seems that N, IS and $LSTT$ are not significant for the determination of the stability of the system, as $p \not< 0.05$. Only TRT has a statistical influence on the standard deviation, and thus stability, of the system. This is true because $p = 0.02$. The standard deviation σ can be correlated with the following formula:

$$\sigma = \beta_0 + \beta_1 \cdot TRT \tag{5.4}$$

with $\beta_0 = 8.65$ and $\beta_1 = 9.49e - 3$. Please take note that the stability of the algorithm increases when σ decreases. The linear regression model used to calculate the stability is then visualized in Fig. 5.9a with all non-statistical variables included, and with only TRT in Fig. 5.9b. Based on this model, increasing TRT increases the standard deviation, lowering the stability of the algorithm. An increase of $TRT = 150$ to $TRT = 300$ will raise the standard deviation with $\sigma = 1.42$. However, the model does not encapsulate the whole data sufficiently, as there are many outliers. It should be expected that increasing N, would increase the stability of the algorithm, however, no proof was found to support this motion. It seems that the variability between trials, no matter the number of robots, or control parameters, is large enough to disguise the differences that result from changing the control parameters.



(a) Stability of MBSO based on N, $LSTT$ and TRT.

(b) Stability of MBSO based on TRT.

Figure 5.9: Linear regression model for correlating stability against TRT.

5.6. Performance of MBSO with $N = 10$ robots

To study the influence of varying control variables $TRT, IS, LSTT$ in MBSO, 8 different sets mentioned in table 4.1 are tested. Every set (bar) visualized in Fig. 5.11 is simulated 30 times, with a simulation time of around 2 h and 15 min per trial. The average percentage of targets found in MBSO with $N = 10$ robots averaged over all sets is 50.97 %, while the average standard deviation with this number of robots is at $\sigma = 11.12$. In comparison with the random walk method, the average percentage of targets found in MBSO is 31.9 % higher. With optimal parameters, this difference becomes 60.5 %, as was shown in Fig. 5.3. The stability of the algorithm changes negligibly between sets. This largest difference between the percentage of targets found is between sets 3 and 5, with a difference of 3.92 %. However, the highest value in terms of performance is set 4.

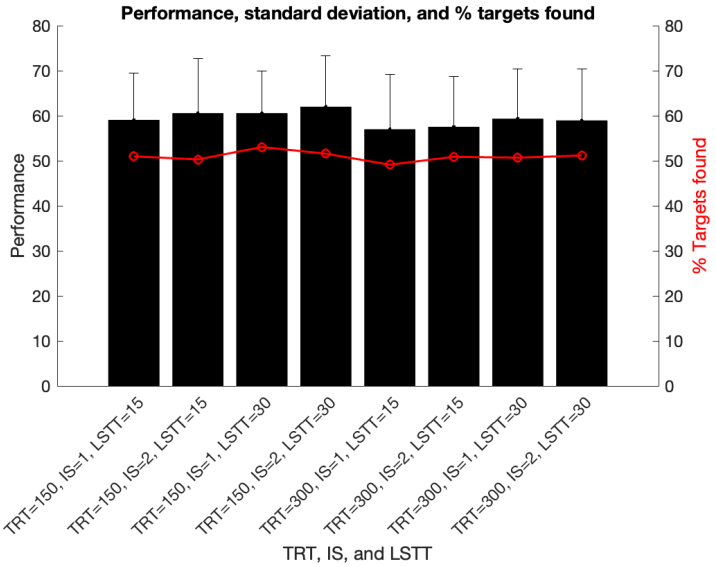


Figure 5.10: Mean performance, standard deviation and % targets found with N = 10.

Table 5.5 is used to generate a linearized model. TRT and $LSTT$ show to be significant for the performance of the algorithm. IS is not significant as $p \not< 0.05$, and is removed. The formula for determining the highest performer with $N = 10$ robots is as follows:

$$\text{Performance} = \beta_0 + \beta_1 \cdot TRT + \beta_2 * LSTT \tag{5.5}$$

with $\beta_0 = 58.06$, $\beta_1 = 0.01$ and $\beta_2 = 0.16e - 2$.

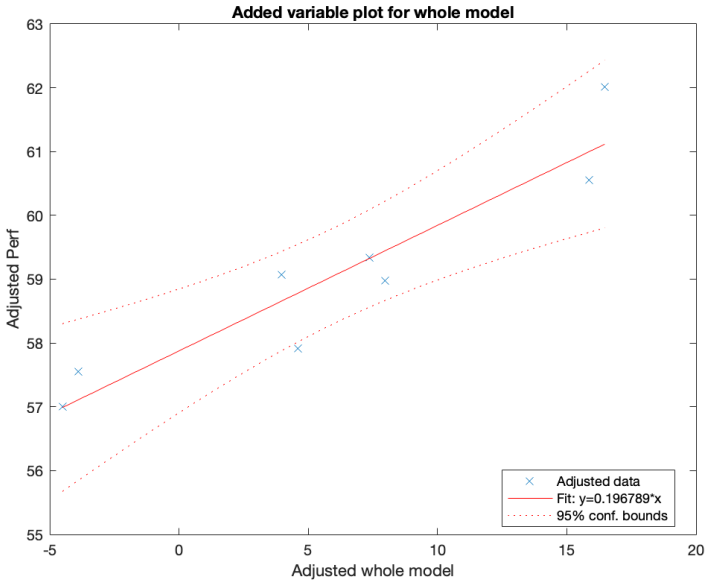
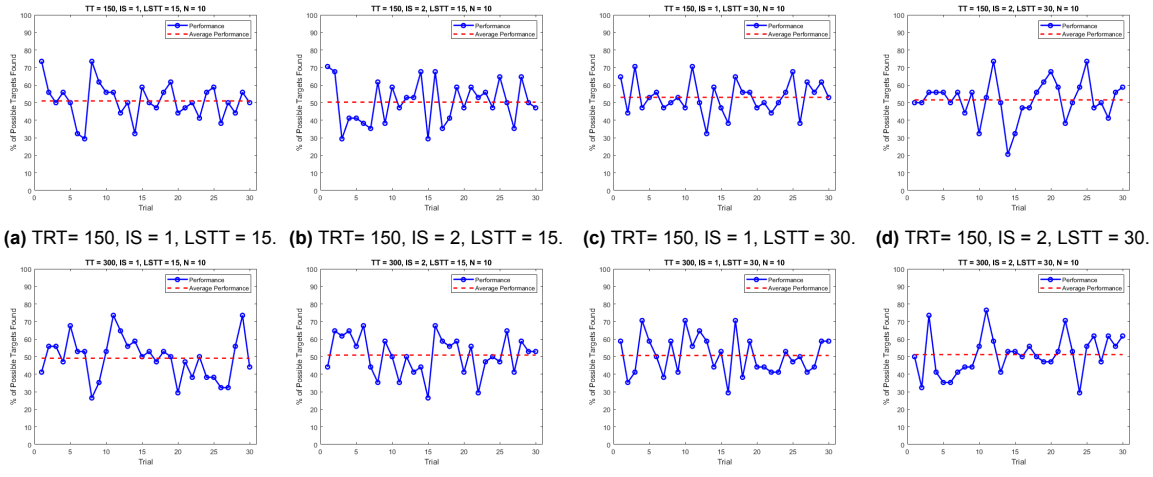


Figure 5.11: Linear regression of the performance of MBSO with control parameters $TRT, LSTT$.

Variable	SumSq	DF	MeanSq	F	pValue
TRT	5.59	1	5.59	9.55	3.66e-2
IS	2.87e-2	1	2.87e-2	4.9e-2	8.36e-1
LSTT	10.92	1	10.92	18.65	1.25e-2

Table 5.5: ANOVA Table to determine the significance of the control variables $TRT, IS, LSTT$ with $N = 10$.

The 30 trials for each of the 8 sets are simulated with $N = 10$ robots are displayed in Fig. 5.12. The variability between trails is large, as there are only $N = 10$ robots. Therefore, it is hard to visually distinguish different sets.

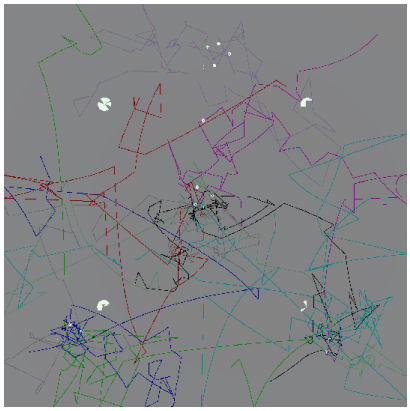


(a) TRT= 150, IS = 1, LSTT = 15. **(b)** TRT= 150, IS = 2, LSTT = 15. **(c)** TRT= 150, IS = 1, LSTT = 30. **(d)** TRT= 150, IS = 2, LSTT = 30.

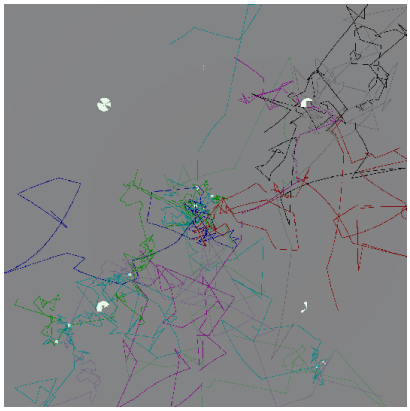
(e) TRT= 300, IS = 1, LSTT = 15. **(f)** TRT= 300, IS = 2, LSTT = 15. **(g)** TRT= 300, IS = 1, LSTT = 30. **(h)** TRT= 300, IS = 2, LSTT = 30.

Figure 5.12: Visualization of the 30 performed trials per set with $N = 10$.

A visualisation of the highest (Fig. 5.13a), and lowest performer (Fig. 5.13b), taken from a random trial of the corresponding set, and are visualized next in Fig. 5.13. The variability between trials is quite large, and the figures are just for visualization purposes.



(a) Visualization of MBSO set 4 with $TRT = 150$, $IS = 2$, $LSTT = 30$ and $N = 10$.



(b) Visualization of MBSO set 5 with $TRT = 300$, $IS = 1$, $LSTT = 15$ and $N = 10$.

Figure 5.13: Visualization of random trials of the best and worst set within $N = 10$.

5.7. Performance of MBSO with $N = 15$ robots

To study the influence of varying control variables $TRT, IS, LSTT$ in MBSO, 8 different sets mentioned in table 4.1 are tested. Every set (bar) visualized in Fig. 5.11 is simulated 30 times, with a simulation time of around 2 h and 15 min per trial. The average percentage of targets found in MBSO with $N = 15$ robots averaged over all sets is 63.42 %, while the average standard deviation with this number of robots is $\sigma = 11.13$. In comparison with the random walk method, the average percentage of targets found in MBSO is 14.7 % higher. With optimal parameters, this difference becomes 26.2 %, as is shown in Fig. 5.3. The stability of the algorithm changes negligibly between sets. The largest difference between the percentage of targets found is seen between sets 1 and 5, with a difference of 6.57 %. However, in terms of performance, there seems to be little difference between sets.

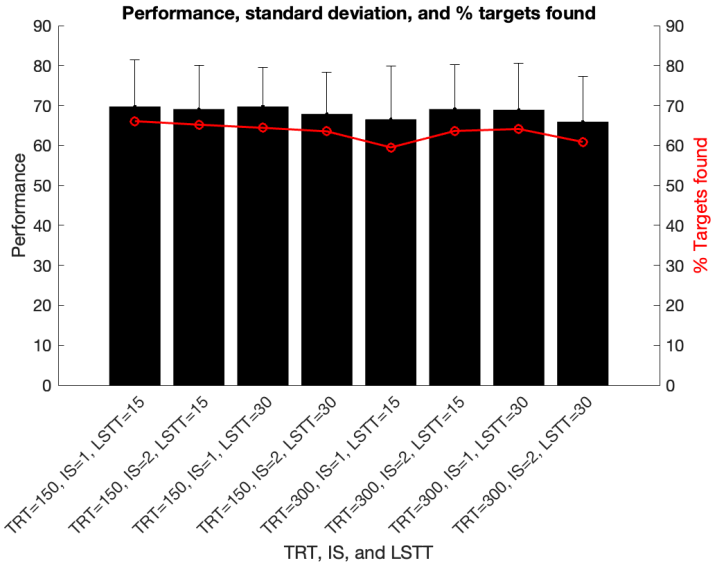


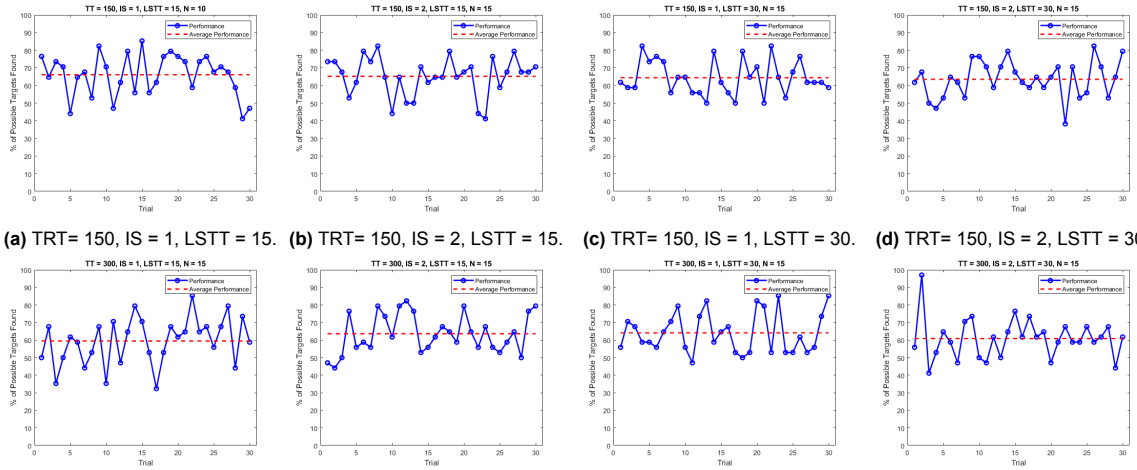
Figure 5.14: Mean performance, standard deviation and % targets found with $N = 15$.

Table 5.6 is used to generate a linearized model. Unfortunately, no significant variable was found that influences the performance of the algorithm. This suggests the algorithm performance is reasonably robust when $N = 15$ robots are used.

Variable	SumSq	DF	MeanSq	F	pValue
TRT	4.49	1	4.49	2.06	2.25e-1
IS	1.09	1	1.09	5.00e-1	5.18e-1
LSTT	5.18e-1	1	5.18e-1	2.37e-1	6.51e-1

Table 5.6: ANOVA Table to determine the significance of control variables $TRT, IS, LSTT$ with $N = 15$.

The 30 trials for each of the 8 sets are simulated with $N = 15$ and displayed in Fig. 5.15. Most sets do not have outliers under $50\,\%$. However, set 5, shown in Fig. 5.15e has extremely low outliers, very often. This is also the lowest-performing set. Set 7 shows to have extremely high outliers, very often. This is one of the higher-performing sets. The difference is that in set 7, robots keep searching longer when a target has been found. This may suggest that, because of the high number of robots, it is more accepted to stay longer at a certain location. The ANOVA table, unfortunately, does not support this hypothesis.

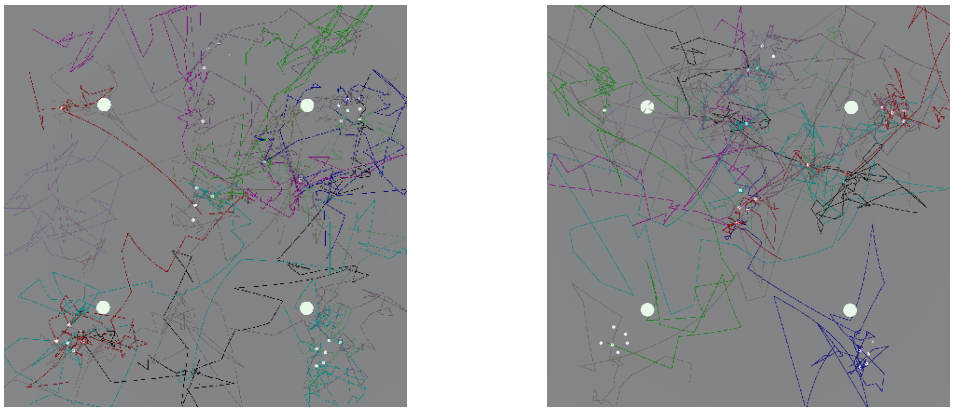


(a) TRT= 150, IS = 1, LSTT = 15. **(b)** TRT= 150, IS = 2, LSTT = 15. **(c)** TRT= 150, IS = 1, LSTT = 30. **(d)** TRT= 150, IS = 2, LSTT = 30.

(e) TRT= 300, IS = 1, LSTT = 15. **(f)** TRT= 300, IS = 2, LSTT = 15. **(g)** TRT= 300, IS = 1, LSTT = 30. **(h)** TRT= 300, IS = 2, LSTT = 30.

Figure 5.15: Visualization of the 30 performed trials per set with $N = 15$.

A visualisation of the highest (Fig. 5.16a), and lowest performer (Fig. 5.16b), taken from a random trial of the corresponding set, and are visualized next in Fig. 5.16. The variability between trials is quite large, and the figures are just for visualization purposes.



(a) Visualization of MBSO set 1 with $TRT = 150$, $IS = 1$, $LSTT = 15$ and $N = 15$.

(b) Visualization of MBSO set 5 with $TRT = 300$, $IS = 1$, $LSTT = 15$ and $N = 15$.

Figure 5.16: Visualization of random trials of the best and worst set within $N = 15$.

5.8. Performance of MBSO with $N = 20$ robots

To study the influence of varying control variables $TRT, IS, LSTT$ in MBSO, 8 different sets mentioned in table 4.1 are tested. Every set (bar) visualized in Fig. 5.11 is simulated 30 times, with a simulation time of around 2 h and 15 min per trial. The average percentage of targets found in MBSO with $N = 20$ robots averaged over all sets is 75.83 %, while the average standard deviation with this number of robots is at $\sigma = 9.91$. In comparison with the random walk method, the average percentage of targets found in MBSO is 14.2 % higher. With optimal parameters, this difference becomes 25 %, as shown in Fig. 5.3. The stability of the algorithm changes negligibly between sets. This largest difference between the percentage of targets found is between sets 2 and 6, with a difference of 9.22 %. The highest values in terms of performance are found in sets 2 and 8.

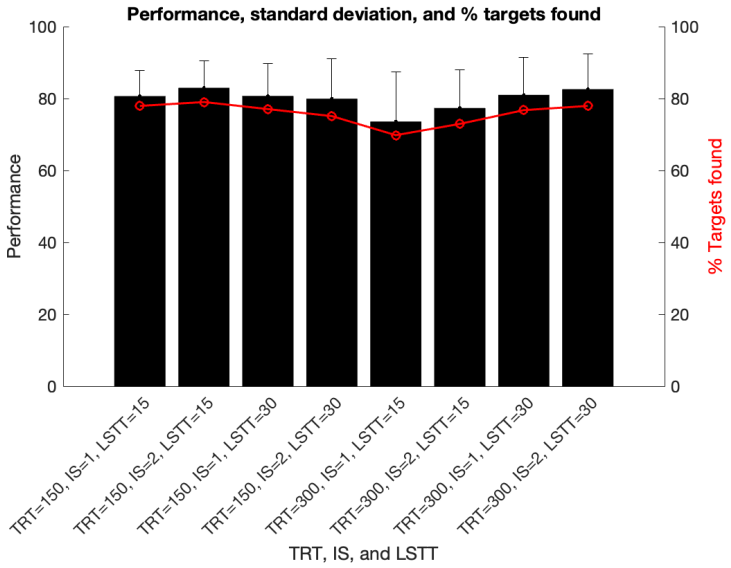


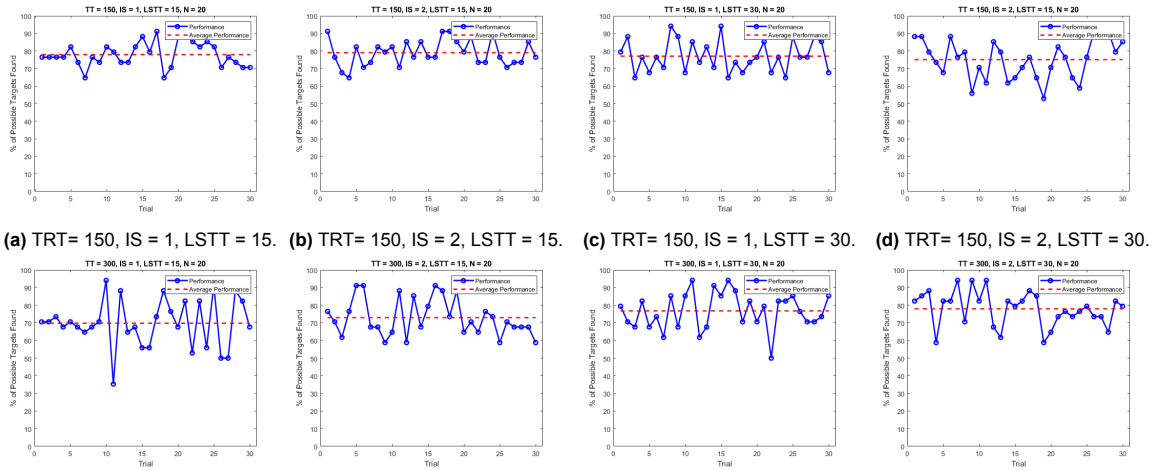
Figure 5.17: Mean performance, standard deviation and % targets found with $N = 20$.

The following table is used to generate a linearized regression model. Unfortunately, there is no significant variable that influences the performance of the algorithm. This suggests the algorithm's performance is reasonably robust with varying control parameters when $N = 20$ robots are present.

Variable	SumSq	DF	MeanSq	F	pValue
TRT	11.73	1	11.73	1.30	3.17e-1
IS	5.92	1	5.92	6.58e-1	4.63e-1
LSTT	11.63	1	11.61	1.29	3.19e-1

Table 5.7: ANOVA Table to determine the significance of control variables $TRT, IS, LSTT$ with $N = 20$.

The 30 trials for each of the 8 sets are simulated with $N = 20$ and displayed in Fig. 5.18. Set 5 shows to have extreme variability in comparison with set 1. TRT seems to be the causing factor of this. The ANOVA table, unfortunately, does not support this motion.

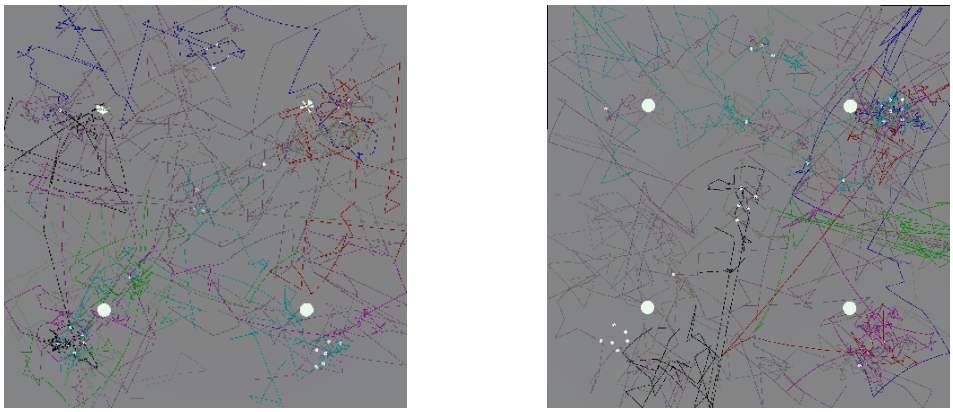


(a) TRT= 150, IS = 1, LSTT = 15. **(b)** TRT= 150, IS = 2, LSTT = 15. **(c)** TRT= 150, IS = 1, LSTT = 30. **(d)** TRT= 150, IS = 2, LSTT = 30.

(e) TRT= 300, IS = 1, LSTT = 15. **(f)** TRT= 300, IS = 2, LSTT = 15. **(g)** TRT= 300, IS = 1, LSTT = 30. **(h)** TRT= 300, IS = 2, LSTT = 30.

Figure 5.18: Visualization of the 30 performed trials per set with $N = 20$.

A visualisation of the highest (Fig. 5.19a), and lowest performer (Fig. 5.19b), taken from a random trial of the corresponding set, and are visualized next in Fig. 5.19. The variability between trials is quite large, and the figures are just for visualization purposes.



(a) Visualization of MBSO set 2 with $TRT = 150$, $IS = 2$, $LSTT = 15$ and $N = 20$.

(b) Visualization of MBSO set 6 with $TRT = 300$, $IS = 2$, $LSTT = 15$ and $N = 20$.

Figure 5.19: Visualization of random trials of the best and worst set within $N = 20$.

5.9. Results summary

The performance of using Modified Bee Swarm Optimization (MBSO) from solely random walking was shown to be significant. In particular, increasing N showed to increase performance. For $N = 10$, increasing TRT increased the performance, however, for larger numbers of robots, this had the opposing effect. The percentage of targets found by MBSO showed to scale linearly when going from $N = 10$ to $N = 15$ and to $N = 20$. However, with optimal control parameters, the percentage of targets found showed to increase exponentially. The largest difference between random walking and MBSO was in the case where $N = 10$ robots were used. Here, a percentual increase of 60.5 % in target finding performance was found. The redundancy of MBSO showed to scale exponentially with N. The total robustness of MSBO showed to be influenced by N, $LSTT$ and TRT. The stability of MBSO was hard to determine due to the large variability between the trials.

Performance levels of $N = 10, 15, 20$ were also analyzed. Only $N = 10$ could be modelled by TRT and $LSTT$. TRT showed to have a positive effect on the performance with $N = 10$, instead of a negative effect, which was the case for the whole model of $MBSO$. For $N = 15, 20$, no other significant differences that influence the performance could be found.

5.10. Final recommendations

MBSO exhibits a linear relationship between scaling the number of robots and the performance of the algorithm, taken over the average of all trials with varying control parameters. With optimal control parameters, scaling N improves the performance exponentially. Generally, increasing performance can be accomplished by increasing N and decreasing TRT (Sect. 5.3). Robustness can be increased by decreasing TRT, and increasing $LSTT$ and N (Sect. 5.4). The stability of the system is affected only by TRT, and invariant to the number of robots (Sect. 5.5). The redundancy can be decreased by decreasing the number of robots. In Case 2, where $TRT = 150$, $IS = 2$, and $LSTT = 15$, reasonably good performance is observed across different swarm sizes.

For a small swarm of $N = 10$ robots (Sect. 5.6), it is beneficial to set a large value for the control variable TRT and $LSTT$ to maximize performance. This means robots should move further for targets, and stay longer at target locations, to improve performance.

For a medium-sized swarm of $N = 15$ robots (Sect. 5.7), altering the control variables does not have a significant impact. Nonetheless, certain combinations may lead to reduced performance, so it is essential to maintain a balanced approach. In general, a lower TRT value shows to have a good impact on the performance. Some good parameters can be $TRT = 150$, $LSTT = 15$ or $TRT = 150$, $LSTT = 30$. Low performance is noticed at $TRT = 300$ and $LSTT = 15$.

For a large swarm of $N = 20$ robots (Sect. 5.8), performance fluctuates with varying parameters, and none of the control variables can be attributed as the sole causal factor. To mitigate this issue, the recommended approach is to utilize the control parameters that yield the highest performance. An overall summary of what to do for each optimization goal is given in Table 5.8.

	Overall performance	**Overall redundancy**	**Overall stability**	**Overall Robustness**	**Performance** $N = 10$
TRT	Decrease	-	Increase	Decrease	Increase
LSTT	-	-	-	Increase	Increase
N	Increase	Decrease	-	Increase	-

Table 5.8: Results of influence of variables on MBSO.

6 Discussion

The discussion is split up into several sections, as there are assumptions made on multiple levels. Simulation methods, the used control parameters, the movement of the swarm, the retrieved results and conclusions and further work are discussed separately.

6.1. Simulation methods

All simulations are performed in Webots [68]. Webots considerably simplifies kinematics and devices to increase simulation speeds. This does not form a problem if the models accurately reflect the real world. This can be minimized by introducing noise. Noise is not modelled in the algorithm, and could decrement the properties of the algorithm. For example, navigational noise. It is often not possible to get precise positioning of targets and robots. Even though the algorithm does not need precise target locations to function, it could nevertheless have an influence by, for example, recognizing one target as two different targets. In the current setup, targets are recognized at 15 m distance. This assumption is most likely not accurate in the case of recognizing targets, it would better fit for a task similar to mine searching. Realistic physical limitations like camera recognition distance, emitter range distance, movement speed, battery usage and more should be considered in more detail. This way, a more accurate simulation with regard to the real world can be done. Further, to accurately reflect the performance of the algorithm, real experiments should be done to validate the performance of the simulation. This cross-validation enables further tweaking and improvement of the algorithm, and the characteristic equations that define the properties (performance, robustness, etc.) of the algorithm. Also, the scaled-down simulation does not represent Huari well, and multiple scales should be tested to verify the scalability of the system. Additionally, to test the full capabilities of the algorithm, the robots need to be able to charge in between, enabling longer simulation times. The introduction of charging would also reveal the influence of utilising cluster repulsion, as robots now have more time to implement this functionality. It would also be useful to have other benchmark methods to accurately compare the proposed algorithm with methods like particle swarm optimization. Also, in the real world, robots sometimes fail, get stuck, or run out of battery. One of the important factors in swarm robotics is the flexibility against individuals failing. This should, therefore, also be modelled in the simulation.

6.2. Control parameters

The effectiveness of the control parameter O has not been proved in this study. However, it is expected that removing this functionality would drastically decrease performance, as there would not be a limit on the number of robots in each sub-swarm. Also, more control parameters like the battery state, the individual performance, or the number of obstacles in the robot's neighbourhood can be added to O, to optimize the exploration-exploitation parameters of the swarm. It would be beneficial to understand the performance difference that results from introducing O. Furthermore, the flexibility and scalability of the swarm system could be analyzed in further research. This could be done by adding or removing robots during a simulation and analyzing the performance, similar to work done by [15]. Other shapes and sizes of simulation spaces should also be tested, similar to [15]. Furthermore, constants $\gamma_{Levy\_exploit}$ and $\gamma_{Levy\_explore}$ are chosen randomly, and can be optimized to increase the

performance and balance of MBSO by managing the speed of which foragers turn back into scouts. In the same way b_{MIN}, should be modelled and optimized to find targets within a cluster, in an optimal way. Other influential variables are shown in Fig. 3.18. These should also be optimized. Lastly, the variance of the control variables can be increased such that their relevance becomes more clear.

6.3. Movement

The Lévy walk method used in MBSO should be optimized by changing b_{MAX} for different cluster distances. This should be determined in the actual simulation framework, instead of the circle method used in Sect. 3.6.3. It could be that varying the average step size changes the characteristics of the swarm drastically. This would enable one to determine the optimal step size for a certain cluster distance. Also, the cut-off for the Lévy step size may interfere with the exploration capabilities that Lévy walk has to offer. A simplistic implementation of the Artificial Potential Field (APF) is used to guide robots in this particular simulation. However, APF can be used in a more elegant way by adding more complex obstacle avoidance and guiding robots away from clusters in a better way. The wall avoidance method introduced in Sect. 3.8 could potentially introduce a bias, as robots always move in a certain direction after hitting a wall. This should be randomized to remove this bias.

6.4. Results and conclusions

In terms of the stability of the algorithm, it seems that the variability between trials, no matter the number of robots, or control parameters, is large enough to disguise the differences that result from changing the control parameters. It can help to, instead of randomizing the robots, initialize them at the same position. This could help generate more robust data, which in term would more accurately reflect the properties of the algorithm. Also, the performance of the algorithm scales linearly, where the redundancy scales exponentially. The performance, however, is equal to the percentage of targets found minus the redundancy. The exponential influence of the redundancy is not seen back in the linear performance measure. This suggests that N has a much larger influence on the performance than on the redundancy. One could, therefore, question the reliability of the overall performance measure, as it is similar to only analysing the percentage of targets visited.

6.5. Further work

Looking at the algorithm in a larger context, an archaeologist would want to use MBSO in the following way. An archaeologist should provide information regarding the distances between clusters, the average cluster size, the required robustness and efficiency, and the number of robots at his or her disposal. The control variables can then be determined based on the characteristic models of the algorithm. This information can be used to determine the right values of b_{MIN},b_{MAX}, TRT, LSTT and even N if the number of robots is not fixed. The robots then find and communicate targets using MBSO. When a robot is empty, it should start returning to some charge station. This example can be extended to other tasks where targets are dispersed in clusters. This is only possible if MBSO can accurately reflect the real task. Adding functionality like a more complex obstacle and wall avoidance and more efficient cluster repulsion would too be useful. Noise from devices should be modelled in the algorithm. Then, a small-scale real-life test can be done. The performance of the algorithm can be analyzed, and the properties of the algorithm can be adjusted such that they better reflect the real world. Webots [68] luckily enables all of the above to be implemented in the algorithm, and be carried over to real robots, as it enables the usage of real robot control architectures like ROS. Further improvements for MBSO are mentioned above.

7 Conclusion

A swarm system is supposed to be autonomous, have local and decentralized communication and sensing capabilities, have no access to global knowledge, and have robots that need to cooperate to tackle a given task effectively. In addition, individual and collective knowledge must be built within the swarm system. The thesis provides such a swarming algorithm that is autonomous, scalable and is developed to efficiently locate and map targets in an unknown environment where targets are spread out in clusters. The algorithm is based on the Bee Swarming Optimization (BSO) algorithm. BSO is a 3-state algorithm that uses communication to share points of interest with others. BSO is somewhat similar to particle swarm optimization in terms of the movement of single agents but has some overhanging framework to determine social behaviour. Standard BSO requires strict communication between experienced foragers and onlookers, and is, therefore, less fit for tasks where good positioning and communication (range) are limited. The algorithm is mostly fit for use cases where an abundance of information is available. In addition, BSO has not been optimized to locate clusters. To solve these issues, Modified Bee Swarm Optimization (MBSO) has been proposed.

7.1. Proposed algorithm

MBSO is different from BSO in that all robots initialize as scouts, this enhances the exploration capabilities of the swarm. Because of the communication range limitation, a target information-sharing method using ad hoc communication is developed. Here, global swarm intelligence is built from sharing of targets array throughout the swarm. This up-to-date target array is also the driver of movement in MBSO, instead of individual robot fitness levels, as is the case in BSO. Exploration and exploitation are balanced by implementing an architecture that enables the dynamic balancing of these properties by signalling when a scout should transition into an onlooker, and how fast an experienced forager changes back to a scout. The provided architecture enables more variables like the battery state, amount of obstacles in the neighbourhood and the performance of the individual to aid in balancing MBSO. Also, an Artificial Potential Field (APF) is added to enable the movement of the swarm members towards targets, and away from clusters. Using an APF enables more complex obstacle avoidance methods to be added to the algorithm. Cluster recognition is also added to the algorithm.

7.2. Performance

Performance is measured as the number of targets found, divided by the number of targets that are re-visited by multiple robots, e.g., the redundancy. Target finding performance is only the number of targets found. MBSO Average, which is the average target finding performance of MBSO over 8 sets of varying parameters, showed to have 31,9%, 14,7%, and 14,2% better target finding performance than random walking for $N = 10, 15, 20$, respectively. The performance difference decreases with increasing number of robots, as expected. This suggests that MBSO, relative to random walking is more optimal with fewer robots. However, if the control parameters of MBSO are set to be optimal, the target finding performance increases with 60,5%, 26,2%, and 25,0%, respectively. Changing control parameters to increase the target finding performance does influence other aspects of the swarm. It

should be determined whether increasing performance, lowering redundancy (increasing efficiency), increasing stability, or increasing robustness are prioritised to be able to reason about changing these parameters. Table 5.8 shows the impact of varying control parameters on MBSO. For large numbers of robots ($N = 15, 20$), lowering the maximum range where robots take into account targets (TRT) increases performance. For $N = 10$, however, increasing TRT, and locally searching for longer times $LSTT$, increases performance. Increasing the number of robots from $N = 10$ to $N = 15$ increases the redundancy of the system greatly. However, going from $N = 15$ to $N = 20$ shows a much less increase. The robustness of MBSO, e.g., how it handles outliers, can be increased by decreasing TRT, and increasing $LSTT$ and N. This swarming algorithm could, with further research, be used to alleviate archaeologists from the initial survey phase. The control parameters can be selected based on the number of available robots, and the optimization preference.

7.3. Answering the sub-questions

The sub-questions inspired from the literature are answered next:

Do the physical range limitations improve the performance of the robots, due to robots being forced to explore?

The physical range limitation of the algorithm can be split into the limitation between robots, and between robots and targets. The limitations between robots are simulated as the emitter range ER, whereas the range limitation between robots and targets is simulated as TRT, e.g., the target recognition threshold. An independent study has been done for $N = 10$ in Fig. A.1 where $TRT = 150, 300, 500$ and $ER = 50, 100, 200$ are varied. A confidence level of 95% is taken, which sets $Z = Z_{0.975} = 1.96$. The desired margin of error is set to 5. From this study, it can be concluded that increasing the emitter range ER increases performance, until a certain value. Then, increasing ER has the opposite effect. In Sect. 5.10, it was shown that the influence of TRT is dependent on the number of robots. When $N = 10$, increasing TRT has a positive effect, on the other hand, $N = 15, 20$ TRT should be decreased to increase the performance of MBSO. Concluding, the physical range limitation between robots, and between robots and targets can improve the performance of MBSO. However, one should proceed with caution as at some point it has the opposite effect, decreasing the performance of the algorithm.

Does the performance of the algorithm scale linearly when adding more robots?

The performance when increasing N linearly increases the performance of MBSO over the average of all sets. However, it is possible to increase the performance exponentially, by varying the control parameters in favour of the number of robots used. The variable N shows a highly significant effect on the target finding performance and overall performance. The large F-value $F = 459.7$ and extremely small p-value $p = 8.9603e - 15$ support the motion for this strong impact. TRT also shows to be significant, as its p-value is less than 0.05 and equal to $p = 2.97e - 2$ with $F = 5.53$. Increasing N drastically increases performance. Adding a single robot should on average give around 2.08% performance gain. However, the steepness of the performance increase can be varied by adjusting the control variables such that more optimal performance is realized, based on the number of robots.

Does migration between sub-swarms increase optimality?

Optimality in this case is seen as any type of improvement, resulting from singular migrations between clusters. There was no proof that migration between sub-swarms improves optimality, as variable IS, or O, responsible for managing the maximum number of robots in a cluster, did not show any significance in tests with $N = 10, 15, 20$ robots. Increasing TRT also has some say in the migration of swarm members between clusters. However, is not directly responsible for sub-swarm migration.

7.4. Characteristics of modified bee swarm optimization

Overall performance

The average performance of MBSO is influenced by TRT and N and can be modelled by the following linear relationship:

$$\text{Performance} = \beta_0 + \beta_1 \cdot N + \beta_2 * TRT$$

with $\beta_0 = 40.671$, $\beta_1 = 2.082$ and $\beta_2 = -1.243 \times 10^{-2}$.

Notably, for $N = 10$, the performance of MBSO increases with increasing values of TRT and $LSTT$ and can be better modelled by the following equation:

$$\text{Performance} = \beta_0 + \beta_1 \cdot TRT + \beta_2 * LSTT$$

with $\beta_0 = 58.06$, $\beta_1 = 1.1e - 2$ and $\beta_2 = 1.56 \times 10^{-4}$.

Overall redundancy

The overall redundancy of MBSO is influenced by N, is not linear, and can be modelled by:

$$\text{Redundancy} = \beta_0 + \beta_1 * N - \beta_2 * N^2$$

with $\beta_0 = 6.091e - 1$, $\beta_1 = 3.38e - 2$, and $\beta_2 = -8.00e - 4$.

Overall robustness

The average robusteness of MBSO is influenced by N, TRT and $LSTT$ and can be modelled by the following linear relationship:

$$\text{Robustness} = \beta_0 + \beta_1 \cdot N + \beta_2 \cdot TRT + \beta_3 \cdot LSTT$$

with $\beta_0 = 20.23$ and $\beta_1 = 2.27$, $\beta_2 = -3e - 2$ and $\beta_3 = 1.62e - 1$.

Overall stability

The overall stability of the MBSO algorithm is influenced by TRT and can be modelled by:

$$\sigma = \beta_0 + \beta_1 \cdot TRT$$

with $\beta_0 = 8.65$ and $\beta_1 = 9.49e - 3$. Please take note that the stability of the algorithm increases when σ decreases. The stability of the algorithm increases with lower values of TRT. It seems that the variability between trials, no matter the number of robots or control parameters, is large enough to disguise the differences that result from changing the control parameters.

General useful parameters for MBSO are $TRT = 150$, $IS = 2$, and $LSTT = 15$ for varying numbers of robots $N = 10, 15, 20$.

References

[1] Reza Akbari, Alireza Mohammadi, and Koorush Ziarati. “A novel bee swarm optimization algorithm for numerical function optimization”. In: (2009). DOI: 10.1016/j.cnsns.2009.11.003. URL: www.elsevier.com/locate/cnsns.

[2] Ross D. Arnold, Hiroyuki Yamaguchi, and Toshiyuki Tanaka. “Search and rescue with autonomous flying robots through behavior-based cooperative intelligence”. In: *Journal of International Humanitarian Action 2018 3:1* 3.1 (Dec. 2018), pp. 1–18. ISSN: 2364-3404. DOI: 10.1186/S41018-018-0045-4. URL: https://link.springer.com/articles/10.1186/s41018-018-0045-4%20https://link.springer.com/article/10.1186/s41018-018-0045-4.

[3] Levent Bayindir Erol Şahin and Levent Bayindir. *A Review of Studies in Swarm Robotics*. Tech. rep. 2. 2007.

[4] Rafael G. Braga et al. “UAV Swarm Control Strategies: a Case Study for LeakDetection”. In: *UAV Swarm Control Strategies: a Case Study for LeakDetection*. 2017. ISBN: 9781538631577.

[5] Manuele Brambilla et al. “Swarm robotics: A review from the swarm engineering perspective”. In: *Swarm Intelligence* 7.1 (Mar. 2013), pp. 1–41. ISSN: 19353820. DOI: 10.1007/s11721-012-0075-2.

[6] Yifan Cai and Simon X. Yang. “A PSO-based approach with fuzzy obstacle avoidance for cooperative multi-robots in unknown environments”. In: *2013 IEEE International Conference on Information and Automation, ICIA 2013*. 2013, pp. 1386–1391. ISBN: 9781479913343. DOI: 10.1109/ICInfA.2013.6720510.

[7] Yifan Cai and Simon X. Yang. “An improved PSO-based approach with dynamic parameter tuning for cooperative target searching of multi-robots”. In: *World Automation Congress Proceedings*. IEEE Computer Society, Oct. 2014, pp. 616–621. ISBN: 9781889335490. DOI: 10.1109/WAC.2014.6936067.

[8] Yifan Cai and Simon X. Yang. “A potential field-based PSO approach for cooperative target searching of multi-robots”. In: *Proceedings of the World Congress on Intelligent Control and Automation (WCICA)*. Vol. 2015-March. March. Institute of Electrical and Electronics Engineers Inc., Mar. 2015, pp. 1029–1034. ISBN: 9781479958252. DOI: 10.1109/WCICA.2014.7052858.

[9] R Cassinis et al. *Strategies for navigation of robot swarms to be used in landmines detection*. Tech. rep. IEEE, 1999.

[10] AV Chechkin et al. “Introduction to the theory of Lévy flights”. In: *Wiley Online Library* (). URL: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9783527622979#page=153.

[11] Yifan Chen, Panagiotis Kosmas, and Sylvain Martel. “A feasibility study for microwave breast cancer detection using contrast-agent-loaded bacterial microbots”. In: *International Journal of Antennas and Propagation* 2013 (2013). ISSN: 16875869. DOI: 10.1155/2013/309703.

[12] Floriano De Rango et al. “Swarm robotics in wireless distributed protocol design for coordinating robots involved in cooperative tasks”. In: *Soft Computing* 22.13 (July 2018), pp. 4251–4266. ISSN: 14337479. DOI: 10.1007/S00500-017-2819-9/FIGURES/19. URL: https://link.springer.com/article/10.1007/s00500-017-2819-9.

[13] Kurt Derr and Milos Manic. “Multi-robot, multi-target particle swarm optimization search in noisy wireless environments”. In: *Proceedings - 2009 2nd Conference on Human System Interactions, HSI ’09*. 2009, pp. 81–86. ISBN: 9781424439607. DOI: 10.1109/HSI.2009.5090958.

[14] Marco Dorigo. “Optimization, Learning and Natural Algorithms. ” PhD thesis. Ph.D. Thesis, Politecnico di Milano, Italian. - References - Scientific Research Publishing, 1992. URL: https://www.scirp.org/(S(351jmbntvnsjt1aadkposzje))/reference/ReferencesPapers.aspx?ReferenceID=1301739.

[15] Miguel Duarte et al. “Evolution of Collective Behaviors for a Real Swarm of Aquatic Surface Robots”. In: *PLOS ONE* 11.3 (Mar. 2016), e0151834. ISSN: 1932-6203. DOI: 10.1371/JOURNAL.PONE.0151834. URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0151834.

[16] R Fujisawa, S Dobata - Proceedings of the 2013 IEEE/SICE, and undefined 2013. “Lévy walk enhances efficiency of group foraging in pheromone-communicating swarm robots”. In: *ieeexplore.ieee.org* (). URL: https://ieeexplore.ieee.org/abstract/document/6776760/.

[17] A Gil et al. “Architecture for multi-robot systems with emergent behavior”. In: *Architecture for multi-robot systems with emergent behavior.* Proceedings on the International Conference on Artificial Intelligence (ICAI), 2015, p. 41.

[18] Aniket Gupta et al. “A Particle Swarm Optimization-Based Cooperation Method for Multiple-Target Search by Swarm UAVs in Unknown Environments”. In: *2021 International Conference on Automation, Robotics and Applications, ICARA 2021*. Institute of Electrical and Electronics Engineers Inc., Feb. 2021, pp. 95–100. ISBN: 9780738142906. DOI: 10.1109/ICARA51699.2021.9376529.

[19] D Guzzoni et al. “Many robots make short work”. In: *ojs.aaai.org* 18.I (1997). URL: https://ojs.aaai.org/index.php/aimagazine/article/view/1274.

[20] H Haklı, H Uğuz - Applied Soft Computing, and undefined 2014. “A novel particle swarm optimization algorithm with Levy flight”. In: *Elsevier* (). URL: https://www.sciencedirect.com/science/article/pii/S1568494614003081.

[21] AT Hayes, A Martinoli, and RM Goodman. “Swarm robotic odor localization: Off-line optimization and validation with real robots”. In: *cambridge.org* (2003). URL: https://www.cambridge.org/core/journals/robotica/article/swarm-robotic-odor-localization-offline-optimization-and-validation-with-real-robots/00A5AE0791542848945A5C9CC2527D1D.

[22] Oliver C. Ibe. *Markov Processes for Stochastic Modeling: Second Edition*. 2013. DOI: 10.1016/C2012-0-06106-6.

[23] Upma Jain, Ritu Tiwari, and W. Wilfred Godfrey. “Multiple odor source localization using diverse-PSO and group-based strategies in an unknown environment”. In: *Journal of Computational Science* 34 (May 2019), pp. 33–47. ISSN: 18777503. DOI: 10.1016/j.jocs.2019.04.008.

[24] Wisnu Jatmiko, Kosuke Sekiyama, and Toshio Fukuda. *A PSO-based Mobile Sensor Network for Odor Source Localization in Dynamic Environment: Theory, Simulation and Measurement*. Tech. rep. 2006.

[25] Aleksandar Jevtic et al. “BUILDING A SWARM OF ROBOTIC BEES”. In: (2010).

[26] Krishnanand N Kaipa et al. *LNCS 4150 - Rendezvous of Glowworm-Inspired Robot Swarms at Multiple Source Locations: A Sound Source Based Real-Robot Implementation*. Tech. rep. 2006.

[27] Yoshiaki Katada and Kaito Yamashita. “Swarm robots using Lévy walk based on nonlinear dynamics for target exploration”. In: *Artificial Life and Robotics* 27.2 (May 2022), pp. 226–235. ISSN: 16147456. DOI: 10.1007/s10015-022-00740-3.

[28] Yoshiaki Katada et al. “Swarm robotic network using Lévy flight in target detection problem”. In: *Artificial Life and Robotics* 21.3 (Sept. 2016), pp. 295–301. ISSN: 16147456. DOI: 10.1007/S10015-016-0298-1.

[29] Yara Khaluf, Mauro Birattari, and Franz Rammig. “Analysis of long-term swarm performance based on short-term experiments”. In: *Soft Computing* 20.1 (Jan. 2016), pp. 37–48. ISSN: 14337479. DOI: 10.1007/S00500-015-1958-0.

[30] Yara Khaluf, Stef Van Havermaet, and Pieter Simoens. “Collective lévy walk for efficient exploration in unknown environments”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 11089 LNAI (2018), pp. 260–264. ISSN: 16113349. DOI: 10.1007/978-3-319-99344-7{_}24.

[31] Yara Khaluf, Stef Van Havermaet, and Pieter Simoens. “Collective lévy walk for efficient exploration in unknown environments”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 11089 LNAI. Springer Verlag, 2018, pp. 260–264. ISBN: 9783319993430. DOI: 10.1007/978-3-319-99344-7{_}24.

[32] Yara Khaluf et al. “Scale invariance in natural and artificial collective systems: a review”. In: *royalsocietypublishing* (2017). DOI: 10.1098/rsif.2017.0662.

[33] Áron Kisdi and Adrian R.L. Tatnall. “Future robotic exploration using honeybee search strategy: Example search for caves on Mars”. In: *Acta Astronautica* 68.11-12 (June 2011), pp. 1790–1799. ISSN: 0094-5765. DOI: 10.1016/J.ACTAASTRO.2011.01.013.

[34] K. N. Krishnanand and D. Ghose. “Detection of multiple source locations using a glowworm metaphor with applications to collective robotics”. In: *Proceedings - 2005 IEEE Swarm Intelligence Symposium, SIS 2005* 2005 (2005), pp. 84–91. DOI: 10.1109/SIS.2005.1501606.

[35] K. N. Krishnanand and D. Ghose. “A glowworm swarm optimization based multi-robot system for signal source localization”. In: *Studies in Computational Intelligence* 177 (2009), pp. 49–68. ISSN: 1860949X. DOI: 10.1007/978-3-540-89933-4{_}3/COVER. URL: https://link.springer.com/chapter/10.1007/978-3-540-89933-4_3.

[36] K. N. Krishnanand and Debasish Ghose. “Glowworm swarm based optimization algorithm for multimodal functions with collective robotics applications”. In: *Multiagent and Grid Systems* 2.3 (Jan. 2006), pp. 209–222. ISSN: 1574-1702. DOI: 10.3233/MGS-2006-2301.

[37] Vignesh Kumar and Ferat Sahin. *Cognitive Maps in Swarm Robots for the Mine Detection Application**. Tech. rep. New York: Rochester Institute of Technology, 2003.

[38] CY Lee, X Yao - Transactions on Evolutionary Computation, and undefined 2004. “Evolutionary programming using mutations based on the Lévy probability distribution”. In: *ieeexplore.ieee.org* (). URL: https://ieeexplore.ieee.org/abstract/document/1266370/.

[39] Natalie J. Lemanski et al. “A Multiscale Review of Behavioral Variation in Collective Foraging Behavior in Honey Bees”. In: *Insects 2019, Vol. 10, Page 370* 10.11 (Oct. 2019), p. 370. ISSN: 2075-4450. DOI: 10.3390/INSECTS10110370. URL: https://www.mdpi.com/2075-4450/10/11/370/htm%20https://www.mdpi.com/2075-4450/10/11/370.

[40] Guanghui Li et al. “Effective improved artificial potential field-based regression search method for autonomous mobile robot path planning”. In: *International Journal of Mechatronics and Automation* 3.3 (2013), p. 141. ISSN: 2045-1059. DOI: 10.1504/IJMA.2013.055612.

[41] Guannan Li, Hongli Xu, and Yang Lin. “Application of Bat Algorithm Based Time Optimal Control in Multi-robots Formation Reconfiguration”. In: *Journal of Bionic Engineering* 15 (2018), pp. 126–138. DOI: 10.1007/s42235-017-0010-8. URL: http://www.springer.com/journal/42235.

[42] Jie Li and Ying Tan. *A Probabilistic Finite State Machine based Strategy for Multi-Target Search Using Swarm Robotics*. Tech. rep. 2019.

[43] Qing Hao Meng et al. “Adapting an Ant Colony Metaphor for Multi-Robot Chemical Plume Tracing”. In: *Sensors 2012, Vol. 12, Pages 4737-4763* 12.4 (Apr. 2012), pp. 4737–4763. ISSN: 1424-8220. DOI: 10.3390/S120404737. URL: https://www.mdpi.com/1424-8220/12/4/4737/htm%20https://www.mdpi.com/1424-8220/12/4/4737.

[44] Sabudin ella Nadira, Che Ku Nor Azie Halima, and Rosil Oma. *(PDF) Potential field methods and their inherent approaches for path planning*. 2016. URL: https://www.researchgate.net/publication/313389747_Potential_field_methods_and_their_inherent_approaches_for_path_planning.

[45] Hannaneh Najd Ataei, Koorush Ziarati, and Mohammad Eghtesad. “A BSO-based algorithm for multi-robot and multi-target search”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7906 LNAI (2013), pp. 312–321. ISSN: 03029743. DOI: 10.1007/978-3-642-38577-3{_}32.

[46] A Nath and SR Walimbe. “Harappan interments at Rakhigarhi, Haryana”. In: *academia.edu* (2015). URL: https://www.academia.edu/download/41060163/Harappan_Interment_at_Rakhigrhi__Haryana.pdf.

[47] J Nauta et al. “Enhanced foraging in robot swarms using collective Lévy walks”. In: *biblio.ugent.be* (2020). DOI: 10.3233/FAIA200090. URL: https://biblio.ugent.be/publication/8676969.

[48] Zedadra Ouarda et al. “Exploring unknown environments with multi-modal locomotion swarm”. In: *Studies in Computational Intelligence* 678 (2017), pp. 131–140. ISSN: 1860949X. DOI: 10.1007/978-3-319-48829-5{_}13.

[49] Bao Pang et al. “A Swarm Robotic Exploration Strategy Based on an Improved Random Walk Method”. In: *Journal of Robotics* 2019 (2019). ISSN: 16879619. DOI: 10.1155/2019/6914212.

[50] Jacques Penders et al. “GUARDIANS: a Swarm of Autonomous Robots for Emergencies”. In: *academia.edu* (2007).

[51] N. Perozo et al. “A Verification Method for MASOES”. In: *IEEE transactions on cybernetics* 43.1 (Feb. 2013), pp. 64–76. ISSN: 2168-2275. DOI: 10.1109/TSMCB.2012.2199106. URL: https://pubmed.ncbi.nlm.nih.gov/22692924/.

[52] Niriaska Perozo, Oswaldo Terán, and Jose Aguilar. “Proposal for a Multiagent Architecture for Self-Organizing Systems (MA-SOS) Autonomic Computing Platforms View project Smart Classroom View project Proposal for a Multiagent Architecture for Self-Organizing Systems (MA-SOS)”. In: *LNCS* 5075 (2008), pp. 434–439. DOI: 10.1007/978-3-540-69304-8{_}45. URL: https://www.researchgate.net/publication/225179251.

[53] GL Possehl. “The Indus civilization: a contemporary perspective”. In: (2002). URL: https://books.google.com/books?hl=nl&lr=&id=pmAuAsi4ePIC&oi=fnd&pg=PR6&dq=+Possehl,+Gregory+L.+(2002),+The+Indus+Civilization:+A+Contemporary+Perspective,+Rowman+Altamira,+p.+72,+ISBN+978-0-7591-0172-2+Quote:+%22The+site+is+about+17+meters+in+height.+The+southern+face+of+the+mounds+is+rather+abrupt+and+steep.+The+northern+side+slo&ots=8B4bx-ZsNZ&sig=Zr1D3t8Z4Gh9SXEOoU-0k2v6Zk8.

[54] Yoan Purbolingga, Achmad Jazidie, and Rusdhianto Effendi. “Modified Ant Colony Algorithm For Swarm Multi Agent Exploration on Target Searching in Unknown Environment”. In: *Modified Ant Colony Algorithm For Swarm Multi Agent Exploration on Target Searching in Unknown Environment*. Ed. by Yoan Purbolingga, Achmad Jazidie, and Rusdhianto Effendi. Surbaya: IEEEXPLORE, 2019. ISBN: 9781538684481.

[55] Alessandro Renzaglia and Agostino Martinelli. “Potential field based approach for coordinate exploration with a multi-robot team”. In: *8th IEEE International Workshop on Safety, Security, and Rescue Robotics, SSRR-2010* (2010). DOI: 10.1109/SSRR.2010.5981557. URL: https://www.researchgate.net/publication/47387448_Potential_Field_based_Approach_for_Coordinate_Exploration_with_a_Multi-Robot_Team.

[56] Erol Sahin. “Swarm Robotics: From Sources of Inspiration to Domains of Application. Swarm robotics View project”. In: *Conference Paper in Lecture Notes in Computer Science* (2004). URL: https://www.researchgate.net/publication/221116606.

[57] Hugo Sardinha, Mauro Dragone, and Patricia A. Vargas. “Combining Lévy Walks and Flocking for Cooperative Surveillance Using Aerial Swarms”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 12520 LNAI (2020), pp. 226–242. ISSN: 16113349. DOI: 10.1007/978-3-030-66412-1{_}15.

[58] Subir Kumar Sarkar, T.G. Basavaraju, and C. Puttamadappa. “Ad Hoc Mobile Wireless Networks : Principles, Protocols and Applications”. In: *Ad Hoc Mobile Wireless Networks* (Oct. 2007). DOI: 10.1201/9781420062229. URL: https://www.taylorfrancis.com/books/mono/10.1201/9781420062229/ad-hoc-mobile-wireless-networks-subir-kumar-sarkar-basavaraju-puttamadappa.

[59] Donny K. Sutantyo et al. “Multi-Robot Searching Algorithm Using Levy Flight and Artificial Potential Field”. In: (Aug. 2011). URL: http://arxiv.org/abs/1108.5624.

[60] Ying Tan and Zhong yang Zheng. “Research Advance in Swarm Robotics”. In: *Defence Technology* 9.1 (Mar. 2013), pp. 18–39. ISSN: 22149147. DOI: 10.1016/J.DT.2013.03.001.

[61] Hongwei Tang et al. “A novel hybrid algorithm based on PSO and FOA for target searching in unknown environments”. In: *Applied Intelligence* 49.7 (July 2019), pp. 2603–2622. ISSN: 15737497. DOI: 10.1007/s10489-018-1390-0.

[62] Hongwei Tang et al. “A multirobot target searching method based on bat algorithm in unknown environments”. In: *Expert Systems with Applications* 141 (Mar. 2020). ISSN: 09574174. DOI: 10.1016/j.eswa.2019.112945.

[63] Qirong Tang et al. “Swarm Robots Search for Multiple Targets Based on an Improved Grouping Strategy”. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 15.6 (Nov. 2018), pp. 1943–1950. ISSN: 15579964. DOI: 10.1109/TCBB.2017.2682161.

[64] Lidio M. Valdez and J. Ernesto Valdez. “From Rural to Urban: Archaeological Research in the Periphery of Huari, Ayacucho Valley, Peru”. In: *Journal of Anthropology* 2017 (Feb. 2017), pp. 1–14. ISSN: 2090-4045. DOI: 10.1155/2017/3597297.

[65] P Visalakshi, Mahesh Mishra, and Snehasish Maity. “Ad hov network- an overview”. In: *International Journal of Modern Engineering Research (IJMER) www.ijmer.com Pp* (), pp. 27–29. ISSN: 2249-6645. URL: http://ovais.khan.tripod.com/papers/Secure_Routing.

[66] Gandhimohan M Viswanathan et al. “Lévy fights in random searches”. In: *Physica A* 282 (2000).

[67] N Walia and H Dar. “Archaeological Connotation and Rural Tourism Development: A Study of Rakhigarhi”. In: (2020). URL: http://www.ezeichen.com/gallery/1600.pdf.

[68] Webots. *http://www.cyberbotics.com*. Ed. by Cyberbotics Ltd. 2023. URL: http://www.cyberbotics.com.

[69] Xin-She Yang. “A New Metaheuristic Bat-Inspired Algorithm, in: Nature Inspired Coop-erative Strategies for Optimization”. In: 284 (2010), pp. 65–74.

[70] XS Yang. “Engineering optimization: an introduction with metaheuristic applications”. In: (2010). URL: https://books.google.com/books?hl=nl&lr=&id=kTi6ul2g9VYC&oi=fnd&pg=PR5&dq=X.-S.+Yang,Engineering+optimization:+an+introduction+withmetaheuristic+applications.++++John+Wiley+%26+Sons,+2010&ots=vYPSccn01L&sig=BNLkczUzCAea5_viBC11X9KYBEI.

[71] Xiaofang Yuan et al. “On a novel multi-swarm fruit fly optimization algorithm and its application”. In: *Applied Mathematics and Computation* 233 (May 2014), pp. 260–271. ISSN: 00963003. DOI: 10.1016/J.AMC.2014.02.005.

[72] Jun Qi Zhang, Yehao Lu, and Mengchu Zhou. “Hybrid Topology-Based Particle Swarm Optimizer for Multi-source Location Problem in Swarm Robots”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 13345 LNCS. Springer Science and Business Media Deutschland GmbH, 2022, pp. 17–24. ISBN: 9783031097256. DOI: 10.1007/978-3-031-09726-3{_}2.

[73] Yiwen Zhang et al. “A novel multi-scale cooperative mutation Fruit Fly Optimization Algorithm”. In: *Knowledge-Based Systems* 114 (Dec. 2016), pp. 24–35. ISSN: 09507051. DOI: 10.1016/J.KNOSYS.2016.09.027.

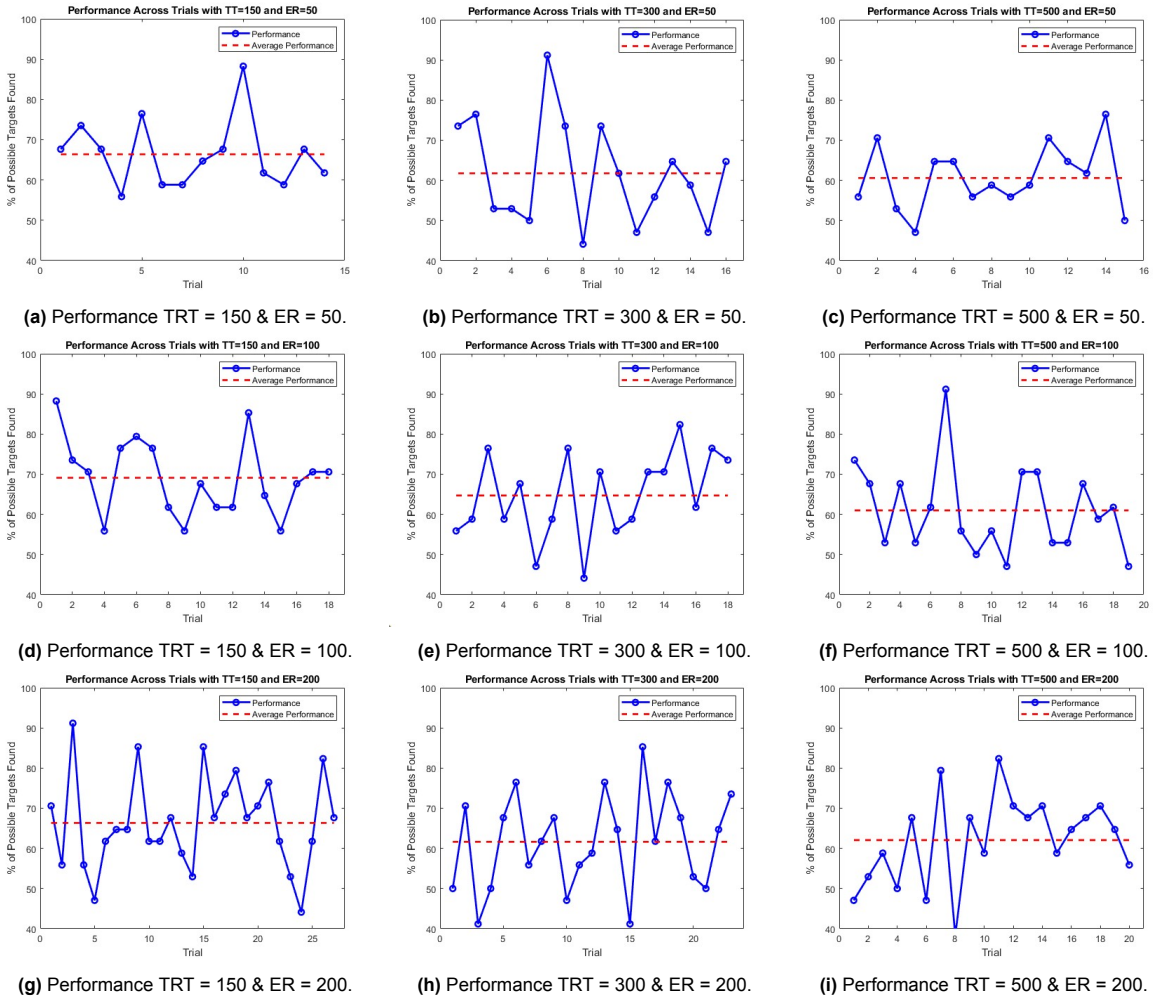
[74] Z Zheng and Tan Ying. “Group explosion strategy for searching multiple targets using swarm robotic”. In: *ieeexplore.ieee.org* (2013). URL: https://ieeexplore.ieee.org/abstract/document/6557653/.

[75] You Zhou et al. “Multitarget Search of Swarm Robots in Unknown Complex Environments”. In: *Complexity* 2020 (2020). ISSN: 10990526. DOI: 10.1155/2020/8643120.

[76] Rui Zou et al. “Particle Swarm Optimization-Based Source Seeking”. In: *IEEE Transactions on Automation Science and Engineering* 12.3 (July 2015), pp. 865–875. ISSN: 15455955. DOI: 10.1109/TASE.2015.2441746.

A

Emitter Range



(a) Performance TRT = 150 & ER = 50. **(b)** Performance TRT = 300 & ER = 50. **(c)** Performance TRT = 500 & ER = 50.

(d) Performance TRT = 150 & ER = 100. **(e)** Performance TRT = 300 & ER = 100. **(f)** Performance TRT = 500 & ER = 100.

(g) Performance TRT = 150 & ER = 200. **(h)** Performance TRT = 300 & ER = 200. **(i)** Performance TRT = 500 & ER = 200.

Figure A.1: Simulations with varying $TRT = 150, 300, 500$ and $ER = 50, 100, 200$.

B

Webots Code

The main controller code is given below. The different robot functions (scout, onlooker, forager) are controlled using a state machine. Additionally, a homing function is added. This functionality is not used. The main code also controls sensor initialization, regulates communication using emitter and receiver functions, cluster recognition, making the repel list, Braitenberg obstacle avoidance, and saving of target array files.

```
/* Main Controller
 * Information that is sent between robot are shorts
 * Local target information that is used for navigation and movement is doubles
 */
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <webots/distance_sensor.h>
#include <webots/motor.h>
#include <webots/robot.h>
#include <webots/emitter.h>
#include <webots/receiver.h>
/* Including function files */
#include "C:\...\recognition.c"
#include "C:\...\levy_distribution.c"
#include "C:\...\normal_distribution.c"
#include "C:\...\fuzzy_treshold.c"
#include "C:\...\MBSO\save.c"
#include "C:\...\APF.c"
/* Definitions for Sensor and Robot*/
#define WHEEL_WEIGHT_THRESHOLD 50
#define MAX_SENSOR_NUMBER 16
#define MAX_SENSOR_VALUE 1024
#define MIN_DISTANCE_SENSOR 3
#define MAX_SPEED 6.4
#define TARGET_RECOGNITION_TRESHOLD 300
#define LOCAL_SEARCH_TIME_TRESHOLD 30 * 60
#define RANGE_OF_CLUSTER 100
#define GAMMA_MAX 33
#define EMITTER_RANGE 100
#define BS_TRESHOLD 0
#define N_Chargers 4
#define MIN_POTENTIAL_DISTANCE 75
/* Enables broadcasting to all channels */
#define WB_CHANNEL_BROADCAST -1
```

```
/* Time */
#define TIME_STEP 64
/* How many targets can the robot find? */
#define AMOUNT_OF_TARGETS 34
#define MAX_ROBOTS 10
/* States */
typedef enum
{
    RESET, //0
    LEVY, //1
    GOTO, //2
    HOMING, //3
    AVOID, //4
    LEFT, //5
    RIGHT, //6
} State;



////////////////////////////////Main script////////////////////////////////
////////////////////////////////Main script////////////////////////////////
////////////////////////////////Main script////////////////////////////////
int main(int argc, const char *argv[])
{
    wb_robot_init();
    srand(time(NULL)); // Seed the random number generator with the current time
    int CUSTOM_SEED = *argv[1];
    WbDeviceTag communication;
    communication = wb_robot_get_device("emitter");
    wb_emitter_set_range(communication, EMITTER_RANGE);
    WbDeviceTag communication_receiver;
    communication_receiver = wb_robot_get_device("receiver");
    wb_receiver_enable(communication_receiver, TIME_STEP);
    wb_robot_battery_sensor_enable(TIME_STEP);
    ////////////////////////////////Wheels init////////////////////////////////
    /* Stores device IDs */
    WbDeviceTag front_left_wheel = wb_robot_get_device("front left wheel");
    WbDeviceTag front_right_wheel = wb_robot_get_device("front right wheel");
    WbDeviceTag back_left_wheel = wb_robot_get_device("back left wheel");
    WbDeviceTag back_right_wheel = wb_robot_get_device("back right wheel");

    /* Init motors */
    wb_motor_set_position(front_left_wheel, INFINITY);
    wb_motor_set_position(front_right_wheel, INFINITY);
    wb_motor_set_position(back_left_wheel, INFINITY);
    wb_motor_set_position(back_right_wheel, INFINITY);

    /* Init speed value */
    double front_left_speed = 0.0;
    double front_right_speed = 0.0;
    double back_left_speed = 0.0;
    double back_right_speed = 0.0;

    /* Set init velocity */
    wb_motor_set_velocity(front_left_wheel, front_left_speed);
    wb_motor_set_velocity(front_right_wheel, front_right_speed);
    wb_motor_set_velocity(back_left_wheel, back_left_speed);
    wb_motor_set_velocity(back_right_wheel, back_right_speed);

    ////////////////////////////////ONLOOKER specific init////////////////////////////////
    WbDeviceTag gps = wb_robot_get_device("gps");
```

```
    WbDeviceTag inertial_unit = wb_robot_get_device("inertial unit");
    wb_gps_enable(gps, TIME_STEP);
    wb_inertial_unit_enable(inertial_unit, TIME_STEP);

    ////////////////////////////////Sensory init////////////////////////////////
    typedef struct
    {
        WbDeviceTag device_tag;
        double wheel_weight[4];
    } SensorData;

    /* Home position array*/
    static short home_array[N_Chargers][2] = {{300, 300}, {300, -300}, {-300, 300}, {-300,
        -300}};

    /* Effect Sensor index[0-15] on wheel index[0-3] */
    static SensorData sensors[MAX_SENSOR_NUMBER] =
    {
    {.wheel_weight = {150, 0, 150, 0}}, {.wheel_weight = {200, 0, 200, 0}},
    {.wheel_weight = {300, 0, 300, 0}}, {.wheel_weight = {600, 0, 600, 0}},
    {.wheel_weight = {0, 600, 0, 600}}, {.wheel_weight = {0, 300, 0, 300}},
    {.wheel_weight = {0, 200, 0, 200}}, {.wheel_weight = {0, 150, 0, 150}},
    {.wheel_weight = {0, 0, 0, 0}}, {.wheel_weight = {0, 0, 0, 0}},
    {.wheel_weight = {0, 0, 0, 0}}, {.wheel_weight = {0, 0, 0, 0}},
    {.wheel_weight = {0, 0, 0, 0}}, {.wheel_weight = {0, 0, 0, 0}},
    {.wheel_weight = {0, 0, 0, 0}}, {.wheel_weight = {0, 0, 0, 0}}
    };

    /* Enabel distance sensors */
    char sensor_name[5] = "";
    int i, j;
    for (i = 0; i < MAX_SENSOR_NUMBER; ++i)
    {
        sprintf(sensor_name, "so%d", i);
        sensors[i].device_tag = wb_robot_get_device(sensor_name);
        wb_distance_sensor_enable(sensors[i].device_tag, TIME_STEP);
    }

    /* Init sensor influence on wheel speed variables*/
    double wheel_weight_total[4] = {0.0, 0.0, 0.0, 0.0};
    double distance, speed_modifier, sensor_value;
    double speed[4] = {0.0, 0.0, 0.0, 0.0};

    /* State control, resetting and updating of targets*/
    int state = LEVY;
    short xy_array_target[2] = {0};

    /* Wall recognition based on field size*/
    int wall_pos[2] = {0};
    bool wall = false;
    int size_field = 1200/2;

    /* Onlooker or Scout?*/
    bool onlooker = false;
    bool home_reached = false;
    bool onlooker_treshold = true;

     /* Print final results only once*/
    bool has_printed = false;

    /* Global position of the target and home*/
```

```
159     int x_t = 0;
160     int y_t = 0;
161     int x_h = 0;
162     int y_h = 0;
163 
164     /* Initalize Normal Generator*/
165     double mu;
166     double sigma = M_PI;
167 
168     /* Initialize distance vector to chosen home or target double because the distance is
            calculated on 0.1 m accuracy*/
169     double distance_target = 0.0;
170 
171     /* Initialize target timer from last target found for cluster forming*/
172     double last_target_time = 0.0;
173 
174     /* Initialize repelling list with unattractive coordinates of clusters saved*/
175     int repel_list[MAX_REPEL_LIST][2]= {0};
176 
177     /* Initialize distance of this cluster list to determine closest cluster*/
178     int distance_to_clusters[MAX_REPEL_LIST] = {0};
179 
180     /* Information states about the robot*/
181     short BS; // Battery state --> Battery level of robot
182     short OS = 0; // Operatation state --> Targets found by current robot
183     short IS = 0; // Interadtion state --> Robots in range
184 
185     ////////////////////////////////Helper functions////////////////////////////////
186     /* Initialize normal generator
187      * Check normal_distribution.c file for generator
188      */
189     double normal = normalize_angle(randn(mu, sigma, CUSTOM_SEED));
190     double reset_normal()
191     {
192         return randn(mu, sigma, CUSTOM_SEED);
193     }
194 
195     /* Initialize gamma to favour exploration
196     * set_gamma sets gamma to favour exploitation when function is run
197     */
198     double gamma_levy = GAMMA_MAX;
199     double set_gamma()
200     {
201         gamma_levy = 5;
202         return gamma_levy;
203     }
204 
205     /* Initialize Levy generator
206     * Check levy_distribution.c.c file for generator
207     */
208     short levy = levy_distribution(CUSTOM_SEED, gamma_levy);
209     short reset_levy()
210     {
211         return levy_distribution(CUSTOM_SEED, gamma_levy);
212     }
213 
214     /* Find first zero in targets_array of current robot */
215     short targets_array[AMOUNT_OF_TARGETS][3] = {0};
216     short findfirstzero(short ROWS, short targets_array[AMOUNT_OF_TARGETS][3])
217     {
218         for (int i = 0; i < ROWS; i++)
```

```
219         {
220             if (targets_array[i][0] == 0 && targets_array[i][1] == 0)
221             {
222                 return i;
223             }
224         }
225         return -1;
226     }
227 
228     /* Euclidean distance between two points (x1, y1) and (x2, y2) */
229     double euclidean_distance(double x1, double y1, double x2, double y2)
230     {
231         return sqrt(pow(x1 - x2, 2) + pow(y1 - y2, 2));
232     }
233 
234     /* Init local buffer for receiving targets by other robots
235      * Init array to send to other robots
236      * last_updated is a counter that helps enhance performance
237      */
238     short localbuffer[AMOUNT_OF_TARGETS * 2] = {0};
239     short targets_array_received[AMOUNT_OF_TARGETS][2] = {0};
240     short send[AMOUNT_OF_TARGETS * 2] = {0};
241     double last_updated[MAX_ROBOTS] = {0.0};
242 
243     ////////////////////////////////Receiver function////////////////////////////////
244     /* Fill targets_array[2] with newly aquired targets
245      * Calculate interaction state IS, based on the distance of other emitters
246      * This is complicated as robots only receive the different values of every emitter
247      * Emitter 1 can send [50,51,51,52]
248      * Emitter 2 can send [66,66,67,68]
249      * The receiver will receive [50,66,51,66,51,67,52,68]
250      * It needs to differentiate between the varying numbers, therefore a more complicated
           scheme is used
251      */
252     short signal_strength;
253     double robot_list[MAX_ROBOTS] = {0.0};
254     void get_receiver_data()
255     {
256         /* Is there at least one packet in the receiver's queue? */
257         if (wb_receiver_get_queue_length(communication_receiver) > 0)
258         {
259           const unsigned char *buffer = wb_receiver_get_data(communication_receiver);
260             signal_strength = sqrt(1 / wb_receiver_get_signal_strength(communication_receiver));
261             for (i = 0; i < MAX_ROBOTS; i++) {
262                 /* Check if the signal strength is 0 or 99 (indicating no signal or signal lost)
                        */
263                 if (signal_strength == 0 || signal_strength == 99) {
264                     robot_list[i] = 0; // No signal or signal lost, set the value to 0
265                     last_updated[i] = wb_robot_get_time(); // Update the last_updated timestamp
266                     break;
267                 }
268                 else {
269                     /* Assign the signal strength to the first empty slot */
270                     if (robot_list[i] == 0) {
271                         robot_list[i] = signal_strength;
272                         last_updated[i] = wb_robot_get_time();
273                         break;
274                     /* Update the existing robot's signal strength instead of creating a new
                            entry */
275                     }
276                     else if (abs(robot_list[i] - signal_strength) <= 1) {
```

```
277                     robot_list[i] = (robot_list[i] + signal_strength) / 2;
278                     last_updated[i] = wb_robot_get_time();
279                     break;
280                 }
281             }
282         }
283         /* Remove duplicates in robot_list */
284         for (i = 0; i < MAX_ROBOTS - 1; i++)
285         {
286             if (robot_list[i] == 0) {
287                 continue; // Skip empty slots
288             }
289             for (j = i + 1; j < MAX_ROBOTS; j++)
290             {
291                 if (robot_list[j] == 0) {
292                     continue; // Skip empty slots
293                 }
294                 if (robot_list[i] == robot_list[j]) {
295                     robot_list[j] = 0; // Duplicate found, set the value to 0
296                 }
297             }
298         }
299         /* Check if any value hasn't been updated in the last 10 seconds and set it to 0 */
300         double current_time = wb_robot_get_time();
301         for (i = 0; i < MAX_ROBOTS; i++) {
302             if (robot_list[i] != 0 && current_time - last_updated[i] > 10.0) {
303                 robot_list[i] = 0;
304             }
305         }
306         /* Copy buffer data into locally defined variable with amount of bytes to be copied,
                now n*2*2 (with short) */
307         memcpy(localbuffer, buffer, AMOUNT_OF_TARGETS*2*2);
308         /* Translate the singular array of localbuffer [targetsx + targetsy] --> [
               AMOUNT_OF_TARGETS][2]
309          * This is how the emitter sends the targets, not as an array but as a one-
               dimensional list
310          */
311         for (i = 0; i < AMOUNT_OF_TARGETS; ++i)
312         {
313             targets_array_received[i][0] = localbuffer[i * 2];
314             targets_array_received[i][1] = localbuffer[i * 2 + 1];
315         }
316         /* Find the first zero of the targets_array */
317         short first_zero_index = findfirstzero(AMOUNT_OF_TARGETS, targets_array);
318         /* The robots target_array is updated with that of the received target array
319          * First we check if the received targets are not already in our own list
320          */
321         for (short i = 0; i < AMOUNT_OF_TARGETS; i++)
322         {
323             short received_target_found = 0;
324             for (short j = 0; j < AMOUNT_OF_TARGETS; j++)
325             {
326                 if (targets_array_received[i][0] == targets_array[j][0] &&
                        targets_array_received[i][1] == targets_array[j][1])
327                 {
328                     received_target_found = 1;
329                     break;
330                 }
331             }
332             if (!received_target_found && (targets_array_received[i][0] != 0) && (
                    targets_array_received[i][1] != 0))
```

```
            {
                targets_array[first_zero_index][0] = targets_array_received[i][0];
                targets_array[first_zero_index][1] = targets_array_received[i][1];
                first_zero_index = -1;
                break;
            }
        }
        /* Receive next package */
        wb_receiver_next_packet(communication_receiver);

        /* Calculation of interction state, a value of 0 or emitter_range-1 means no signal
            again */
        int num_zeros = 0, num_non_zeros = 0;
        for (i = 0; i < MAX_ROBOTS - 1; i++)
        {
            if (robot_list[i] == 0 || robot_list[i] == EMITTER_RANGE-1)
            {
                num_zeros++;
            }
            else
            {
                num_non_zeros++;
            }
        }
        IS = num_non_zeros;
        //printf("robot_list[%d]\n",IS );
        // for (i = 0; i < MAX_ROBOTS; i++)
        // {
        // printf("index [%d] is robot_list[%f]\n",i, robot_list[i]);

        // }
    }

}
////////////////////////////////Emitter function////////////////////////////////
/* Send our own target_array list filled with targets to other robots
 * Update our operation state, which is how many targets have we found in the last 30
     minutes
 */
void send_emitter_data()
{
    /* Fill targets_array[2] with newly aquired targets*/
    wb_emitter_set_channel(communication, WB_CHANNEL_BROADCAST);
    bool target_found_exists = false;
    short target_index = -1;
    for (i = 0; i < AMOUNT_OF_TARGETS; i++)
    {
        if (xy_array_target[0] == targets_array[i][0] && xy_array_target[1] == targets_array
            [i][1])
        {
            target_found_exists = true;
            break;
        }
        else if (targets_array[i][0] == 0 && targets_array[i][1] == 0 && target_index == -1)
        {
            target_index = i;
        }
    }
    if (!target_found_exists && target_index != -1)
    {
        targets_array[target_index][0] = xy_array_target[0];
```

```
391             targets_array[target_index][1] = xy_array_target[1];
392             OS = OS + 1;
393             // printf("OS is equal to %d\n",OS);
394             for (i = 0; i < AMOUNT_OF_TARGETS; ++i)
395             {
396                 send[i * 2] = targets_array[i][0];
397                 send[i * 2 + 1] = targets_array[i][1];
398             }
399         }
400         wb_emitter_send(communication, send, AMOUNT_OF_TARGETS * 2 * sizeof(short)); //Changed
                from float to short
401     }
402
403
404     ////////////////////////////////Cluster centre function////////////////////////////////
405     /* Calculate the cluster centre to move away from after LSTT is reached*/
406     double average_cluster_position[2] = {0,0};
407     int lastClusterIndex = 0;
408     void calculate_average_position(short targets_array[AMOUNT_OF_TARGETS][3], double range,
            double average_cluster_position[2])
409     {
410     double cluster_distance_inbetween[AMOUNT_OF_TARGETS] = {0};
411     short sum_x = 0;
412     short sum_y = 0;
413     short count = 0;
414
415     // Ignore targets with coordinates (0, 0) and start from the last cluster index
416     for (int i = lastClusterIndex; i < AMOUNT_OF_TARGETS; i++) {
417         if (targets_array[i][0] == 0 && targets_array[i][1] == 0) {
418             continue;
419         }
420
421         for (int j = i + 1; j < AMOUNT_OF_TARGETS; j++) {
422             if (targets_array[j][0] == 0 && targets_array[j][1] == 0) {
423                 continue;
424             }
425
426             cluster_distance_inbetween[i] = euclidean_distance(targets_array[i][0],
                    targets_array[i][1], targets_array[j][0], targets_array[j][1]);
427             //printf("j is [%d] cluster_in_between is %f \n", j, cluster_distance_inbetween[i]);
428
429             if (cluster_distance_inbetween[i] <= range) {
430                 sum_x += targets_array[i][0] + targets_array[j][0];
431                 sum_y += targets_array[i][1] + targets_array[j][1];
432                 count += 2;
433             }
434         }
435
436         // Update the last cluster index after each iteration
437         lastClusterIndex = i;
438     }
439
440     // Calculate cluster centre position
441     if (count > 0) {
442         short average_x = sum_x / count;
443         short average_y = sum_y / count;
444         average_cluster_position[0] = average_x;
445         average_cluster_position[1] = average_y;
446         printf("Average position of targets within %.2f meters: (%.2d, %.2d)\n", range,
                average_x, average_y);
447     }
```

```
}
    ////////////////////////////Add to repel_list function//////////////////////////
    void addToRepelList(int last_cluster_position_x, int last_cluster_position_y) {

           // Check if the new obstacle position is too close to existing values
       for (int i = 0; i < MAX_REPEL_LIST; i++) {
           // Calculate the distance between the received last_obstacle_position and existing
               repel_list
           int last_cluster_distance = sqrt((repel_list[i][0] - last_cluster_position_x) * (
               repel_list[i][0] - last_cluster_position_x) +
                       (repel_list[i][1] - last_cluster_position_y) * (repel_list[i][1] -
                           last_cluster_position_y));

           if (last_cluster_distance < 2) {
              // If the distance is too small, ignore the new position
              return;
           }
       }
       // Shift the repel_list to the right
       for (int i = MAX_REPEL_LIST-1; i > 0; i--) {
           repel_list[i][0] = repel_list[i - 1][0];
           repel_list[i][1] = repel_list[i - 1][1];
           // Because of some weird error it adds a number randomly after the first shift. This
                solves it.
           if (repel_list[i][0] == 0 || repel_list[i][1] == 0)
           {
            repel_list[i][0] = 0;
            repel_list[i][1] = 0;
           }
       }
       // Add the new obstacle position at the beginning of the repel_list
       repel_list[0][0] = last_cluster_position_x;
       repel_list[0][1] = last_cluster_position_y;
       for (int i = 0; i < MAX_REPEL_LIST; i++)
       {
         printf("Cluster Repel List Updated [%d][%d]\n", repel_list[i][0], repel_list[i][1]);
       }

    }

    ////////////////////////////////Main loop////////////////////////////////
    ////////////////////////////////// :) ///////////////////////////////////
    ////////////////////////////////// :) ///////////////////////////////////
    while (wb_robot_step(TIME_STEP) != -1)
    {
       //printf("levy is %d\n",levy);
       // Get GPS position */
       const double *position = wb_gps_get_values(gps);
       double x_r = position[0];
       double y_r = position[1];
       /* Run helper functions
        * recognition updates xy_array_target which in turn updates targets_array if a target
            is found
        * get and receive your targets_array
        */
       recognition(xy_array_target);
       get_receiver_data();
       send_emitter_data();

       /* Used to print performance of robot during simulation */
       for (i = 0; i < AMOUNT_OF_TARGETS; i++)
```

```
504         {
505             int x = targets_array[i][0];
506             int y = targets_array[i][1];
507             int z = targets_array[i][2];
508             static int prev_x[AMOUNT_OF_TARGETS] = {0};
509             static int prev_y[AMOUNT_OF_TARGETS] = {0};
510             static int prev_z[AMOUNT_OF_TARGETS] = {0};
511             if (x != prev_x[i] || y != prev_y[i] || z != prev_z[i])
512             {
513                 printf("Onlooker is [%d], Repel is [%d], Gamma is [%f], OS is [%d], IS is [%d],
                        target [%d] is at = [%d,%d,%d] \n", onlooker_treshold, cluster_to_repel,
                        gamma_levy, OS, IS, i, x, y, z);
514                 prev_x[i] = x;
515                 prev_y[i] = y;
516                 prev_z[i] = z;
517             }
518         }
519
520         /* Get the battery percentage of the current robot */
521         BS = (wb_robot_battery_sensor_get_value() / 933120) * 100;
522
523         // Get time of synchronized simulation */
524         double time_robot = wb_robot_get_time();
525         double time_since_target = time_robot - last_target_time;
526         /* Slowly decline the operation state to represent the performance over half an hour */
527         if ((time_since_target > 30*60) && (OS!= 0) && !(time_since_target > 30.05*60))
528         {
529             OS-=1;
530         }
531         /* Get the yaw value of the current robot in radians*/
532         const double *ground_truth_attitude = wb_inertial_unit_get_roll_pitch_yaw(inertial_unit)
                ;
533         double yaw = ground_truth_attitude[2]; // Is in rad
534         /* Make sure the yaw value is always between [0,360], normalize function is found in
                file APF */
535         yaw = normalize_angle(yaw);
536          //printf("yaw is %f\n",yaw);
537
538         /* Calculate sensor influences on wheels
539          * Initialize memory at the beginning of the loop
540          */
541         memset(wheel_weight_total, 0, sizeof(double) * 4);
542         for (i = 0; i < MAX_SENSOR_NUMBER; ++i)
543         {
544             sensor_value = wb_distance_sensor_get_value(sensors[i].device_tag);
545             /* If the sensor doesn't see anything, we don't use it for this round */
546             if (sensor_value == 0.0)
547             {
548                 speed_modifier = 0.0;
549             }
550             else
551             {
552                 /* Computes the actual distance to the obstacle, given the value returned by the
                        sensor */
553                 distance = 5.0 * (1.0 - (sensor_value / MAX_SENSOR_VALUE)); // lookup table
                        inverse.
554                 /* Here we compute how much this sensor will influence the direction of the robot
                         */
555                 if (distance < MIN_DISTANCE_SENSOR)
556                     speed_modifier = 1 - (distance / MIN_DISTANCE_SENSOR);
557                 else
```

```
                speed_modifier = 0.0;
            }
            /* Add the modifier for all wheels */
            for (j = 0; j < 4; ++j)
            {
                wheel_weight_total[j] += sensors[i].wheel_weight[j] * speed_modifier;
            }
        }

        /* If a robot decides to go to a target, it calculates the closest one
         * It then sets the most suitable target to x_t and y_t
         * TARGET
         */
        double closest_d_target = 1000000; // initialize closest distance to a very large value
        int closest_i_target = -1; // initialize index of closest target to an invalid value
        for (int i = 0; i < AMOUNT_OF_TARGETS; i++)
        {
            int x_temp_target = targets_array[i][0];
            int y_temp_target = targets_array[i][1];
            int d_temp_target = euclidean_distance(x_r, y_r, x_temp_target, y_temp_target);
            if (d_temp_target < TARGET_RECOGNITION_TRESHOLD && (x_temp_target != 0 &&
                y_temp_target != 0) && d_temp_target < closest_d_target)
            {
                closest_d_target = d_temp_target;
                closest_i_target = i;
            }
        }
        if (closest_i_target != -1 && targets_array[closest_i_target][2] == 0)
        { // if a suitable target was found
            x_t = targets_array[closest_i_target][0];
            y_t = targets_array[closest_i_target][1];
        }

        if (x_t == 0 && y_t == 0)
        {
            distance_target = 0;
        }
        else
        {
            distance_target = euclidean_distance(x_r, y_r, x_t, y_t);
        }
        /* If a robot decides to go to a charging station, it calculates the closest one
         * It then sets the most suitable target to x_t and y_t
         * CHARGE
         */
        double closest_d_home = 1000000; // initialize closest distance to a very large value
        int closest_j_home = -1; // initialize index of closest target to an invalid value
        for (int j = 0; j < N_Chargers; j++)
        {
            int x_temp_home = home_array[j][0];
            int y_temp_home = home_array[j][1];
            int d_temp_home = euclidean_distance(x_r, y_r, x_temp_home, y_temp_home);
            if ((x_temp_home != 0 && y_temp_home != 0) && d_temp_home <= closest_d_home)
            {
                closest_d_home = d_temp_home;
                closest_j_home = j;
            }
        }
        if (closest_j_home != -1)
        { // if a suitable target was found
            x_h = home_array[closest_j_home][0];
```

```
618             y_h = home_array[closest_j_home][1];
619         }
620
621         /* Decide if a robot should become an onlooker or just keep scouting
622          * First the onlooker_treshold returns a true, to become an onlooker or false to stay
                scout
623          * When a robot is a onlooker and becomes a scout, gamma_levy=GAMMA_MAX to favour global
                search
624          */
625         onlooker_treshold = fuzzy_treshold(AMOUNT_OF_TARGETS, OS, IS);
626         if (distance_target < TARGET_RECOGNITION_TRESHOLD && distance_target > 5 &&
                distance_target != 0 && targets_array[closest_i_target][2] != 1 &&
                onlooker_treshold && (BS >= BS_TRESHOLD))
627         {
628             onlooker = 1;
629         }
630         else if(!onlooker_treshold || targets_array[closest_i_target][2] == 1 )
631         {
632             onlooker = 0;
633         }
634         /* In experienced forager mode (After finding a target),
635          * gamma_levy is slowly increased from exploiting to exploring
636          * After LSTT is reached the value goes to GAMMA_MAX rather quickly
637          * Additionally, if LSTT is reached, onlooker is set to 0 (to become a scout
638          * THis often happens in clusteredf areas, thus the robot is reppelled from the cluster
639          */
640         if (gamma_levy <= GAMMA_MAX-0.00035 && time_since_target < LOCAL_SEARCH_TIME_TRESHOLD
                && time_since_target != time_robot)
641         {
642             gamma_levy += 0.00035;
643             if(!onlooker_treshold)
644             {
645                 gamma_levy += 0.003;
646             }
647
648         }
649         else if (gamma_levy < GAMMA_MAX && time_since_target >= LOCAL_SEARCH_TIME_TRESHOLD)
650         {
651             if(time_since_target < LOCAL_SEARCH_TIME_TRESHOLD+1)
652             {
653                 onlooker = 0;
654                 gamma_levy = GAMMA_MAX;
655                 calculate_average_position(targets_array, RANGE_OF_CLUSTER,
                        average_cluster_position);
656                 printf("average_cluster_position is at [%f, %f] with gamma %f\n",
                        average_cluster_position[0], average_cluster_position[1], gamma_levy);
657                 if(average_cluster_position[0] != 0 && average_cluster_position[1]!= 0)
658                 {
659                     addToRepelList((int) average_cluster_position[0], (int)
                            average_cluster_position[1]);
660                 }
661             }
662         }
663
664         // We get the cluster that is closest to us, index cluster_to_repel from repel_list
665         for (int i = 0; i < MAX_REPEL_LIST; i++)
666         {
667             if (repel_list[i][0] != 0 && repel_list[i][1] != 0)
668             {
669                 distance_to_clusters[i] = sqrt((x_r - repel_list[i][0]) * (x_r - repel_list[i
                        ][0]) +
```

```
670                 (y_r - repel_list[i][1]) * (y_r - repel_list[i][1]));
671             }
672         }
673         int min_distance = size_field*2;
674         for (int i = 0; i < MAX_REPEL_LIST; i++)
675         {
676             if (distance_to_clusters[i] <= MIN_POTENTIAL_DISTANCE && distance_to_clusters[i] !=
                    0)
677             {
678                 if (distance_to_clusters[i] < min_distance)
679                 {
680                     cluster_to_repel = i;
681                 }
682             }
683         }
684 
685         // If closest_index == -1, there is nothing to repel, onlooker can be true
686         if (distance_to_clusters[cluster_to_repel] > MIN_POTENTIAL_DISTANCE)
687         {
688            cluster_to_repel = -1;
689         }
690         // If the distance between the robot and the wall is larger then TRT, wall = 0
691         if (euclidean_distance(x_r,y_r, wall_pos[0], wall_pos[1]) > TARGET_RECOGNITION_TRESHOLD)

692         {
693             wall = 0;
694         }
695         /* Calculate the force vector when an obstacle or cluster is recognized*/
696 
697         /* STATE MACHINE
698          *
699          * RESET --> Resets levy and normal distribution numbers and add current time to
                simulation
700          * LEVY --> If no obstacles are found then: Go straight with step from levy distribution
                ,
701          * then turn left or right depending on normal distributed number for time
702          * that is also dependent on normal distribution
703          * GOTO --> Goes to the first target that has been found by the robots
704          * HOMING --> If BS < BS_treshold, go home to charg
705          * FORWARD --> This state decides if the robot should turn left or right depending on
                sensory information
706          * LEFT --> Turn left
707          * RIGHT --> Turn right
708          */
709         ////////////////////////////////State Machine////////////////////////////////
710         /* RESET
711          * Resets time
712          * Returns to LEVY
713          * Sends and receives target arrays
714          */
715         //printf("state is %d\n",state);
716         //printf("levy is [%d], time_robot is [%f] normal is [%f], state is [%d]\n", levy,
                time_robot, normal, state);
717         switch (state)
718         {
719         case RESET:
720             levy = reset_levy() + time_robot;
721             // Number is even
722             mu = (double)rand()/ RAND_MAX * (2.0 * M_PI); // Scale to the desired range
723             normal = normalize_angle(reset_normal());
724             if (normal>360)
```

```
725             {
726                 normal = normal-360;
727             }
728             state = LEVY;
729             break;
730             //printf("RESET\n");
731         break;
732 
733         /* Levy walker
734          * Onlooker = 0
735          */
736         case LEVY:
737             // OBSTACLE AVOIDANCE
738             if((BS >= BS_TRESHOLD) &&
739             (wheel_weight_total[0] > WHEEL_WEIGHT_THRESHOLD || wheel_weight_total[1] >
                    WHEEL_WEIGHT_THRESHOLD))
740             {
741                    state = AVOID;
742                    break;
743             }
744             // NO OBSTACLES
745             else if (wheel_weight_total[0] == 0 || wheel_weight_total[1] == 0)
746             {
747                 // TARGET RECOGNITION
748                 if (onlooker && (x_t != 0 && y_t != 0) && (BS >= BS_TRESHOLD))
749                 {
750                     state = GOTO;
751                     break;
752                 }
753                 //RESETTING STEP GENERATOR
754                 if (!onlooker && (BS >= BS_TRESHOLD) &&
755                 (time_robot >= levy + abs(normal)))
756                 {
757                     state = RESET;
758                     break;
759                 }
760                 // BATTERY MONITERING
761                 else if (BS < BS_TRESHOLD)
762                 {
763                     state = HOMING;
764                     break;
765                 }
766                 // FORWARD STEP
767                 else if (!onlooker && (BS >= BS_TRESHOLD) && (time_robot < levy))
768                 {
769                    // Option 1: We repel with a set angle away from cluster, or wall
770                    if ((cluster_to_repel != -1) || (wall && distance_target >
                           TARGET_RECOGNITION_TRESHOLD))
771                    {
772                        get_receiver_data();
773                        send_emitter_data();
774                        Force_vector(MIN_POTENTIAL_DISTANCE, repel_list, cluster_to_repel, x_r, y_r
                               , 0, 0);
775                        calculateWheelSpeeds(repulsion_angle_tot, yaw, speed);
776                    }
777                    // Option 2: We repel away from wall, towards a unvisisted target closert then
                           TRT
778                    else if(wall && distance_target < TARGET_RECOGNITION_TRESHOLD && targets_array
                           [closest_i_target][2] != 1)
779                    {
780                        get_receiver_data();
```

```
781                     send_emitter_data();
782                     Force_vector(MIN_POTENTIAL_DISTANCE, repel_list, cluster_to_repel, x_r, y_r
                            , x_t, y_t);
783                     calculateWheelSpeeds(attraction_angle_atr, yaw, speed);
784                 }
785                 // Option 3: Nothing to repel, straight step
786                 else
787                 {
788                     get_receiver_data();
789                     send_emitter_data();
790                     speed[0] = MAX_SPEED;
791                     speed[1] = MAX_SPEED;
792                     speed[2] = MAX_SPEED;
793                     speed[3] = MAX_SPEED;
794                 }
795             }
796             // NORMAL ROTATION
797             else if ((!onlooker && !(wall) && (BS >= BS_TRESHOLD) && (time_robot > levy)) ||
                    (!onlooker && (BS >= BS_TRESHOLD) && (time_robot < levy + 2) && !wall))
798             {
799             //printf("normal is %f, yaw is %f, normal-yaw is %d\n", normal, yaw, abs(normal-
                    yaw));
800                 if (normal > normalize_angle(mu) && abs(normal- yaw) > 20)
801                 {
802                     speed[0] = 0.7 * MAX_SPEED;
803                     speed[1] = -0.7 * MAX_SPEED;
804                     speed[2] = 0.7 * MAX_SPEED;
805                     speed[3] = -0.7 * MAX_SPEED;
806                 }
807                 else if (normal <= normalize_angle((mu)) && abs(normal- yaw) > 20)
808                 {
809                     speed[0] = -0.7 * MAX_SPEED;
810                     speed[1] = 0.7 * MAX_SPEED;
811                     speed[2] = -0.7 * MAX_SPEED;
812                     speed[3] = 0.7 * MAX_SPEED;
813                 }
814                 else if(abs(normal-yaw) < 20 || (time_robot > levy + 5))
815                 {
816                     state = RESET;
817                 break;
818                 }
819             }
820         }
821     break;
822     /* GOTO
823     * We have a goal to go towards, how do we deal with obstacles?
824     */
825     case GOTO:
826         if (!(onlooker))
827         {
828             onlooker = 0;
829             levy = time_robot;
830             last_target_time = time_robot;
831             set_gamma();
832             state = RESET;
833             break;
834         }
835         else if (onlooker && targets_array[closest_i_target][2] != 1)
836         {
837             // BATTERY MONITORING
838             if (BS < BS_TRESHOLD)
```

```
839                 {
840                     state = HOMING;
841                     break;
842                 }
843                 // OBSTACLE AVOIDANCE
844                 if (((wheel_weight_total[0] > WHEEL_WEIGHT_THRESHOLD || wheel_weight_total[1] >
                        WHEEL_WEIGHT_THRESHOLD))) {
845                     state = AVOID;
846                     break;
847                 }
848                 // Nothing to repel, so just go towards the target
849                 if ((wheel_weight_total[0] < WHEEL_WEIGHT_THRESHOLD) && (wheel_weight_total[1] <
                        WHEEL_WEIGHT_THRESHOLD) && distance_target > 0)
850                 {
851                     get_receiver_data();
852                     send_emitter_data();
853                     Force_vector(MIN_POTENTIAL_DISTANCE, repel_list, cluster_to_repel, x_r, y_r,
                            x_t, y_t);
854                     calculateWheelSpeeds(attraction_angle_atr, yaw, speed);
855                     if (distance_target < 5)
856                     {
857                         set_gamma();
858                         last_target_time = time_robot;
859                         targets_array[closest_i_target][2] = 1;
860                         onlooker = 0;
861                         levy = time_robot;
862                         state = RESET;
863                         break;
864                     }
865                 }
866             }
867         break;
868         /* HOMING
869         * Onlooker = 1
870         */
871         case HOMING:
872             if (!home_reached)
873             {
874                 Force_vector(MIN_POTENTIAL_DISTANCE, repel_list, cluster_to_repel, x_r, y_r, x_h,
                        y_h);
875                 calculateWheelSpeeds(attraction_angle_atr, yaw, speed);
876             }
877             else if (!(BS = 100))
878             {
879                 speed[0] = 0;
880                 speed[1] = 0;
881                 speed[2] = 0;
882                 speed[3] = 0;
883             }
884             else
885             {
886                 levy = time_robot;
887                 last_target_time = time_robot;
888                 state = RESET;
889             }
890         break;
891         /* AVOID
892          * Decide to turn LEFT or RIGHT
893          */
894         case AVOID:
895           if (x_r > size_field-5 || y_r > size_field-5 || x_r < -size_field+5 || y_r < -
```

```
            size_field+5)
            {
                wall_pos[0] = (int) x_r;
                wall_pos[1] = (int) y_r;
                wall = 1;
                //printf("Wall is found at [%d,%d]\n",wall_pos[0],wall_pos[1]);
            }
        /* Left sensors are over treshold --> turn left */
        if (wheel_weight_total[0] > WHEEL_WEIGHT_THRESHOLD)
        {
            state = LEFT;
            break;
        }
        /* Right sensors are over treshold --> turn right */
        else if (wheel_weight_total[1] > WHEEL_WEIGHT_THRESHOLD)
        {
            state = RIGHT;
            break;
        }
        /* Go back to LEVY walking */
        else
        {
            state = LEVY;
            break;
        }
    break;
    /* LEFT
     * When the robot has started turning, it will go on in the same direction until no more obstacle.
     * This will prevent the robot from being caught in a loop going left, then right, and so on.
     */
    case LEFT:
        if (wheel_weight_total[0] > WHEEL_WEIGHT_THRESHOLD || wheel_weight_total[1] > WHEEL_WEIGHT_THRESHOLD)
        {
            speed[0] = 0.7 * MAX_SPEED;
            speed[1] = -0.7 * MAX_SPEED;
            speed[2] = 0.7 * MAX_SPEED;
            speed[3] = -0.7 * MAX_SPEED;
        }
        else if (wheel_weight_total[0] == 0)
        {
            state = LEVY;
            break;
        }
    break;
        /* RIGHT
         * When the robot has started turning, it will go on in the same direction until no more obstacle.
         * This will prevent the robot from being caught in a loop going left, then right, and so on.
         */
    case RIGHT:
        if (wheel_weight_total[0] > WHEEL_WEIGHT_THRESHOLD || wheel_weight_total[1] > WHEEL_WEIGHT_THRESHOLD)
        {
            speed[0] = -0.7 * MAX_SPEED;
            speed[1] = 0.7 * MAX_SPEED;
            speed[2] = -0.7 * MAX_SPEED;
            speed[3] = 0.7 * MAX_SPEED;
```

```
950             }
951             else if (wheel_weight_total[1] == 0)
952             {
953                 state = LEVY;
954                 break;
955             }
956         break;
957         /* Default state of state machine */
958         default:
959             state = LEVY;
960         }
961         /* Init motors velocity from obstacles seen */
962         wb_motor_set_velocity(front_left_wheel, speed[0]);
963         wb_motor_set_velocity(front_right_wheel, speed[1]);
964         wb_motor_set_velocity(back_left_wheel, speed[2]);
965         wb_motor_set_velocity(back_right_wheel, speed[3]);
966 
967         if ((BS < 3 && (!has_printed)) || ((wb_robot_get_time() > 2*3600 + 10*60) && (!
                has_printed)))
968         //if (((wb_robot_get_time() > 2*3600 + 5*60) && (!has_printed)))
969         //if( (((BS < 3) && (wb_robot_get_time() > (2*3600+ 10*60))) && (!has_printed) )|| ((
                wb_robot_get_time() > (2*3600+ 10*60))&& (!has_printed)) )
970         {
971             for (i = 0; i < AMOUNT_OF_TARGETS; i++)
972             {
973                 printf("target [%d] is at = %d,%d,%d \n", i, targets_array[i][0], targets_array[i
                        ][1], targets_array[i][2]);
974             }
975             char filename[40];
976             time_t current_time = time(NULL);
977             struct tm *local_time = localtime(&current_time); // Convert to local time
978             strftime(filename, sizeof(filename), "targets_data_%Y%m%d_%H%M%S.txt", local_time);
                    // Format the timestamp and append to filename
979             // sprintf(filename, "targets_data_%d.txt", CUSTOM_SEED);
980             save_targets_data(targets_array, filename); // Save the data to the new filename
981             has_printed = true;
982         }
983     }
984     wb_robot_cleanup();
985     return 0;
986 }
```

B.1. Artifical potential function

The code for the artificial potential function used in modified bee swarm optimization is given next. In addition, the yaw angles are normalized between $[0, 2\pi]$.

```
1  #include <time.h>
2  #include <stdlib.h>
3  #define MAX_SPEED 6.4
4  #define MAX_REPEL_LIST 10
5  double k_b = 5000; // Repulsion gain coefficient
6  double k_a = 1; // Attraction gain coefficient
7  double x_robot_to_obstacle = 0; // position robot_x - position target_x
8  double y_robot_to_obstacle = 0; // position robot_y - postition target_y
9  double x_robot_to_goal = 0;
10 double y_robot_to_goal = 0;
11 double Frep_x = 0;
12 double Frep_y = 0;
13 double Fatt_x = 0;
14 double Fatt_y = 0;
```

```
15 double Ftot_x = 0;
16 double Ftot_y = 0;
17 double pos_goal_x = 0;
18 double pos_goal_y = 0;
19 double magnitude = 0;
20 double repulsion_angle_tot = 0;
21 double repulsion_angle_rep = 0;
22 double attraction_angle_atr = 0;
23 short cluster_to_repel = -1;
24 /* Make sure the yaw value is always between [0,360], is used for yaw determination in main
25  * and the determination of the repulsion_angle in Force_vector in current file.
26  * rad2deg!
27  */
28 double normalize_angle(double angle_to_normalize)
29 {
30   if (angle_to_normalize > 2*M_PI)
31   {
32     angle_to_normalize = (angle_to_normalize -2*M_PI) * 180 / M_PI;
33   }
34   else if (angle_to_normalize < 0)
35   {
36     angle_to_normalize = (angle_to_normalize + 2*M_PI) * 180 / M_PI;
37   }
38    // else if (angle_to_normalize < 2*M_PI)
39   // {
40     // angle_to_normalize = (angle_to_normalize + 2*M_PI) * 180 / M_PI;
41   // }
42   else
43   {
44     angle_to_normalize = (angle_to_normalize)*180 / M_PI;
45   }
46
47   return (double)angle_to_normalize;
48   // return angle_to_normalize*180 / M_PI;
49 }
50
51
52 /* Calculates the force vector for the artifical potential field by getting the robot position
53 * receiving points to be reppeled from (or clusters or obstacles), and generating random goal
       positions
54 * to be attracted to.
55 * repel_list[][2] contains a list of [repel_list][2] coordinates. So repel_list[0][2] is t
56 */
57
58
59 int previous_index;
60 // MAY WANT TO REMOVE DISTANCE_REPEL_LIST!
61 void Force_vector(int MIN_POTENTIAL_DISTANCE, int repel_list[][2], short cluster_to_repel,
       double x_robot_position, double y_robot_position, double pos_goal_x, double pos_goal_y)
62 {
63    //pos_goal_x = 0; // Declare pos_goal_x as a static variable
64    //pos_goal_y = 0; // Declare pos_goal_y as a static variable
65    // closest_index changed, update pos_goal_x and pos_goal_y
66    if (pos_goal_x == 0 && pos_goal_y == 0 && cluster_to_repel != -1)
67    {
68        if (repel_list[cluster_to_repel][0] > 0 && repel_list[cluster_to_repel][1] > 0 )
69        {
70            // pos_goal_x = x_robot_position - 300;
71            // pos_goal_y = y_robot_position - 300;
72            attraction_angle_atr = 225*M_PI / 180;
73        }
```

```
74        else if (repel_list[cluster_to_repel][0] < 0 && repel_list[cluster_to_repel][1] < 0 )
75        {
76            // pos_goal_x = x_robot_position + 300;
77            // pos_goal_y = y_robot_position + 300;
78            attraction_angle_atr = 45*M_PI / 180;
79        }
80        else if (repel_list[cluster_to_repel][0] < 0 && repel_list[cluster_to_repel][1] > 0 )
81        {
82            // pos_goal_x = x_robot_position + 300;
83            // pos_goal_y = y_robot_position - 300;
84            attraction_angle_atr = 315*M_PI / 180;
85        }
86        else if (repel_list[cluster_to_repel][0] > 0 && repel_list[cluster_to_repel][1] < 0 )
87        {
88            // pos_goal_x = x_robot_position - 300;
89            // pos_goal_y = y_robot_position + 300;
90            attraction_angle_atr = 135*M_PI / 180;
91        }
92
93    }
94
95    x_robot_to_goal = x_robot_position-pos_goal_x;
96    y_robot_to_goal = y_robot_position-pos_goal_y;
97    Fatt_x = - k_a * (x_robot_to_goal);
98    Fatt_y = - k_a * (y_robot_to_goal);
99    double total_repulsion_x = 0;
100   double total_repulsion_y = 0;
101   for (int i = 0; i < MAX_REPEL_LIST; i++) {
102       x_robot_to_obstacle = x_robot_position - repel_list[i][0];
103       y_robot_to_obstacle = y_robot_position - repel_list[i][1];
104       double distance_obstacle_robot = sqrt(x_robot_to_obstacle * x_robot_to_obstacle +
              y_robot_to_obstacle * y_robot_to_obstacle);
105       if (distance_obstacle_robot < MIN_POTENTIAL_DISTANCE) {
106           Frep_x = k_b * (((1 / distance_obstacle_robot) - (1 / 10)) *
107               (1 / distance_obstacle_robot) * (1 / distance_obstacle_robot) *
108               (x_robot_to_obstacle / distance_obstacle_robot));
109
110           Frep_y = k_b * (((1 / distance_obstacle_robot) - (1 / 10)) *
111               (1 / distance_obstacle_robot) * (1 / distance_obstacle_robot) *
112               (y_robot_to_obstacle / distance_obstacle_robot));
113
114           total_repulsion_x += Frep_x;
115           total_repulsion_y += Frep_y;
116       }
117       else
118       {
119           Frep_x=Frep_y=0;
120       }
121    }
122    Ftot_x = Fatt_x + total_repulsion_x;
123    Ftot_y = Fatt_y + total_repulsion_y;
124    if (pos_goal_x == 0 && pos_goal_y == 0 && cluster_to_repel != -1)
125    {
126        attraction_angle_atr = attraction_angle_atr;
127    }
128    else
129    {
130        attraction_angle_atr = normalize_angle(atan2(Fatt_y,Fatt_x))*M_PI / 180;
131    }
132
133    repulsion_angle_rep = normalize_angle(atan2(Frep_y,Frep_x))*M_PI / 180;
```

```
134     repulsion_angle_tot = attraction_angle_atr+repulsion_angle_rep;
135 
136     if (cluster_to_repel == -1)
137     {
138        x_robot_to_obstacle = 0;
139        y_robot_to_obstacle = 0;
140        x_robot_to_goal = 0;
141        y_robot_to_goal = 0;
142     }
143 }
144 
145 /* Using the repulsion angle calculated by taking the cartesian components of the Force Vector,
    * a yaw angle is decided upon. This angle will influence the wheel speeds, reducing one side
    * to minimize the difference of the angle.
    */
149 void calculateWheelSpeeds(double repulsion_angle, double yaw, double *speed)
150 {
151     double angle_diff = repulsion_angle - yaw * M_PI / 180;
152     // Normalize the angle difference to be within the range [-pi, pi] radians
153     if (angle_diff >= M_PI)
154     {
155        angle_diff -= 2 * M_PI;
156     }
157     else if (angle_diff < -M_PI)
158     {
159        angle_diff += 2 * M_PI;
160     }
161     //printf(" angle_diff is %f\n", angle_diff);
162     // Calculate the wheel speeds based on the angle difference
163     speed[0] = MAX_SPEED; // Left front
164     speed[1] = MAX_SPEED; // Right front
165     speed[2] = MAX_SPEED; // Left back
166     speed[3] = MAX_SPEED; // Right back
167 
168     // Reduce the speed of the wheels on one side based on the angle difference
169     double reduction_factor = 1.0 - fabs(angle_diff) / (0.5 * M_PI); // Modify the reduction
        factor
170     if (reduction_factor < 0.0)
171     {
172        reduction_factor = 0.0;
173     }
174     if (angle_diff >= 0)
175     {
176        speed[0] *= reduction_factor; // Decrease left front speed
177        speed[2] *= reduction_factor; // Decrease right back speed
178     }
179     else if (angle_diff < 0)
180     {
181        speed[1] *= reduction_factor; // Decrease right front speed
182        speed[3] *= reduction_factor; // Decrease left back speed
183     }
184 }
```

B.2. Implementation of AMEB

This function implements AMEB in MBSO. For the current simulations, only IS is used.

```
int prev_OS = -1, prev_IS = -1, prev_treshold = -1;
bool fuzzy_treshold(int AMOUNT_OF_TARGETS, short OS, short IS)
{
   //printf("value of IS is %d\n" ,IS);
   bool onlooker = false;
   int treshold = 0;
   int IS_TRESHOLD = 2;
   int OS_TRESHOLD = 2;

   int IS_value, OS_value;

   if (IS > IS_TRESHOLD) // Triggered when IS = 2,3,4,5,6
   {
      IS_value = -1;
   }
   else if (IS <= IS_TRESHOLD) // Triggered when IS = 0,1
   {
      IS_value = 1;
   }
      if (OS < OS_TRESHOLD)
   {
      OS_value = -1;
   }
   else if (OS >= OS_TRESHOLD)
   {
      OS_value = 1;
   }

   //treshold = BS_value + SS_value + IS_value;
     treshold = IS_value;
   if (prev_OS != OS_value || prev_IS != IS_value || prev_treshold != treshold)
   {
      prev_IS = IS_value;
      prev_OS = OS_value;
      prev_treshold = treshold;
   }

   ////////////////////////
   if (treshold < 0)
   {
      onlooker = false;
   }
   else if (treshold >= 0)
   {
      onlooker = true;
   }

   return onlooker;
}
```

B.3. Normal generator

Next, the normal generator is shown. This function generates a random angle between $[0, 2\pi]$

```c
#include <math.h>
#include <stdlib.h>
#include <time.h>

double randn(double mu, double sigma, int custom_seed)
{
  double U1, U2, W, mult;
  static double X1, X2;
  static int call = 0;
  srand(time(0) + custom_seed);
  if (call == 1)
  {
    call = !call;
    double normal1= (mu + sigma * (double)X2);
    //printf("normal1 is %f\n", normal1);
    return (normal1);
  }

  do
  {
    U1 = -1 + ((double)rand() / RAND_MAX) * 2;
    U2 = -1 + ((double)rand() / RAND_MAX) * 2;
    W = pow(U1, 2) + pow(U2, 2);
  }
  while (W >= 1 || W == 0);
  mult = sqrt((-2 * log(W)) / W);
  X1 = U1 * mult;
  X2 = U2 * mult;

  call = !call;
  double normal2=(mu + sigma * (double)X1);
    //printf("normal2 is %f\n", normal2);
  return (normal2);


}
```

B.4. Lévy generator

The Lévy generator returns a Lévy step size based on the Lévy distribution. Additionally, a random number is generated using the Box-Muller transform, which is used in the Lévy generator.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
double randomBoxMuller()
{
    double u1 = rand() / (double)RAND_MAX; // Generate a random number between 0 and 1
    double u2 = rand() / (double)RAND_MAX; // Generate another random number between 0 and 1

    double z1 = sqrt(-2.0 * log(u1)) * cos(2.0 * M_PI * u2); // Apply Box-Muller transform
    double z2 = sqrt(-2.0 * log(u1)) * sin(2.0 * M_PI * u2); // Apply Box-Muller transform

    // Transform the generated numbers to the desired range
    z1 = (z1 + 3) / 6; // Adjust the mean and standard deviation
    z2 = (z2 + 3) / 6; // Adjust the mean and standard deviation

    // Return one of the generated numbers
```

```
    return z1;
}


short levy_distribution(int seed, double gamma_levy)
{ double u= randomBoxMuller();
    //double u = ((double)rand() / RAND_MAX); // Generate a random number between 0 and 1
    double levy = gamma_levy*pow(u, -1.919);
    if (levy >= (gamma_levy/33)*800)
    {
        levy = (gamma_levy/33)*800;
    }
    //printf("gamma_levy is %f [constant], levy is[stepsize] %f\n", gamma_levy, levy);
    return levy;
}
```

B.5. Recognition

Robots recognize targets by using object detection. It is assumed that robots can recognize targets at 15 m distance. The robot uses a fictional camera with a field of view of 1.05, a pixel density of 100x100 pixels (to improve simulation speed), and anti-aliasing. Targets are recognized by the camera and are tagged at their relative position to the camera. The yaw of the robot, with the GPS position, is then used to pinpoint the location of the targets.

```
#include <stdio.h>
#include <webots/camera.h>
#include <webots/camera_recognition_object.h>
#include <webots/inertial_unit.h>
#include <webots/gps.h>
#define TIME_STEP 64

void recognition(short arr[])
{
  // Get devicetags from robot
  WbDeviceTag camera = wb_robot_get_device("camera");
  WbDeviceTag gps = wb_robot_get_device("gps");
  WbDeviceTag inertial_unit = wb_robot_get_device("inertial unit");

  // Enable all units
  wb_camera_enable(camera, TIME_STEP);
  wb_camera_recognition_enable(camera, TIME_STEP);
  wb_gps_enable(gps, TIME_STEP);
  wb_inertial_unit_enable(inertial_unit, TIME_STEP);

  // Determine yaw
  const double *ground_truth_attitude = wb_inertial_unit_get_roll_pitch_yaw(inertial_unit);
  double yaw = ground_truth_attitude[2]; // is in radian

  // Get current number of object recognized
  int i;
  int number_of_objects = wb_camera_recognition_get_number_of_objects(camera);
  const WbCameraRecognitionObject *objects = wb_camera_recognition_get_objects(camera);

  // Get position of found objects
  for (i = 0; i < number_of_objects; ++i)
  {
    const double *position = wb_gps_get_values(gps);

    // Global position of the robot
    double x_global_robot = position[0];
    double y_global_robot = position[1];
```

```
38
39      // Relative position of target to robot
40      double sideways_distance_target = objects[i].position[0];
41      double depth_distance_target = objects[i].position[1];
42      double angle_robot_target = atan2(depth_distance_target, sideways_distance_target);
43      double target_vector = sqrt(depth_distance_target * depth_distance_target +
            sideways_distance_target * sideways_distance_target);
44
45      // Get target position and get global x and y vectors from them
46      double global_robot_angle = yaw + angle_robot_target - 3.14;
47      arr[0] = round(x_global_robot - cos(global_robot_angle) * target_vector);
48      arr[1] = round(y_global_robot - sin(global_robot_angle) * target_vector);
49      // printf("Model %s, Id %d\n\n", objects[i].model ,objects[i].id);
50      // printf("gps_robot: %lfm %lfm \n", x_global_robot, y_global_robot);
51      // printf("global_target x, y in m new: %lfm %lfm \n", round(arr[0]), round(arr[1]));
52    }
53  }
```

B.6. Saving files

The following script is used to save the target array list to the controller map. Files are recognized by a timestamp.

```
1  #include <stdio.h>
2  #include <stdbool.h>
3  #define AMOUNT_OF_TARGETS 34
4
5  void save_targets_data(short targets_array[AMOUNT_OF_TARGETS][3], const char *filename)
6  {
7      // Open file for writing
8      FILE *file = fopen(filename, "w");
9      if (!file)
10     {
11         printf("Error: Unable to open file for writing\n");
12         return;
13     }
14     // Write data to file
15     for (int i = 0; i < AMOUNT_OF_TARGETS; i++)
16     {
17         fprintf(file, "%d,%d,%d\n", targets_array[i][0], targets_array[i][1], targets_array[i
               ][2]);
18     }
19     // Close file
20     fclose(file);
21     printf("Data saved to file: %s\n", filename);
22 }
```

B.7. Supervisor

The supervisor function Webots enables to automatize the simulations. The following script is used to automatically randomize, set the battery state and reset the robots and simulation.

```
1  #include <webots/supervisor.h>
2  #include <stdlib.h>
3  #include <time.h>
4  #include <stdio.h>
5  #include <webots/robot.h>
6  #define FIELD_SIZE 1000.0
7  #define MAX_ROBOTS 10
8  #define TIME_STEP 64
```

```
9
10 int main(int argc, char **argv) {
11   WbNodeRef robot_node[MAX_ROBOTS];
12   WbFieldRef trans_field[MAX_ROBOTS];
13   WbFieldRef rotation_field[MAX_ROBOTS];
14   WbFieldRef battery_field[MAX_ROBOTS];
15   srand(time(NULL)); // initialize random seed
16   wb_robot_init();
17
18   // Place the robots randomly within the field
19   for (int i = 0; i < MAX_ROBOTS; i++) {
20     char node_name[15];
21     sprintf(node_name, "Scout_%d", i + 1);
22     robot_node[i] = wb_supervisor_node_get_from_def(node_name);
23     trans_field[i] = wb_supervisor_node_get_field(robot_node[i], "translation");
24     double x = (int) rand() % 1200 - 600;
25     double y = (int) rand() % 1200 - 600;
26     double rot = rand() % 200*M_PI - 100*M_PI;
27     wb_supervisor_field_set_sf_vec3f(trans_field[i], (const double[]){x, y, 0.01});
28     battery_field[i] = wb_supervisor_node_get_field(robot_node[i], "battery");
29     wb_supervisor_field_set_mf_float(wb_supervisor_node_get_field(robot_node[i], "battery")
           ,0,933120);
30     rotation_field[i] = wb_supervisor_node_get_field(robot_node[i], "rotation");
31     double rotation[4] = {0, 0, 1, rot/100}; // random axis-angle rotation
32     wb_supervisor_field_set_sf_rotation(rotation_field[i], rotation);
33   }
34
35   while (wb_robot_step(TIME_STEP) != -1) {
36     // Check if all battery values are zero
37     double battery_values[MAX_ROBOTS];
38     bool all_batt_empty = true;
39     for (int i = 0; i < MAX_ROBOTS; i++) {
40       battery_values[i] = wb_supervisor_field_get_mf_float(battery_field[i], 0);
41       if (battery_values[i] != 0.0) {
42         all_batt_empty = false;
43       }
44     }
45
46     // Perform actions if all battery values are zero
47     if ((all_batt_empty || (wb_robot_get_time() > (2*3600+ 15*60) && wb_robot_get_time() <
           (2*3600 + 15.2*60))))
48     {
49       char filename[40];
50       time_t current_time = time(NULL);
51       struct tm *local_time = localtime(&current_time); // Convert to local time
52       strftime(filename, sizeof(filename), "screenshot_%Y%m%d_%H%M%S.png", local_time); //
             Format the timestamp and append to filename
53       wb_supervisor_export_image(filename, 50); // save screenshot of current view
54       wb_supervisor_world_save(NULL);
55       wb_supervisor_world_reload();
56       for (int i = 0; i < MAX_ROBOTS; i++) {
57         wb_supervisor_field_set_mf_float(wb_supervisor_node_get_field(robot_node[i], "battery"),
               0, 933120);
58       }
59     }
60   }
61   wb_robot_cleanup();
62   return 0;
63 }
```

C

Literature review

Literature Review

Multi-target search in unknown environments using swarm robotics

Ramin Ariana

Delft University of Technology

TUDelft

Literature Review

Multi-target search in unknown environments using swarm robotics

by

Ramin Ariana

Full name	Study	Student Number
Ramin Ariana	MSc. Mechanical Engineering- Biorobotics	5181585

Project Coach: Prof. dr. ir. J Hellendoorn TU Delft

TU Delft

Abstract

Exploring an unknown environment to find multiple targets is one of the problems researchers are trying to solve with Swarm Robotics (SR). Swarming algorithms are used in much more than robotics, for example (un)constrained optimization, multi-solution problems, multi-objective problems and dynamic optimization problems. However, in this paper, only SR is considered. In particular, a group of mobile robots that are using their sensory perception, while interacting with their neighbours, to navigate towards multiple target points or signals. These mobile robots are necessary for environments that are too dangerous, or unreachable for humans. The reason for implementing several robots instead of one is that they are able to complete a task much faster than one single mobile robot, by actuating in several places in parallel [1].

A problematic dilemma in SR is choosing the right algorithm for the right task. No algorithm is created equal- and they all have their own specific use case. One can think of exploration in unexplored domains like the moon, searching underground targets like gas pockets, an archaeologist looking for buried artefacts, the detection of mines [2, 3], signal source localisation [4–7], finding caves on mars [8], and more. Various multi-task (robotic) swarming algorithms have been proposed in the literature. Most of these originate from Particle Swarm Optimization (PSO), as it is an effective tool for solving these kinds of problems [7]. Many other variations exist, like Ant Colony Optimization (ACO) [9–11], Fruit Fly Optimization Algorithm (FOA) [12, 13], Glowworm Swarm Optimization (GSO) [14, 15], Bee Swarm Optimization (BSO) [16, 17], Firework Algorithm (FWA), [18], Bat Algorithm (BA) [19–21], Group Explosion Strategy (GES) [18], and more.

Because swarming algorithms are also used for non-robotic optimization applications. It is often deemed sufficient to simulate and benchmark these algorithms with standard mathematical functions, to figure out if they are able to find the global optimum, or get stuck in local optima [22]. Robots are also often modelled as point particles with no mass. A good swarming algorithm, therefore, does not necessarily mean it is a good *robotic* swarming algorithm. This review tries to bridge the gap between these algorithms and the robot, by also doing research on a few additional subjects. First, global robot architectures are researched. This defines the framework of how robotic inputs are connected to outputs and how this enables a single robot to plan actions or react. Secondly, swarming architectures are researched. In SR, some kind of collective emergence must emerge from individual robot behaviour. The introduced architecture ensures this. Thirdly, the communication techniques between robots (and/or base) are researched. This is important because SR in unknown environments suggests one can not assume that robots have the ability to communicate everything at all times to each other. A feasible communication method is therefore elementary and therefore considered. Next, obstacle and collision avoidance techniques are reviewed. Lastly, simulation methods are researched. This is also important because testing a robotic swarming algorithm can be a large investment, and it is thus smart to be able to simulate this beforehand.

This literature study examines differences between various swarming algorithms, robotic architectures, swarming architectures, communication architectures, obstacle and collision avoidance techniques and simulation methods in order to identify potential avenues for developing a scalable robotic swarming algorithm capable of conducting multi-target search in unknown environments. It provides a strong foundation for the development of such an algorithm by considering the various approaches and techniques that have been proposed in the literature.

Contents

List of Abbreviations

ACO	Ant Colony Optimization
AMEB	Architecture Multi-robot systems heterogeneous robots with Emergent Behavior
BA	Bat Algorithm
BSO	Bee Swarm Optimization
DDMR	Differential Drive Mobile Robot
DE	Differential evolution
ER	Evolutionary Robotics
FOA	Fruit Fly Optimization Algorithm
FSM	Finite State Machine
FWA	Firework Algorithm
GES	Group Explosion Strategy
GNSS	Global Navigation Satellite System
GSO	Glowworm Swarm Optimization
ISO	International organization of Standardization
LR-WPAN	Low Rate Wireless Personal Area Network
MANET	Mobile Ad-hoc Network
MRS	Multi-Robot Systems
OSI	Open System Interconnection
PFSM	Probabilistic Finite State Machine
PSO	Particle Swarm Optimization
RL	Reinforcement Learning
SPA	Sense Plan Act
SR	Swarm Robotics

1

Introduction

A group of archaeologists arrive at the 21 ha site. The site is known for its harsh weather. It is -10° Celsius during the day and even colder during the night. Normally, they start searching the area for clues to find points of interest by finding artefacts. The best time for field walking is considered to be in the winter months after the land has settled from ploughing, and rain softened the ground and washed the artefacts [23]. The terrain contains cracks and crevices that are hard to access, sometimes hazardous and completely out of reach for them. This would normally take them 25 days to prospect the area. They thought of using a drone with lidar, which would be able to fly for a few minutes and cover a very basic view of the area. However smaller artefacts like coins and pottery shards would be much harder to be spotted. Luckily, the team deployed their relatively cheap and agile scouting swarm robots a week earlier. These robots cooperatively and fully autonomously have searched for potential areas of interest. The only thing that needed to be done beforehand was to set up a few small generator-sized base stations that function as charging points, at easily reachable locations and the robots take it from there. Some robots have gotten stuck in the process, but this is no problem for the functionality of the swarm. The swarm returns a heat map of potential locations of interest, based on patterns on the ground, magnetic flux, found artefacts, or any other sensor that has been slapped on the robots. Archaeologists can now focus their attention on these high-potential areas using their more expensive tools. The vast, hard-to-reach area is now split up into manageable pieces, by using an inexpensive, fully autonomous robotic mobile swarm.

This vignette is a fictional scenario that could one day be real. Within the last decade, autonomous mobile robots have become one of the most promising industries of the near future. The decreasing cost of computation, the accuracy of sensors and the efficiency of communication protocols allow robots to complete tasks that recently were out of the scope of robotics. In particular in Swarm Robotics (SR). SR is not a new field, however, its definition of it is still quite vague. Terms like collective robotics, distributed robotics, and robot colonies often overlap [24]. As a starting point, swarming is defined as: ”The study of how large numbers of relatively simple physically embodied agents can be designed such that a desired collective behaviour emerges from the local interactions among agents and between the agents and the environment [24]”. The main characteristics of swarm robotics are defined as [25]:

1. robots are autonomous;
2. robots are situated in the environment and can act to modify it;
3. robots sensing and communication capabilities are local;
4. robots do not have access to centralized control and/or to global knowledge;
5. robots cooperate to tackle a given task.

Similar frameworks are defined by others, adding that the swarm must be homogeneous or heterogeneous, that is, composed of a number of homogeneous groups [24]. A good overview of the differences between SR and other similar systems is given in table 1.1.

	Swarm robotics	Multi-robot system	Sensor network	Multi-agent system
Population size	Variation in great range	Small	Fixed	In a small range
Control	Decentralized and autonomous	Centralized or remote	Centralized or remote	Centralized or hierarchical or network
Homogeneity	Homogeneous	Usually heterogeneous	Homogeneous	Homogeneous or heterogeneous
Flexibility	High	Low	Low	Medium
Scalability	High	Low	Medium	Medium
Environment	Unknown	Known or unknown	Known	Known
Motion	Yes	Yes	No	Rare
Typical Applications	Post-disaster relief Military application Dangerous application	Transportation Sensing Robot football	Surveillance Medical Care Environmental protection	Net resources management Distributed control

Table 1.1: Comparison of SR, MRS, Sensor networks and Multi-agent systems by [26]

SR is characterized by being: robust, flexible and scalable [27]. Robustness can be defined as the functionality of the system in the presence of failure of a part of the system, or other unexpected conditions. Flexibility can be defined as adaptability to changing requirements of the environment. Lastly, scalability can be defined as the ability to change the number of individuals in the system without drastically changing performance [27]. Swarming design typically is split up into non-adaptive/behaviour-based methods and automatic/learning methods. Only non-adaptive/behaviour-based methods are considered in this review, these are also the most used ones. In the global research phase of this study, only two studies are found that use automatic/learning methods [28, 29]. These methods are not given any further attention. To give the reader an overview of the field of SR, some taxonomies are added to the appendix in fig. A.1, fig. A.2 and fig. A.3 [25, 27].

1.1. Problem Statement

This survey focuses on bridging the gap between swarming algorithms and robotic swarming algorithms. The aim of this survey is therefore to provide a robust basis to enable one to develop a multi-target searching ground mobile *robotic* swarming algorithm that can be used in unknown environments, with limited information sharing and connectivity. The review should provide the reader with different options to choose from, depending on the kind of search problem. To solve the main objective, the following sub-objectives are formulated.

- Describe current swarming algorithms, what algorithm is best fit for what kind of target search?
- Describe and compare current obstacle and collision avoidance methods.
- What robot architectures are promising for robotic swarming?
- What swarming architectures are promising for swarm robots?
- What communication architectures are promising for swarm robots?
- What simulation methods could simulate a swarm of robots searching for multiple targets?

1.2. Outline of the survey

First, this review focuses on different swarming algorithms, and their use in SR (sections 2.1 to 2.8). The algorithms are found using a keyword search (chapter 2). Finally, in (section 2.9), through reasoning, and considering actual robotic use cases, potential best-use cases are designed for these different algorithms. To transition from algorithm to robot, a good robotic foundation must be decided upon. This has been realised by presenting a robotic architecture (section 3.1), robotic control architecture (section 3.2), swarming architecture(section 3.3) and communication architecture (section 3.4), in that order. Additionally, obstacle and collision avoidance methods are reviewed (section 3.5). Then, simulation methods are reviewed (section 3.6). Lastly, a problem definition and research plan are formulated (chapter 4).

2

Swarming Algorithms

Swarming papers that were taken into account, are assumed to comply with the statements and framework set in chapter 1. The keywords that are used to search the literature can be found in table 2.1. The table can be read from left to right, an example of a possible search query can be: ”swarming robotics multiple target localisation unknown environments” or ” swarm robots multi-source foraging unknown environment”. In addition to these search terms, other, more specific material is also reviewed. The following search engines were used: IEEE/IEE Electronic Library via IEEE Xplore, Sciencedirect by Elsevier, Scopus and Springerlink. In addition to these search engines, Google scholar was also used.

		OR		OR		
Swarm * AND Robot*	AND	mult* AND target	AND	explor*	AND	unknown AND environment
		mult* AND signal		local*		
		mult* AND source		search		
		mult* AND plume		foraging		

Table 2.1: Advanced keyword search used

The following chapters elaborate on the discovered algorithms, on a basic level- and a more advanced level. The algorithms are then summarized and put in place in a situation where they assume to belong. The best qualities and deficiencies of every one of them are also noted in this general summary in section 2.9.

2.1. Exploration Algorithms

Most algorithms need some method of random walking to explore. Therefore, this is the first thing that has been researched. A key task of autonomous mobile robots is to explore unknown environments with limited energy capacity, and other constricting conditions. The exploration method must therefore be efficient, and ensure the maximum area is covered without too much overlapping. In an information-lacking environment, random walking is the most flexible strategy. The following two methods are commonly used in robotic exploration, but there are many more options.

2.1.1. Levy Walk

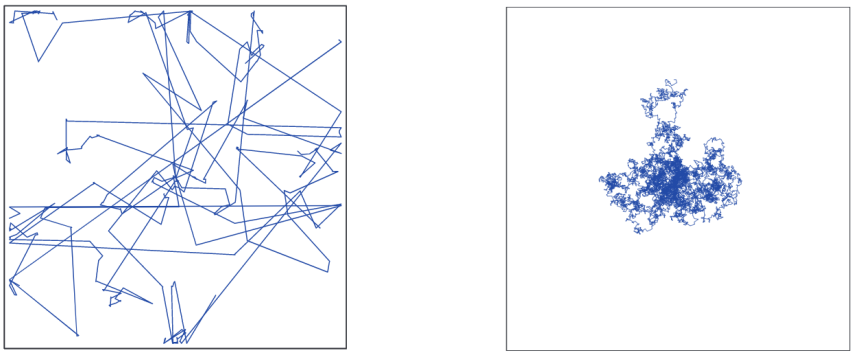
Levi walk stems from chaos theory and is useful for stochastic simulations for random or pseudo-random natural phenomena. Levy walk (inspired by levy flight) is one of the most efficient random walk patterns, amongst natural collective systems [30]. Its probability distribution is heavy-tailed, therefore favouring global search more than local search. It is used when the distribution of the targets is sparse, and the searchable area is large. An example of a levy distribution for the step size of a robot is shown in 2.1. Where $a \sim N(0, \sigma^2)$ and $b \sim N(0, 1)$ are two independent random variables with a normal Gaussian distribution.

$$v = \frac{a}{b^{\frac{1}{\alpha}}} \tag{2.1}$$

Levy Walk can be used to determine the step size of the robot. It generates smaller step sizes in high frequency with an occasional larger step size. This behaviour is visualised in fig. 2.1a. An example of a multi-robot collective Levy Walk is given by [31]. Robots move with a fixed linear speed and sample the duration of their next step from a Levy distribution. After the linear step, a rotating state is initialised. The rotation is done over a constant angular velocity, sampled from a uniform distribution.

2.1.2. Brownian walk

Brownian motion comes from random particle motion in a fluid, resulting from particle interaction. This behaviour was first mathematically formulated by Albert Einstein [32]. The robot will move a given step size, produced by the Brownian motion, and then randomly turn. Brownian motion can be viewed as a continuous-time stochastic process that is described by the Wiener process [33]. Brownian walk is more fit for local search. However, robots are prone to repeatedly search the same area. This can be solved by enlarging the step size, so the robots are less likely to re-explore their own paths. Also, robots can accidentally repeat the travelled paths of others. This behaviour is visualised in fig. 2.1b. [33] proposed a method where the step-size sizes scale accordingly to the robot density. This solves the problem in the multi-robot environment.



(a) Levy walk with 10 robots **(b)** Brownian walk with 10 robots

Figure 2.1: Two popular random walk methods in Swarm Robotics (SR) [33]

2.2. Particle Swarm Optimization (PSO)

One of the most suitable algorithms for guiding multiple robots is PSO. Here, robots are represented as particles that are searching for solutions in a virtual place. It has roots in two main methodologies. First in artificial life, like birds flocking, fish schooling and other swarming theory. It also has ties to evolutionary programming and genetic algorithms. The algorithm can be implemented using a few lines of code and primitive mathematical operators, making it computationally inexpensive.

2.2.1. Biological foundation

Different scientists have tried to create computer simulations of the movement of organisms in a bird flock or fish school. The swarming behaviour of a fish school can be seen in fig. 2.2. Reynolds, being of the more notable, was intrigued by the choreography and aesthetics of flocking birds [34]. Birds and fish adjust their physical group movement to avoid predators, seek food and mates, and manage temperature and other environmental parameters. Fundamentally, sharing social information in a group offers an evolutionary advantage [35]. PSO is originally inspired by the movement of flocking birds [35].

Figure 2.2: Swarming behavior in a school of fish [36]

2.2.2. Algorithm

A representation of a single particle in PSO is given in fig. 2.3. Let us denote x_{ij} as the position of particle i in the j-dimensional search space at time t. The position of one particle is changed by adding a velocity vector $v_{ij}(t+1)$ to the previous position, i.e.:

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \tag{2.2}$$

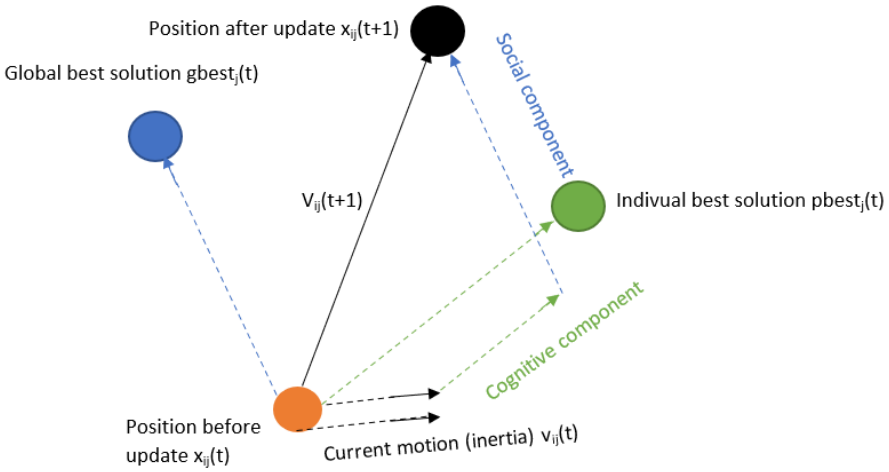


Figure 2.3: Particle representation of PSO

$v_{ij}(t+1)$ drives the particles, to a better position. This vector is influenced by its personal best experience $pbest$ also called the cognitive component. It is also influenced by the best position of some particle in the swarm $gbest$, also known as the social component. When $pbest$ is the best overall solution between all particles, it becomes $gbest$ for all particles in that set. The velocity of particle i, in dimension $j = 1, ..., n_x$, can then be calculated by 2.3.

$$v_{ij}(t+1) = v_{ij}(t) + c_1 \cdot r_{1j}(t) \cdot (pbest_{ij}(t) - x_{ij}(t)) + c_2 \cdot r_{2j} \cdot (gbest_j(t) - x_{ij}(t)) \tag{2.3}$$

where; $v_{ij}(t) =$ current particle velocity $r_{1j}, r_{2j} =$ uniformly distributed random number [0:1], $c_1, c_2 =$ acceleration constants, $pbest_{ij} =$ best personal position of particle i. $gbest_j =$ best position between all particles.

Inertia component

To balance global and local search in PSO, often a term called the inertia component w_i is introduced to the velocity update equation. A large inertia component favours global search, whereas a small value favours local search. This value is often based on the amount of iteration of the program, and the size of the already explored region. The velocity update equation changes to eq. (2.4).

$$v_{ij}(t+1) = w_i v_{ij}(t) + c_1 \cdot r_{1j}(t) \cdot (pbest_{ij}(t) - x_{ij}(t)) + c_2 \cdot r_{2j} \cdot (gbest_j(t) - x_{ij}(t)) \tag{2.4}$$

[37] researched PSO-based multiple-target search in unknown environments, by using Unmanned Areal Vehicles (UAVs). In this paper, different inertial function weights were evaluated. The influence of adding an inertial value is clearly indicated in table 2.2. w_{max} and w_{min} are the upper and lower bound of the explored space w, $iteration_{max}$ is the maximal amount of iterations and $Rand()$ is a random number between 0 and 1. It shows how this variable can be optimized for different kinds of tasks.

Function	Mathematical Form	Advantage
Constant function	$\omega = 0.7$	1. Facilitates Global search and Manual control over the exploration-exploitation criteria
Random Inertial Weight	$\omega = 0.5 + \frac{Rand()}{2}$	1. Continuously exploits the data, ideal for situations with large numbers of targets.
Squared Decreasing	$\omega = w_{max} - (w_{max} - w_{min}) * (\frac{iteration}{iteration_{max}})^2$	1. High exploration at the beginning which gradually decreases over time. 2. Ideal for a moderate number of targets
Sigmoid Increasing	$\omega = \frac{(w_{start} - w_{end})}{(1+e^{10^{log(gen)-2)} * (k - n * gen)})} + w_{end}$	1. High exploration till no targets are found, exploration decreases when targets appear. 2. Ideal for a small number of targets.
Chaotic Inertia weight	$z = 4 * z(1-z)$ $\omega = (w_{start} - w_{end}) * \frac{iteration_{max} - iteration}{iteration_{max}} + w_{end} * z$	1. High exploitation when targets are found. Results in faster convergence to targets. 2. Ideal in cases where immediate response is required.

Table 2.2: Influence of different inertial weights to the velocity update equation in 2.4 [37].

Fitness

A fitness function steers the particle in the right direction. It quantifies performance or quality of the particle or solution. This is often a simple function like the distance between the target t_x, t_y and the particle x_x, x_y, shown in eq. (2.5). $lbest$ and $gbest$ are influenced by this fitness function.

$$\text{fitness} = \sqrt{(t_x - x_x)^2 + (t_y - x_y)^2} \tag{2.5}$$

Balancing parameters w, c_1, r_{1j}, c_2 and r_{2j}

In section 2.2.2, the influence of the inertial component w has been explained. However the acceleration components c_1 (nostalgia), c_2 (envy), and random vectors r_{1j}, r_{2j} control the stochastic influence and balance of the cognitive and social components. Generally, a good balance is to have $c_1 \approx c_2$. Often, $c_1 = c_2$ is used. [38] showed that in a robotic search, it is better to take $c_2 > c_1$, such that the social component plays the major role in PSO for robotic target searching tasks. Furthermore, if $c_1 \gg c_2$, particles will wander excessively to their own personal best location. However when $c_1 \ll c_2$, particles will all be attracted to the global optimum and rush into optima. c_1, c_2 also have an influence on trajectory smoothness, where low values will result in a smooth trajectory, allowing slow exploration before being pulled to good regions. High values will result in abrupt movements. [38] opted that parameters w, c_1 and c_2 should be tested in an offline simulated environment, before using them in the field. r_1, r_2 are introduced to add some stochastic influence, balancing the social and cognitive components.

Advanced

[6] used sub-swarms to search for multiple cell phones signals. Robots that search for the same target form a mini swarm. These only communicate their $pbest$ value with other members of the mini swarm. The best $pbest$ value of the mini swarm becomes the local best value $lbest$. Each particle will stay in this local best neighbourhood in all iterations of the program. This enables multi-target search with PSO. Another problem that they tackled was robots overshooting their targets. This was solved by introducing a dynamically weighted value W_{RSS}. This value dynamically slows the robot down, when approaching the target, as seen in 2.6.

$$v_{ij}(t+1) = (w_i v_{ij}(t) + c_1 \cdot r_{1j}(t) \cdot (pbest_{ij}(t) - x_{ij}(t)) + c_2 \cdot r_{2j} \cdot (lbest_j(t) - x_{ij}(t))) \cdot W_{RSS} \tag{2.6}$$

[39, 40] used a constriction factor K. This was originally introduced by [41]. The main idea is that this constriction of PSO is taking advantage of the mathematical nature of K, which guarantees the convergence of the algorithm. Effectively, it is equivalent to the inertia-weighted method. The difference is that; no velocity clamping (prevents particles from exploding out of the search space) is needed, and it guarantees convergence. However, if velocity clamping and constriction are used together, it leads to faster convergence rates [42]. Using constriction changes the velocity update equation into:

$$v_{ij}(t+1) = K \cdot (v_{ij}(t) + c_1 \cdot r_{1j}(t) \cdot (pbest_{ij}(t) - x_{ij}(t)) + c_2 \cdot r_{2j} \cdot (lbest_j(t) - x_{ij}(t))) \tag{2.7}$$

with $K = \frac{2}{|2 - \phi - \sqrt{\phi^2 - 4\phi}|}$, where $\sigma = c_1 + c_2$, $\sigma > 4$. However, the random vectors r_1, r_2 are often also added to the equation such that $\sigma = c_1 \cdot r_1 + c_2 \cdot r_2$.

2.2.3. Summary

PSO is a widely used, simple and inexpensive algorithm for multi-target robotic swarming [6, 7, 37, 39, 40, 43–47]. To find multiple targets, the swarm must be divided into sub-swarms. This also helps with not getting stuck in local optima. PSO guides all robots in the sub-swarm at the same time towards targets by having a social component. This requires these robots to communicate with each other. The information they share is their personal best location. This is compared within the subswarm and used to determine the best next position. PSO in swarm robotics should be used if robots are expected to move like a flock of birds, relatively close to each other, stochastically moving away from each other to then join each other again.

2.3. Ant Colony Optimization (ACO)

ACO is often used to solve stochastic combinatorial optimization problems. It is good at best-path exploitation, and also has a sensible exploration mechanism in the form of probabilistic path selection. ACO is characterized by using positive feedback, distributed computation and constructive greedy heuristic [48]. Traditional ACO enforces the search space to be divided into discretized grids. ACO is also not guaranteed to be optimal.

2.3.1. Biological foundation

The first ACO algorithm is inspired by the foraging behaviour of ants and is developed by [49]. It started with the binary bridge experiment as seen in fig. 2.4. Foraging behaviour in Argentian ants was studied. Foraging means searching and returning food to the nest. A double bridge, with the same start- and end-point was constructed. It was assumed that ants deposit the same amount of pheromone, and no evaporation occurs. Eventually, the shorter path would have more pheromones and therefore be chosen with a higher probability. This concept was eventually realised into an ant optimization algorithm (SACO or Simple Ant Colony Optimization) for shortest path problems. This was iterated by introducing heuristic information for determining the probability of selecting a link, memory in the form of a tabu list, and different pheromone update rules using local and/or global information about the environment.

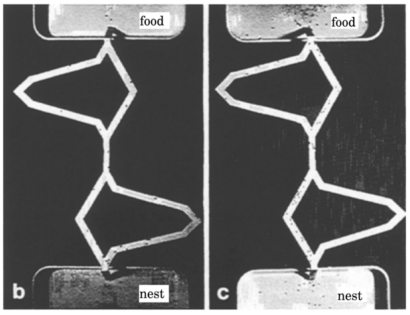


Figure 2.4: Studying foraging behavior of a colony of Argentian ants *Iridomyrex humilis* [50]

2.3.2. Algorithm

The probability of ant k moving from node i to j in time t is the following:

$$p_{ij}^k(t) = \begin{cases} \frac{\tau_{ij}^\alpha(t)\cdot\eta_{ij}(t)^\beta}{\sum_{u\in N_i^k(t)}\tau_{iu}^\alpha(t)\cdot\eta_{iu}^\beta(t)}, & j \in N_i^k(t) \\ 0, & j \notin N_i^k(t) \end{cases} \tag{2.8}$$

where τ_{ij}^α is the pheromone concentration of link (i, j), and represents how useful it has been in the past, serving as a memory. η_{ij}^β represents the pheromone attractiveness of the move, computed using some heuristics. For example, a distance measure can be introduced here to bias the search towards closer objectives. α and β further help to balance individual preference against global preference. N_i^k are the set of feasible nodes for ant k, from node i, in time t. If a node is visited, it is added to the ant's tabu list, and therefore removed from the feasible set N_i^k.

Pheromone Deposition and Evaporation
Pheromones are deposited and evaporate in time. Evaporation can facilitate better exploration and convergent behaviour. Pheromones are updated and evaporated using 2.9.

$$\tau_{ij}(t+1) = \rho\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}(t) \tag{2.9}$$

With $(1-\rho)$ as the evaporation, and $\Delta\tau_{ij}(t)$ as the amount of deposited pheromones. Four deposition methods introduced by [48] are shown in 2.10. The equation in the high-left corner is called Ant-cycle. Here, the pheromone deposits are inversely proportional to the quality of the link (global information), Q is a positive constant. The Lower-left equation is used when it is a maximisation task, for example maximising a certain signal value. The upper right formula is called Ant-density, essentially each ant deposits the same amount. The number of ants that follow a link will improve the desirability of that link. Lower-right is called Ant-quantity, the distance between two links (local information) is used to determine pheromone deposition. The shortest links are preferred here.

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} \frac{Q}{f(x^k(t))}, & \text{if link } (i,j) \text{ occurs in path } x^k(t) \\ 0, & \text{otherwise} \end{cases} \quad = \begin{cases} Q, & \text{if link } (i,j) \text{ occurs in path } x^k(t) \\ 0, & \text{otherwise} \end{cases}$$
$$= \begin{cases} Qf(x^k(t)), & \text{if link } (i,j) \text{ occurs in path } x^k(t) \\ 0, & \text{otherwise} \end{cases} \quad = \begin{cases} \frac{Q}{d_{ij}}, & \text{if link } (i,j) \text{ occurs in path } x^k(t) \\ 0, & \text{otherwise} \end{cases} \tag{2.10}$$

Balancing $\tau_{ij}(t)$ and $\eta_{ij}(t)$
Balancing between the ant's desire to choose existing pheromone levels on a link, and heuristic information is a task-specific variability in ant algorithms. In [9], an ant colony metaphor for multi-robot chemical plume tracing was constructed. Here, they chose two exponential functions (2.11, 2.12) to characterise their system. C, D are constants, $\rho_j(t)$ is the pheromone level of the j-th node at time t and $d_{ij}(t)$ is the distance euclidean distance between the ith and jth grid.

$$\tau_{ij}(t) = e^{\frac{\rho_j(t)}{C}} \tag{2.11}$$

$$\eta_{ij}(t) = e^{\frac{-d_{ij}(t)}{D}} \tag{2.12}$$

[10] used a greedy selection technique in a modified version of ACO, called Anti-Pheromone. The only difference is that ants now prefer lower pheromone levels, this changes $\tau_{ij}(t)$ into $\tau_{max} - \tau_{ij}(t)$. A multi-agent swarm was tasked to explore and search targets in an unknown environment. Here, with some probability, the links with the best combination of short distance and small pheromone levels are favoured. Ant k, located at node i, will choose node j in time t, according to 2.13. q is a random number, and $q0$ controls how often this greedy selection is used.

$$j = \begin{cases} argmax_{u \in N_k(i)}[\tau_{max} - \tau_{ij}(t)]^{\alpha} \cdot \eta_{ij}(t)^{\beta}, & q \leq q0 \\ p_{ij}^{k}(t), & q > q0 \end{cases} \tag{2.13}$$

Advanced
There are many ways to improve on the standard ACO algorithm. For swarm search, the anti-pheromone algorithm with greedy selection also mentioned in section 2.3.2 seems the most interesting. It drives robots away from visited 'places'. This requires them to partition the grid into nodes. The probability of ant k, moving from i to j in time t then changes into:

$$\mathsf{p}_{ij}^{k}(t) = \begin{cases} \frac{[\tau_{max} - \tau_{ij}(t)]^{\alpha} \cdot \eta_{ij}(t)^{\beta}}{\sum_{u \in N_i^k(t)} [\tau_{max} - \tau_{ij}(t)]^{\alpha} \cdot \eta_{iu}^{\beta}(t)}, & j \in N_i^k(t) \\ 0, & j \notin N_i^k(t) \end{cases} \tag{2.14}$$

Also, in the original ACO, it takes a long time for ants to reinforce paths. This can be solved by reinforcing frequently occurring solutions more than others [51]. Another method to increase efficiency is to rank ants based on their contribution [52].

2.3.3. Summary

Traditional ACO is perfectly adequate to solve stochastic, discrete combinatorial optimization problems [49] like the TSP problem. ACO in robotics is often applied in combination with other algorithms [9–11]. ACO its strength unfortunately lies in best-path exploitation, it could therefore find a better place in path optimization. ACO also requires the pheromone concentrations of discretized nodes or 'locations', to be saved and iterated upon. This calls for some centralized hierarchy or at least full-swarm global updates of these pheromone concentrations. Anti-Pheromone ACO [10] however does have some promising for swarm robotic, ants in this system are attracted to less visited notes, by inverting the consequence of pheromones, which in its turn improves exploration. Robots could use this Anti-Pheramone method to make sure they don't visit the same place twice or don't visit an already visited place by another robot. Using ACO can also be beneficial for path optimization (for example to truncate data throughout the swarm).

2.4. Bee Swarm Optimization (BSO)

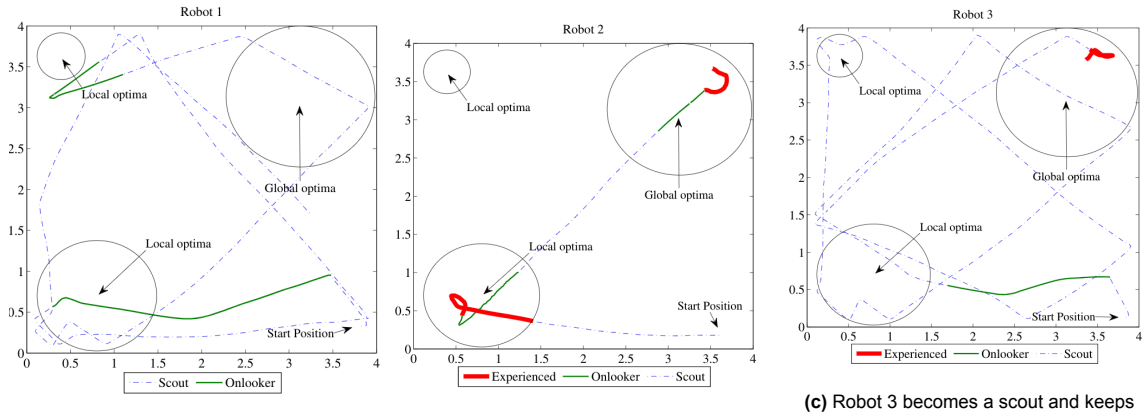
In BSO, bees get assigned different tasks. The search space is explored by a scout bee. On the other hand, patterns are formed by exploiting good areas by foragers and onlooker bees. This automatically introduces a good contrast between exploration and exploitation.

2.4.1. Biological foundation

Honey bees (*Apis mellifera*) are capable to perform complex tasks like collecting, processing nectar, advertising nectar, brood care, hive construction and foraging, only using relatively simple decentralized rules. In foraging, learning plays an important role. Honey bees need to be able to identify suitable resources, return them repeatedly, and communicate the location of the quality of the resources [53].

2.4.2. Algorithm

There are three kinds of bees in a colony: scouts, onlookers and foragers. These are divided into the following types: scout, onlooker, experienced forager and elite bee. Scouts are searching for new food sources, and the experienced forager and onlooker bees have the task of exploring the search space. Onlooker bees choose an experienced forager with high fitness, using a roulette wheel, receiving their position. This selected experienced forager now becomes an elite bee for this particular onlooker bee. In fig. 2.5, an example of how a bee algorithm performs is given. Here, bees change their type, based on their proximity to the target, and interactions with each other [17].



(a) Robot 1 stays onlooker after being a scout **(b)** Robot 2 becomes experienced forager **(c)** Robot 3 becomes a scout and keeps searching

Figure 2.5: Visualisation of the three tasks in BSO by [17]

Basic BSO uses three different patterns of searching, which can help balance exploitation and exploration. In the BSO algorithm, bees are first initialised. Some fitness function, like the euclidean distance between the bee and the target, or the intensity of the target signal, is calculated. A certain amount of bees with the poorest fitness values are selected as *scouts*. This selection makes sense, as these particular bees are not in the best place, and should keep exploring. The rest of the bees are halved,

and the best half become *experienced foragers*, while the other bees are selected as *onlooker bees*. The scout bees i random walks (RW) a region in dimension $j = 1..,n_x$ with radius τ, in time t, as seen in 2.15. Often, this radius decreases throughout iterations [16]. This helps with the exploitation of the search area.

$$x_{ij}(t+1) = x_{ij}(t) + RW(\tau, x_{ij}(t)) \tag{2.15}$$

The trajectory of the onlooker bee i, is calculated by 2.16. c_2 is an acceleration constant, and r_2 is a random variable in range [0,1], similar to PSO. $elite_j(t)$ is the position of the chosen experienced forager (elite bee), by onlooker bee i, using a roulette wheel.

$$x_{ij}(t+1) = x_{ij}(t) + c_2 * r_2 * (elite_j(t) - x_{ij}(t)) \tag{2.16}$$

Lastly, the experienced foragers update their position by using 2.17. c_1, c_2 are acceleration constants, and r_1, r_2 are random variables in the range [0,1], similar to PSO. $pbest_{ij}$ is the best position of experienced forager i in dimension $j = 1..,n_x$. $gbest_j$ is the position found between all experienced foragers. The relative importance and stochasticity of cognitive ($pbest_{ij}(t)$) and social ($gbest_j(t)$) components are determined by the acceleration constants and random variables defined before.

$$x_{ij}(t+1) = x_{ij}(t) + c_1 * r_1 * (pbest_{ij}(t) - x_{ij}(t)) + c_2 * r_2 * (gbest_j(t) - x_{ij}(t)) \tag{2.17}$$

Advanced
A problem in BSO, is that onlooker bees only move based on past experience of the forager bees. Because of this, the population could quickly ignore a part of the search space. [16] solves this by introducing a repulsion factor. This encourages the bees to randomly ignore the direction of the rest of the swarm. The onlookers and experienced foragers change their position update formula respectively into 2.18 and 2.19.

$$x_{ij}(t+1) = x_{ij}(t) + sign(r) * c_2 * r_2 * (elite_j(t) - x_{ij}(t)) \tag{2.18}$$

$$x_{ij}(t+1) = x_{ij}(t) + sign(r) * [c_1 * r_1 * (pbest_{ij}(t) - x_{ij}(t)) + c_2 * r_2 * (gbest_j(t) - x_{ij}(t))] \tag{2.19}$$

with $sign(r)$ equal to 2.20, P_{rf} equal to some predefined probability, and r some random number. This helps increase the diversity of the populations and introduces some additional stochastic influence.

$$sign(r) = \begin{cases} 1, & \text{if } (r \leq P_{rf}) \\ -1, & \text{if } (r > P_{rf}) \end{cases} \tag{2.20}$$

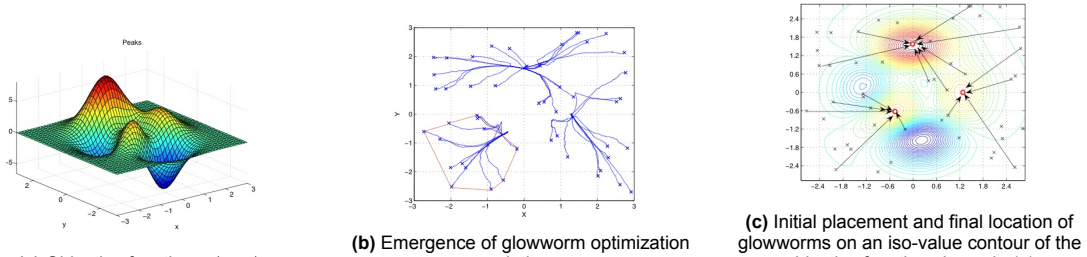
[16] gave the experienced foragers memory, enabling them to use historical information about previously visited high-fitness locations. This makes the algorithm stronger, and less prone to be stuck in local minima.

2.4.3. Summary

BSO is a 3-state algorithm that uses intensive communication to share points of interest with others. BSO is somewhat similar to PSO in terms of the movement of single agents but has some overhanging framework to determine social behaviour. BSO requires strict communication between experienced foragers and onlookers, and is, therefore, less fit for tasks where good positioning and communication (range) are limited. The algorithm is mostly fit for use cases where targets are spread out. Robots (scouts) start by searching the entire space randomly, and the initial placement, therefore, does not influence the performance of the algorithm. It also allows a 'base'-station to be used. Here, onlooker robots can for example rest and charge till they get the information from the experienced foragers.

2.5. Glowworm Swarm Optimization (GSO)

GSO is a simple algorithm often used to find the optima of multi-modal functions. The algorithm uses an adaptive local decision domain. This means that worms can only sense and communicate with other worms within their sensor range. Glowworms in GSO are able to identify and localize their neighbours, select a leader among neighbours, update the associated fitness value, and make a step towards the preferred neighbour. An illustration of sub-swarms of glowworms iterating towards several targets is given in fig. 2.6.



(a) Objective function $J(x, y)$

(b) Emergence of glowworm optimization solution.

(c) Initial placement and final location of glowworms on an iso-value contour of the objective function shown in (a)

Figure 2.6: Visualisation of a multi-agent glowworm swarm iterating towards multiple solutions [14]

2.5.1. Biological foundation

Glowworms (*lampyris noctiluca*) are a typical species of the common glowworm. They use luminescence to attract mates. Glowworms will often glow for two hours, and then go back to their hiding place, or stop when they find a mate. A brighter light indicates a better fitness of the woman worm, producing the light. It indicates it is larger and can lay more eggs. The term glowworm is used for both adults and larvae of firefly species such as Lampyris noctiluca [54].

2.5.2. Algorithm

Agents in GSO carry a luminescence quantity called luciferin. Glowworms are thought to be agents that emit light, whose intensity is proportional to this luciferin value. The more luminous glow of other glowworms attracts neighbouring worms. GSO agents are thought to only have a short measurement field and are only able to make decisions based on this perceived range. The decision range of an agent is determined by the radial sensor range of the agent. This is visualised in fig. 2.7.

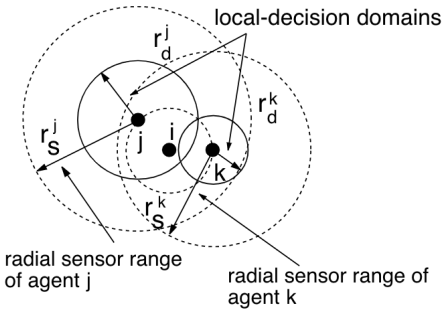


Figure 2.7: Local-decision domain $r_d^k < d(i,k) = d(i,j) < r_d^j < r_s^k < r_s^j$. Worm i is in the sensor range of both j and k. However, worm k is not in the decision-domain of worm j

First, glowworms are randomly placed in the environment, and all of the glowworms get the same luciferin value. In each iteration, luciferin values are updated based on the objective function, and then movement is decided based on a transition rule. This luciferin update rule is given by 2.21. Where ρ is the decay constant $(0 < \rho < 1)$ (simulating luciferin decay) and γ is a luciferin enhancement constant. $J_i(t)$ is the value of the objective function for worm i at time t.

$$l_i(t+1) = (1-\rho) * l_j(t) + \gamma J_i(t+1) \tag{2.21}$$

Next, the transition to the movement phase is done using a probability mechanism. Worms move towards other worms with a higher luciferin value, that is, if the glowworm is in the decision domain. The probability of glowworm i, moving towards worm j in the set of potential worms of $N_i(t)$, is given by 2.22.

$$p_j(t) = \frac{l_j(t) - l_i(t)}{\sum_{k \in N_i(t)} l_k(t) - l_i(t)} \tag{2.22}$$

where $j \in N_i(t)$, $N_i(t) = \{j : d_{i,j}(t) < r_d^i(t); l_i(t) < l_j(t)\}$. $d_{i,j}$ is the euclidean distance between the worms i and j at time t. $l_j(T)$ is the luciferin level of worm j at time t. $r_d^i(t)$ is the variable local-decision range of worm i and r_S is the radial range of the sensor. The next move of the glowworm can then be defined as seen in 2.23. Here, s is the step size of worm i.

$$x_i(t+1) = x_i(t) + s\frac{x_j(t) - x_i(t)}{||x_j(t) - x_i(t)||} \tag{2.23}$$

Local-decision update rule
The glowworms depend on only local information to decide their movements. It is also assumed that no *a priori* information about the objective function is available. In order for the glowworms to detect multiple peaks, the sensor range must be made variable. Each agent i is therefore equipped with a local-decision domain whose radial range $r_d^i(t)$ is dynamical in nature $0 < r_d^i \leq r_s^i$ [14]. A dynamic update rule has first been proposed [55], however due to oscillatory behaviour, enhanced [14]. The update rule limits the decision range based on the number of neighbours, dynamically, and has been shown to substantially enhance performance.

Advanced
[15] introduces a leapfrogging effect to the basic GSO. This gives rise to a local-search behaviour between two glowworm pairs along a single ascent direction. The way this algorithm works is when glowworm i approaches neighbour j, due to j having a higher luciferin value, and is actually closing onto j, closer than step size s. In the next iteration, i leapfrogs over the position of j and becomes a leader to j. Now i stays stationary and j overtakes the position of i. This solves the collision avoidance problem together with the algorithm being prone to deadlock, however, seems redundant and not efficient.

2.5.3. Summary
GSO is a truly simple algorithm basing movement solely on one factor; luciferin value or 'glow'. It can be used for multi-robot search tasks, localisation or rendezvousing, in environments where robots have a limited communication range [14, 15, 55, 56]. Unfortunately, GSO does not have good exploration properties, as agents do not explore at all. They can therefore easily get stuck in local optima. To use this algorithm for several targets, robots must be initially spread out well. GSO is computationally inexpensive. This algorithm would come to good use in environments where next to no communication is available, and robot can sense their neighbours. An example could be in caves, where robots use actual light to communicate their findings. If there are enough robots that are spread out well enough, the algorithm could suffice in finding all targets, and therefore be sufficient.

2.6. Fruit Fly Optimization Algorithm (FOA)
FOA is a simple but effective localisation algorithm, only requiring simple parameters, and is widely used in intelligent evolutionary algorithms [57, 58]. Compared to other optimization algorithms, FOA is relatively easy to understand and computationally inexpensive. This is especially true for solving multi-swarm problems [12, 13].

2.6.1. Biological foundation
The kind of fly is called the *Drosophila*, more commonly known as 'fruit fly'. The species is superior in sensing and perception, especially in osphresis and vision. The osphresis organs can smell all kinds of food, even to 40 km away. After smelling the food, it can use its sensitive vision to move towards the food.

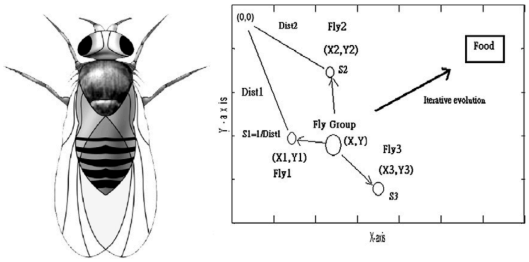


Figure 2.8: Illustration of FOA [57]

2.6.2. Algorithm

FOA is a method used for global optimization based on the food-finding behaviour of the fruit fly. The phases of this behaviour are shown next.

1. Random location of fruit fly swarm is initialized $(InitX\_axis; InitY\_Axis)$
2. Give the random direction and distance for the search of food using osphresis by an individual fruit fly: $X_i = X\_Axis + Randomvalue$, $Y_i = Y\_Axis + Randomvalue$
3. The food location is unknown, therefor this is estimated by $Dist = \sqrt{X_i^2 + Y_i^2}$. A judgement value (S) is calculated, which is reciprocal of the distance $S_i = 1/D = Dist_i$
4. The smell concentration judgement value is evaluated with a fitness function $Smell_i = Function(S_i)$
5. Find out fruit fly with best smell concentration among the whole swarm $[bestSmellbestIndex] = max(Smell)$
6. Keep the best smell concentration value and x, y coordinate, the fruit fly swarm will use vision to fly individually towards this location $Smellbest = bestSmell$, $X\_Axis = X(bestIndex)$, and $Y\_Axis = Y(bestIndex)$.
7. Iterative optimization, and repeat steps 2-5, then judge if the smell concentration is better than the previous iteration, if so, implement steps 6

Fruit fly optimization characterises itself in that all flies fly independently towards the best-found food location.

Advanced

[12] used a hybrid PSO and FOA algorithm. First, FOA was improved to work with multiple swarms. This solves the issue of being trapped in a local optimum and premature convergence [58]. The sub-swarms independently search the space, and one sub-swarm FOA particle is seen as a robot. The output of the sub-swarm FOA (x, y coordinate of best smell concentration) is assigned as the input value of the PSO algorithm ($gbest$). This way, FOA is used to provide a better next optimal sub-swarm position value for PSO. Additionally, using sub-swarms enhances the diversity and achieves effective exploration to avoid getting stuck in local optima and premature convergence. It is also necessary when searching for multiple targets. [13] also implemented a multi-swarm FOA approach, separating the fruit fly swarm into multiple swarms.

2.6.3. Summary

FOA is an all-or-nothing algorithm, directing all flies towards the best position at that moment. Multi-swarm robotic FOA exist [13, 58], however using it in a hybrid form can improve on the limitations of just using FOA [12]. Instead of PSO, where all particles are continuously influencing each other's motion by balancing their personal goal and global goal, here the one best global position is shared with all flies. They then individually fly towards it using sight. However, the limited search space and swarm diversity weaken the global search ability. Unfortunately, this does make it prone to getting stuck in local optima. The advantages are that it is an easy and computationally inexpensive algorithm. FOA could be great for quick exploitation, as all flies fly towards the best position. Say there are only two targets in the search space, two sub-swarms can be created, and both of them can be directed to find one target. All robots are directed to find this target and do not have any other goal, they navigate towards this goal individually, not requiring any further communication.

2.7. Bat Algorithm (BA)

To some degree, BA can be considered as a combination between standard PSO and intensive local search, which is controlled by loudness and pulse rate [19]. BA enables bats to change their measurement, based on the distance from the bat to their target. The bat algorithm is a novel approach to multi-robot target search and has many advantages, such as frequency tuning, fast solving speed, high precision, little adjustment parameters, and global convergence [20].

2.7.1. Biological foundation

Bats are the only mammals with wings. They have an advanced capability of echolocation. Their size range from the bumblebee bat (1.5 g to 2 g), to giant bats with a wingspan of 1 m and weight up to 1 kg. Bats use sonar for echolocation, detecting prey and avoiding obstacles. Most bats use short, frequency-modulated signals that sweep through an octave, while others use a constant frequency. Pulses often last about 8 ms to 10 ms. The frequency of typical bats is around 25 kHz to 100 kHz. These ultrasonic waves can be as loud as 110 dB. When hunting for prey, the pulse rate is sped up. A bat can process the receiving pulse with a speed of $300\,\mu$s till $400\,\mu$s. They use the time difference between their ears, and loudness variations of the echos, to build a three-dimensional surrounding. This Doppler effect process by discriminating between variations in wing flutter rates enables bats to know the difference between targets.

2.7.2. Algorithm

In the bat algorithm, several characteristics are idealised. First, all bats use echolocation to sense distance and know the difference between food/prey and obstacles/the environment. Secondly, bats initially fly randomly, searching for targets, with velocity v_{ij}, at position x_{ij}, with frequency f_{min}, varying wavelength λ, and loudness A_0. They can then change frequency or wavelength based on the distance to the target. It is assumed that the frequency range is linearly correlated with the wavelength, using $\lambda = \frac{v}{f}$. The frequency range, for example; [20 kHz, 500 kHz], corresponds to a wavelength from 0.7 mm to 17 mm with $v = 340$ m/s. Both frequency f_i or wavelength λ_i can therefore be changed, depending on the use case. The product $\lambda_i \cdot f_i$ is constant and is the velocity increment. Adjusting one of the values while keeping the other constant will adjust the velocity accordingly. Here, the frequency is used. The movement of bat i, in dimension $j = 1.., n_x$, and time t, is determined by equations 2.24,2.25 and 2.26.

$$f_i = f_{min} + (f_{max} - f_{min})\beta, \tag{2.24}$$

$$v_{ij}(t+1) = v_{ij}(t) + (x_{ij}(t+1) - x_{j*})f_i \tag{2.25}$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1), \tag{2.26}$$

f_i changes the velocity depending on the frequency that is used. Also, $\beta \in [0, 1]$ is a random factor from a uniform distribution. x_{j*} is the current global best location or solution. The next position of the bat $x_{ij}(t+1)$ is updated by adding a velocity $v_{ij}(t+1)$ to the previous position, like in Particle Swarm Optimization (PSO).

Once a solution is selected within the current best solutions, local search begins. A new local search solution for each bat in dimension $j = 1.., n_x$, is generated using a random walk method as seen in 2.27 [19, 21].

$$x_{new,j} = x_{old,j} + \epsilon A^t_{mean} \tag{2.27}$$

Where $\epsilon \in [-1, 1]$ is a random number, and $A_{mean}(t) =< A_i(t) >$ is the average loudness of all bats. A high average loudness will result in more exploration, which is resourceful at the beginning of the algorithm, it then slowly goes down in time. This can be seen in 2.29. Loudness and pulse emission are controlled via 2.28. Where α, γ are constants. The loudness and emission rates are only updated when new solutions are improved, meaning bats are moving towards the optimal solution. As a bat approaches a target (moves towards the optimal solution), loudness A_i decreases and the rate of pulse emissions $r_i \in [0, 1]$ increases to the initial set value. This makes sense as the bat is trying to iterate to a continuously closing-in target, and thus needs more information to track it precisely, but does not need full loudness, as the target is already close.

$$A_i(t+1) = \alpha A_i(t), \qquad r_i(t+1) = r_i(0)[1 - exp(-\gamma t)] \tag{2.28}$$

With $0 < \alpha < 1$ and $\gamma > 0$.

$$A_i(t) \to 0,\ r_i(t) \to r_i(0), \qquad \text{as } t \to \infty \tag{2.29}$$

Advanced
Standard BA has strong randomness in the frequency calculation. [21] has therefore added the Doppler effect to let the frequency be modulated by the proximity to a target. f_i hereby changes to 2.30

$$f_i = f_{min} + (f_{max} - f_{min}) * d_i \tag{2.30}$$

with d_i equal to 2.31.

$$d_i = (\frac{v \pm v_i^t}{v \mp v_s}) * d_o \tag{2.31}$$

Where d_0 is the original frequency of the target, v is the propagation speed of the wave, and v_i^t is the moving speed of the robot. This value is positive for approaching the emission source and negative if far away. v_s is the speed of the target if approaching the robot. This value is negative if the target is in front and positive if it is on the back of the robot. Concluding, the frequency d_i is increased if either robot or emission is moving towards each other. The speed of the robot is adjusted by adaptively adjusting the frequency to avoid premature convergence.

2.7.3. Summary
Bat algorithms already show potential for solving optimization problems [19, 20]. However, only one use case for multi-target search in unknown environments, using swarm robotics, has been found [21]. To find multiple targets, the swarm must be divided into sub-swarms. This also helps with not getting stuck in local optima. BA guides all robots in the sub-swarm at the same time towards targets by having a social component. This requires these robots to communicate with each other. The information they share is their personal best location. This is compared within the subswarm and used to determine the best next position. BA is very similar to Particle Swarm Optimization (PSO), the differences being that (i) measuring intensity or ”loudness” goes down in time, and the measuring rate goes up if a target is approached (ii) frequency of bats influence speed update equation (randomly or by Doppler effect). This requires more complex sensors and receivers and some extra synchronisation between bats. Using echolocation in complex environments nonetheless cause unexplored problems, as bats are expected to magically know the difference between food/prey and background barriers. The concept of dynamic measuring based on target distances could however be interesting for Swarm Robotics (SR), if used properly. It could then potentially increase energy efficiency.

2.8. Group Explosion Strategy (GES)
GES comes from a family inspired by explosion phenomena. An example of a similar kind of algorithm is the Firework Algorithm [59]. Explosions begin from a centre point, fragments scatter around this centre. This centre is now regarded as a robot, and in the same sense, neighbouring robots are the fragments of the explosion. This encourages concentrating on a point of interest, while still effectively exploring the search space around the point of interest. In each iteration, robots decide their movement, selecting new explosion centres again and again.

2.8.1. Algorithm
The goal of the GES algorithm is to get the centre of a group, centred around one robot, to the best position within the group, as seen in fig. 2.9. Robots around the centre robot move in a way, such that the group centre is the current position of the robot with the highest fitness value, among all robots within the group. The swarm has no leader, nor unique IDs, and shares no common coordinate system or global position system. Each robot senses the fitness of its position and has a limited sensing range to detect the relative position of its neighbours. All robots execute the same algorithm but act independently. Synchronization is not needed. Robots also have the ability to store 10 past states.
The group centre, seen by robot i can be calculated by 2.32. This is the centroid of all robots within the group.

$$C(i) = \frac{\sum_{j \in N(i)} P(j) + P(i)}{|N(i)| + 1} \tag{2.32}$$

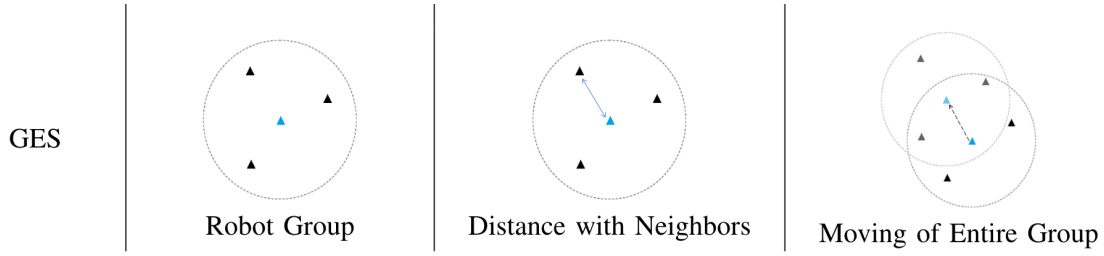


Figure 2.9: Group Explosion tactic [18]

where $P(i)$ is the current position of robot i, $N(i)$ is all the neighbouring robots around robot i. This centre can be calculated without sharing direct location information with each other. Given this group centre, the grouping component can then be calculated, as seen in 2.33.

$$G(i) = (P(b) - C(i)) * R_s \tag{2.33}$$

where robot b is the robot with the best fitness in the group. R_S is a scaling factor selected from $1-\beta_G$, 1 or $1+\beta_G$. β is a constant. This influences the shape of the group, controlling the distance of robots to the group centre. The proposed algorithm is visualised in fig. 2.10. The splitting of the group is shortly explained in the next chapter.

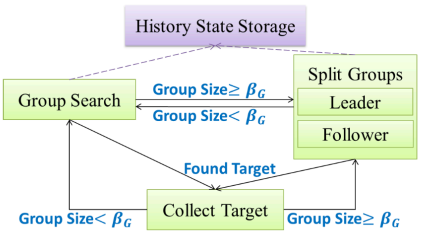


Figure 2.10: Group Explosion flow chart [18]

Group Splitting

If a group exceeds a certain threshold β_G, the swarm is split into two smaller ones. The two robots with the best fitness leader one $L1$ and leader two $L2$ are selected as the new leaders. These leaders repulse each other and move away as much as possible. This repulsive factor is calculated in 2.34. β_R is a predefined coefficient. A more elaborate explanation can be found in [18].

$$R(L1) = (P(L1) - P(L2)) * \beta_R \qquad R(L2) = (P(L2) - P(L1)) * \beta_R \tag{2.34}$$

Utilizing History States

When a robot searches in the wrong direction, meaning its observed fitness decreases, it can get back to making use of the history component. It then chooses the position in that list with the highest fitness.

2.8.2. Summary

GES can be interesting for robotic multi-target search [18]. The chaotic nature of the algorithm helps the initial exploration phases. Robots 'explode' in a ring formation around the robot with the highest fitness. With enough robots in the field, these explosions can continuously happen, increasing exploration by exploding, and exploitation around interesting areas. The fact that this algorithm doesn't need synchronization and is simple and easy to implement can be interesting for swarm robotics. The exploration properties however are chaotic, and hard to control. An example of where this algorithm would be the algorithm of choice is when multiple targets are scattered around each other. For example, finding parts of an exploded car. Robots will converge towards this target, and around it, searching for nearby targets in a circular formation, and continuously exploding around each other.

2.9. Summary of algorithms

Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), Bee Swarm Optimization (BSO), Glowworm Swarm Optimization (GSO), Fruit Fly Optimization Algorithm (FOA), Bat Algorithm (BA) and Group Explosion Strategy (GES) were reviewed in the previous chapters. In every chapter of each algorithm, the biological foundation and basic algorithm are first explained. Then a more advanced explanation is presented, adding examples and possible improvements to the algorithm. Lastly, a summary of the algorithm and the significance of using it in robotics is given. In fig. 2.17, a flowchart is provided, to exemplify and elaborate on which algorithm would prosper in what conditions. Algorithms are split up based on: initial robot placement, the ability of the swarm to find multiple targets without functionally splitting the swarm up (ease of scalability), and the need for synchronisation and/or communication between the swarm. All examples have a designated number from 1-9. Underlined algorithms (like BSO) can be used in both conditions.

To start with ACO, it is best used in places where best-path planning is critical to finding targets. This 'style' could be associated with the game minesweeper. Robots however start from a home or base position and try to figure out how to reach the targets. They divide the search space into a grid and create nodes. Robots can then move from node to node, releasing pheromones. The anti-pheromone ACO system could also be used to permanently direct robots away from for example lakes, gaps, or other potentially dangerous places.

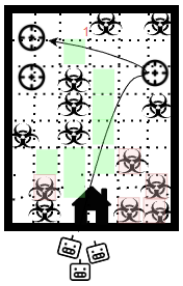
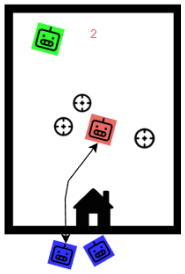
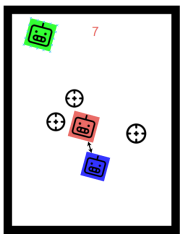


Figure 2.11: Best conditions for using ACO. The house represents a base station where robots are originally situated. They then move from this base station and explore the best path to targets while reinforcing paths

BSO robots can be used in both conditions: 'One starting point or hub' and 'spread out in the field'. Both cases are deemed feasible and are considered in examples 2 and 7 respectively. Green robots represent scout bees, blue robots are onlooker bees and red robots are experienced foragers. Scout bees search around for targets, and whenever they find a target they become experienced foragers. Experienced foragers need to be able to communicate with onlooker bees to make this algorithm work. The algorithm then tasks onlookers to help foragers in searching high-density areas by exploiting the exploration-exploitation balance. The difference between examples 2 and 7 is that in example 2, onlooker robots charge or rest at the base station until they are recruited to aid the experienced foragers in their mission. In 7, these robots are idle in the field.



(a) The house represents a base station where robots are initially situated. Green robots are scouts and search the area for targets. Red robots are experienced foragers that have found targets and recruit blue robots (onlookers) to help search the area



(b) The same situation as fig. 2.12a now exists, however, robots are spread out in the field at initialisation. Onlookers are redundant till they are called upon

Figure 2.12: Best conditions for using BSO

In example 3, the FOA swarm must be divided into several sub-swarms to be able to find multiple targets. This algorithm is optimal if there are only a few targets that need to be found quickly. This is because all members of the subswarm will iterate towards that one solution without further exploration. An example could be, looking for a few people in a burning house. Whenever a robot is out of range, it will not partition in the group searching activity of the swarm anymore and become somewhat redundant.

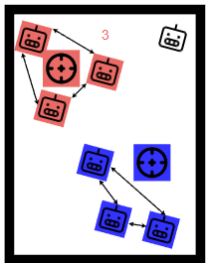
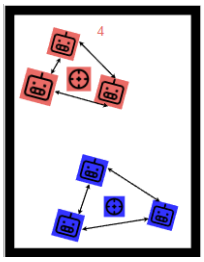
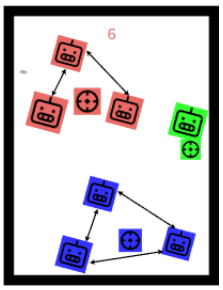


Figure 2.13: Best conditions for using FOA. Here, there are only two targets. Every sub-swarm searches one target and iterates towards it quickly, the different swarms are separated by colour. The white robot is out of the signal reach and is unable to participate in the group searching activity.

The BA in examples 4 and 6 must also be subdivided into multiple swarms to enable it to find multiple targets. BA is best used to find many targets, as exploitation and exploration are balanced well in comparison with FOA. The bat algorithm uses individual and collective preferences to figure out the next best location for that robot, similar to PSO. The robots should therefore be perfectly fine and still find targets if they fall outside of the communication range of their swarm. Nonetheless, the BA is divided into *'Need synchronisation and communication to work'*, and *'Doesn't need synchronisation and communication to work'*. This is because members of the sub-swarm in BA base their measurement loudness and thus exploration properties on the average loudness of all bats. A bat getting out of the communication range is not rudimentary (it can still find a target, see green robot) but does influence the execution of the overall algorithm and is therefore placed in both factions.



(a) All robots within their sub-swarm are in communication with each other and exchange their loudness levels.



(b) All robots within their sub-swarm are in communication with each other to exchange their loudness levels. When a robot is outside of a subswarm (green robot) it can go on and find targets alone, however, the exploration properties of the algorithm are not synchronised.

Figure 2.14: Best conditions for using BA

PSO in example 5 is similar to the BA in example 6. PSO would be good enough if the cost of measuring and deciding the fitness of the robot is low (using low-energy consuming light sensors). On the other hand, if measurements have a high energy cost, like using sonar to penetrate the ground, the BA could potentially bring the energy requirement down by increasing measurements only when fitness values increase or/and distance to target decreases. Another example to use BA above PSO could be if the target signal is dynamic. The BA would also be great in following a moving target.

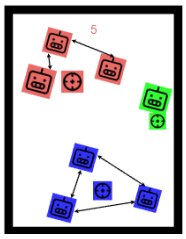
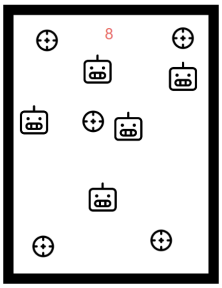
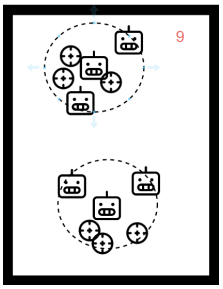


Figure 2.15: All robots within their sub-swarm are in communication with each other to exchange their loudness levels. When a robot is outside of a subswarm (green robot) it can go on and find targets alone

Examples 8 and 9 are somewhat similar to each other. Both algorithms can find multiple targets without being split up into distinct subswarms. They both also do not need any communication nor synchronisation to work. Both algorithms need to be spread out well initially to increase the chance of finding all targets. These are truly simple yet effective algorithms for multiple target searching. The difference is that GSO would be more effective if the targets are scattered and GES would be a better choice to find targets that are spread in clusters. Both algorithms require many robots to be in the field and work well using only local sensing.



(a) GSO is better used if targets are scattered far, and many local sensing robots can be used as swarms. These swarms do not need to be divided into sub-swarms for this algorithm to work.



(b) GES is better used if targets are scattered in clusters, and many local sensing robots can be used as swarms. These swarms do not need to be divided into sub-swarms for this algorithm to work.

Figure 2.16: GSO and GES and their use cases

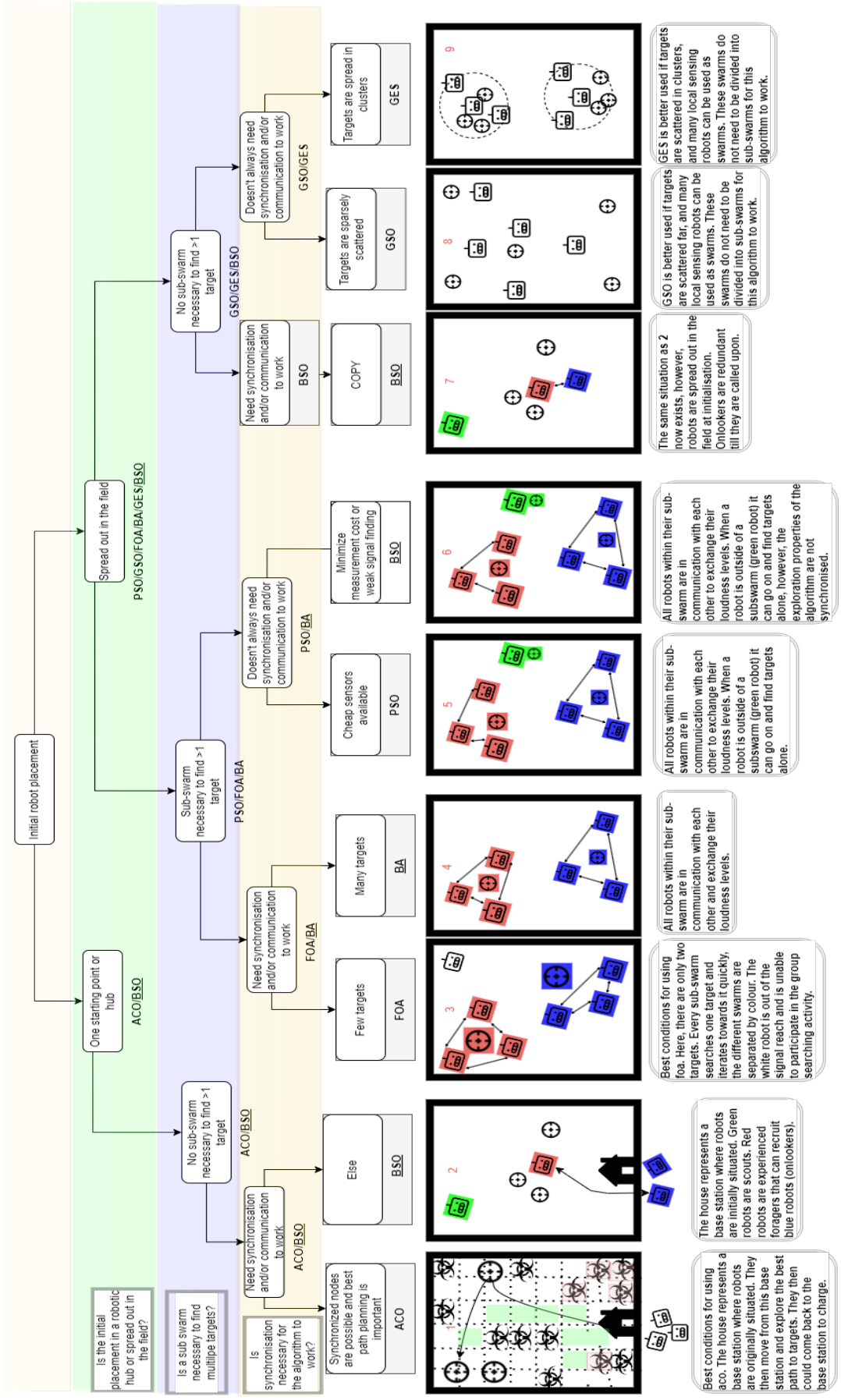


Figure 2.17: Flowchart of choosing the right algorithm for specific environments

3

From Algorithm to Robot

This chapter is dedicated to finding the best methods to prepare any multi-target searching swarming algorithm for being used on a robot. This is done by researching the following topics: global robotic architecture, swarming architecture, communication architecture and simulation methods. A global but strong theoretical framework to create a robotic swarming algorithm for multi-target search in unknown domains should be achievable after reading this chapter.

3.1. Global robot architecture

A strong robotic framework is critical to connecting the different inputs of a robot to desired outputs. Simply said, the global robot architecture defines the way different modules communicate with each other, and are organised, to eventually find a way to connect inputs to output commands and acquire the desired result. To avoid confusion, a robotic architecture is different from a control system. A control system is tasked specifically with a task, like following a path or maintaining a certain speed. Contrarily, the robot architecture can include this control system as a module within this global architecture.

3.1.1. Deliberative

The first robotic architecture was applied to artificial intelligence techniques, heavily relying on the presence of an internal model of the robot. This first architecture was named Sense Plan Act (SPA). It is a top-down method where high-level constraints are broken down into low-level commands. It is also known as the deliberative navigation architecture. SPA is composed of three subsystems, namely; sensing, planning and execution. This method was widely used for many years. In the early 1980s, researchers realised that SPA had many problems. The planning phase was very slow and unfeasible for most real-world problems. Also, acting based on only the internal model, without relying on sensory information was not efficient. Lastly, there exists no direct mapping between robot sensors and the internal world model due to noise in the sensors [60].

3.1.2. Reactive

In an attempt to solve the problems that arose from the deliberative architecture in section 3.1.1, the subsumption architecture was introduced. This is a bottom-up approach, where high-level constraints are not integrated into the action generation process [61]. The architecture can be analogised with 'reflexes'. This architecture is also known as subsumption architecture. It is heavily associated with behaviour-based systems. The structure is composed of asynchronous modules which communicate using low-demanding channels. Every channel, or module, is essentially a finite state machine which interconnects sensors to actuators directly. These modules are also known as behavioural layers. In figure fig. 3.1, the semantic work of Brooks (1986) [62] introduces this control architecture. This architecture is similarly defined in something called the motor schema architecture.
This subsumption architecture consists of competence levels. These levels are visualized in fig. 3.2. The higher the competence level, the more specific the behaviour is, i.e. the more constraints are put on the behaviour. A module in the subsumption architecture is built by blocks like the one seen in

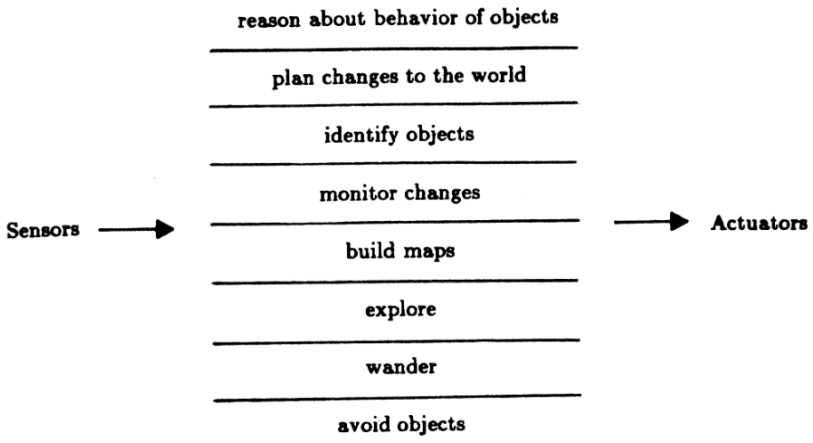


Figure 3.1: Mobile robot control architecture in the subsumption architecture [62]

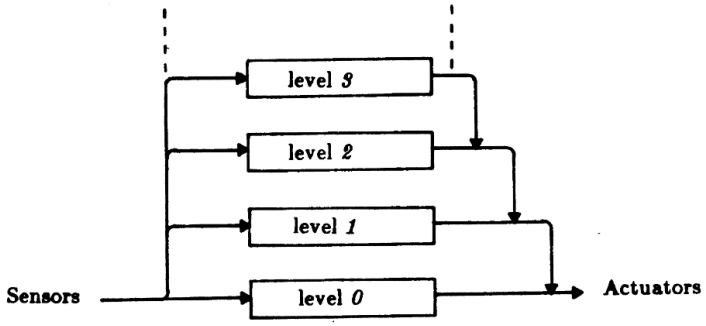


Figure 3.2: Subsumption Architecture competence layers [62]

fig. 3.3. An output line from one module can be connected to the input of another. Additionally, higher levels of competence can inhibit signals coming from lower levels. For example, obstacle avoidance can inhibit target search tasks. A more elaborate example of such a subsumption schematic is given in fig. A.4. Here, a level 2 subsumption architecture is displayed, characterised by the two inhibiting levels present.

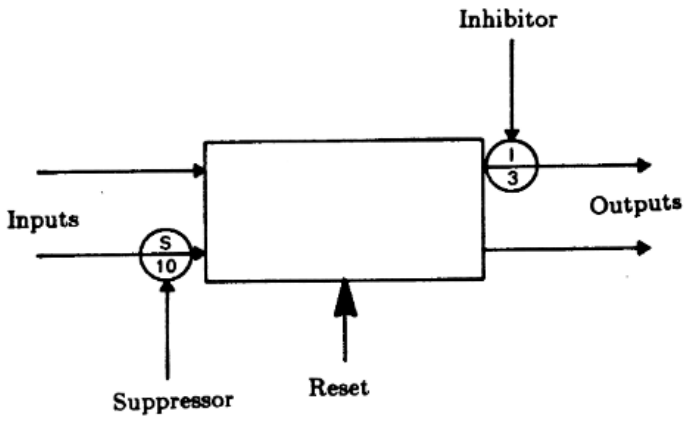


Figure 3.3: Schema representing a simple subsumption architecture model [62]

The individual behaviour of a robot consists of a simple Finite State Machine (FSM). The transitions from one state to another are governed by a response threshold model in the form of a probability function, as seen in 3.1 [62].

$$P = \frac{N}{N + k^{\beta}} \tag{3.1}$$

N represents a certain threshold, β represents a sensitivity parameter, and k stands for some stimuli received from social interaction or the environment.

3.1.3. Hybrid

The reactive architecture establishes a successful framework for robust mobile robot navigation. However, to autonomously perform well in the real world, both features from the deliberative and reactive architecture should be combined. This architecture is then a so-called 'hybrid'. There are three hybrid styles, the first is the *managerial* style. Here, the deliberative module plans the higher level and sends the information to a lower, reactive module to execute. A second method is called the *state-hierarchical* style. Here, knowledge of past, present, and future states is used. The deliberative layer can use the past state to predict the future (For example, the old state space is used for path planning of future motor actions). The reactive layer then functions in the present state, executing the commands. More commonly used hybrid models however to consist of three layers, this model was first introduced by Perez (2003) [63]. In this model. the deliberative layer is the highest layer, used to develop an optimal plan. Problems like sensor fusion, map building and planning are done here. These optimized commands are transferred down to the control execution layer. This layer supervises the interaction between the higher deliberative and lower reactive layer by issuing commands. The reactive layer is then used to generate simpler robot actions. Using a hybrid architecture forms a hybrid, flexible and robust architecture for control systems [61].

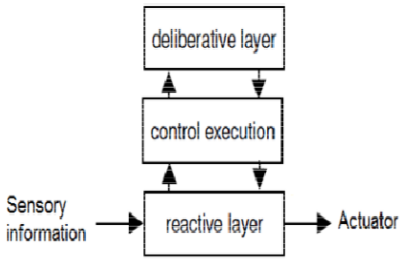


Figure 3.4: Commonly used hybrid control architecture for mobile robots [63]

3.1.4. Summary global robot architecture

The single robot architecture will be tasked with connecting inputs to output commands. The first architecture introduced was the SPA architecture. This is a top-down method where high-level constraints are broken down into smaller, manageable low-level commands. Unfortunately, this architecture is very slow and has no direct mapping of inputs to the model due to sensor noise. In an attempt to solve this, the subsumption architecture was introduced. Here, asynchronous low-demanding channels act as Finite State Machine modules or behavioural layers that interconnect sensors to actuators directly. Higher-level modules can inhibit lower levels from working. State transitions are managed by a probability threshold function. A combination of the SPA model and subsumption model results in a more flexible and robust architecture for robotic systems. There are many of these kinds of hybrid architectures. The most commonly used hybrid method is the one where the deliberative SPA layer develops a high-level plan. This plan is then communicated to the reactive, lower layer to execute. In addition, a control execution layer is put in between to govern if the plans generated by the highest layer are indeed executed by the lowest layer as requested. Swarm robots are often robots with low processing capability and simple sensors that need to solve complex tasks and actions. SPA requires a robot to have a good internal model which is not feasible for swarm robots. The subsumption architecture could be a good option for robots if only reacting, and not planning, is sufficient to navigate the world. If the robot is required to make a plan for its next set of actions, the hybrid method introduced by Perez (2003) [63] would be a flexible and robust alternative.

3.2. Control architecture

Swarm Robotics (SR) algorithms in multi-target search usually need some kind of positioning to communicate with their neighbours about their own position or the position of a target. Swarm robots are often modelled as a simple non-holonomic Differential Drive Mobile Robot (DDMR), capable of moving in a two-dimensional field. These types of robots have a good trade-off between complexity (i.e. costs and maintenance) and kinematic abilities. They are also able to turn on on the spot, reducing the room needed for manoeuvring [64].

3.2.1. Kinematics DDMR

Consider a DDMR robot moving in a two-dimensional plane with the following kinematics.

$$\begin{cases} \dot{x} = v_i \cdot \cos(\theta_i) \\ \dot{y} = v_i \cdot \sin(\theta_i) \\ \dot{\theta} = \omega_i \end{cases} \tag{3.2}$$

where $p_i = (x_i, y_i)^T$ is the Cartesian coordinate system of robot i under the global frame. θ_i is the steering angle or orientation. v_i is the driving, linear or translational velocity. Lastly, w_i is the angular or steering velocity. The robot input is often given as $(v, \omega)^T$ or the rotational velocity of the wheels (ω_R, ω_L). Moving from position p_1 to p_2 requires information about the orientation θ and position p_1 of robot 1. This movement is visualised in fig. 3.5. The desired position and orientation at p_2 then result in a turning angle β. Odometer data (incremental encoder data) is often used in addition to GNSS to estimate position and orientation. This kind of kinematics is widely used to define these kinds of robots in SR.

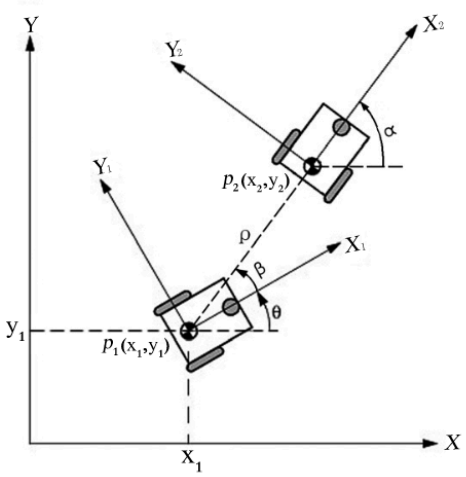


Figure 3.5: Kinematics local robot coordinate system in the global system, moving from p_1 to p_2 [64]

3.2.2. Control system DDMR

Whenever robots have to decide their own position and speed, a control system is required. A commonly used control system in robotics is the closed-loop feedback system like the one shown in fig. 3.6. Generalized coordinates of such systems are relatively simple $q = [x; y; \theta]$. The auxiliary control law makes sure the error distance between the current and desired position is minimized. The inverse kinematic controller (control laws) then outputs commands to the robot. This then translates into the desired movement. The robot needs to know its position inside the global frame for this control system to work.

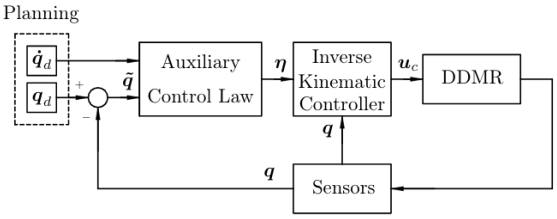


Figure 3.6: Control Architecture DDMR [65]

3.2.3. Control system DDMR with local sensing

Sending and receiving the position and orientation of neighbouring robots is not always possible. To solve this issue, [64] introduced a control system purely based on local sensing. The kinematics of the proposed system are first shown in fig. 3.7. Here, relative angle α_{ij} is the angle between robots i and j. Distance e_{ij} is the distance where robot i senses robot j.

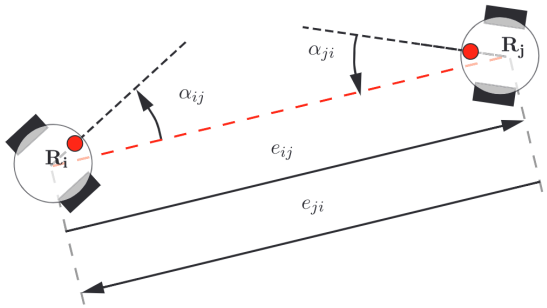


Figure 3.7: Local Sensing kinematic [64]

The control architecture is somewhat similar to fig. 3.6. The important thing to notice is that the controller output $[e_{ij} : \alpha_{ij}]^T$ is now influenced by the relative positioning between robot i, $q_1 = [x_i : y_i : \sigma_i]^T$ and robot j, $q_2 = [x_j : y_j : \sigma_j]^T$. Here, angle σ means the same as previously defined θ in fig. 3.6. Also, $N(i)$ represents the neighbouring set of the i-th robot. This means that robots only communicate with their perceivable neighbours.

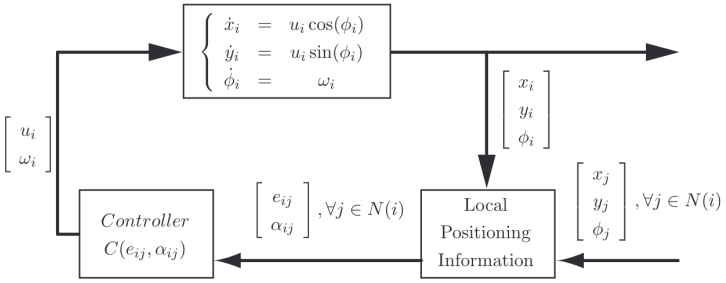


Figure 3.8: Control Architecture for DDMR [64]

3.2.4. Summary control architecture

Swarm robots are often modelled as a Differential Drive Mobile Robot. These types of robots can move in a two-dimensional plane, have a good trade-off between complexity and kinematic abilities and can turn around their axis. The basic kinematics and control systems of this robot assume that robots continuously share position and orientation. However, in many swarming applications, robots can send limited data, and some kind of local sensing control is necessary. The usage of local sensing could then suffice as a feasible individual robot control system in swarm robotics environments.

3.3. Swarm architecture

Controlling a swarm of robots to find multiple targets asks for efficient coordination and communication between these robots. This can be hard because robots in these environments often have limited communication capabilities. To call a system a swarming system, and not a multi-robot system, the system architecture must allow decentralized control and collective robot intelligence to be created. Therefore it is necessary to define an architecture that is fit for (i) decentralized control (ii) building individual and collective knowledge (iii) collective behaviour to emerge from individual robotic behaviour.

3.3.1. AMEB

Architecture Multi-robot systems heterogeneous robots with Emergent Behavior (AMEB) is an architecture specifically designed for managing swarm robots, allowing the emergence of self-organizing behaviour. The architecture is split up into an individual level, a collective level, and a level of knowledge management and learning. The general operation of this system is displayed in fig. 3.9. Robots use environmental stimuli to calculate a satisfaction index. This generated emotional state results in individual robot behaviour. Meanwhile, building individual knowledge. The combined actions of singular robots emerge into collective behaviour. Both individual robot knowledge and collective behaviour are then used to build collective knowledge. For example, robots individually discover target locations. These targets are then combined into the collective knowledge system. This 'total targets found' information can be used to decide on the performance of the swarm, guiding the system as a whole.

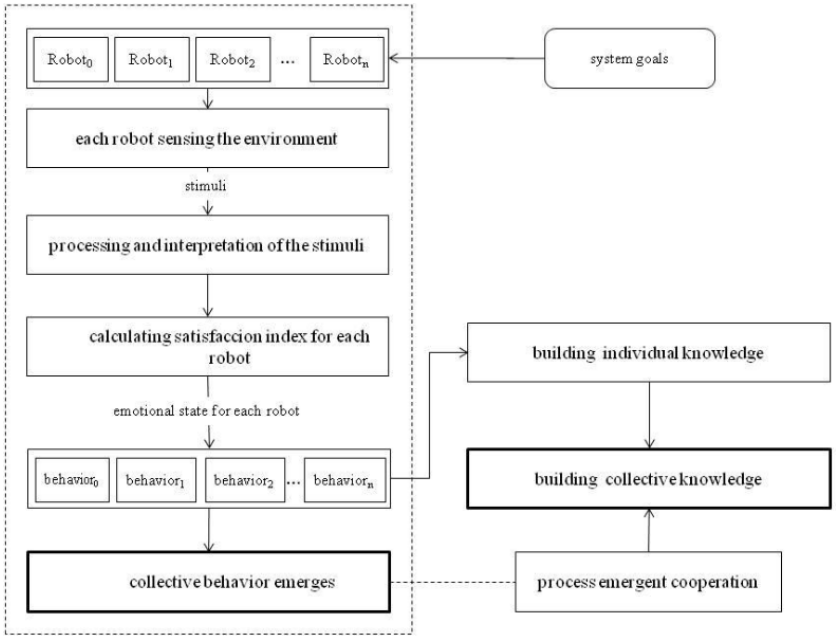


Figure 3.9: General operation of AMEB [66]

This kind of architecture has an emotion tied to the behaviour of the robots, as seen in fig. 3.10. The emotional model was initially proposed in [67, 68]. Joy for example drives the robot to be social and imitate other robots, whereas anger stimulates reactive and individual behaviour.

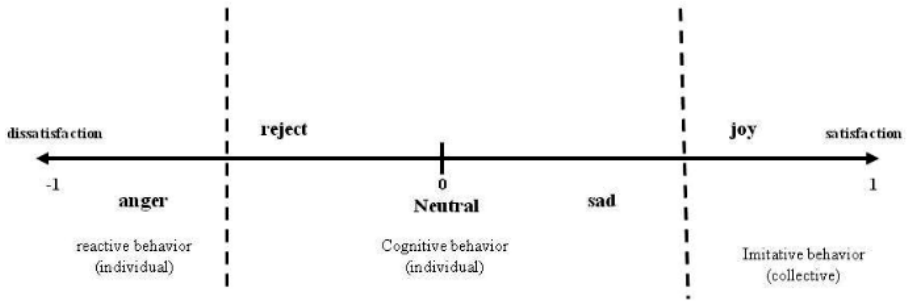


Figure 3.10: Emotional model [67, 68]

The emotional model is affected by environmental stimuli and the current robot's state. The robot state has been defined by [66] as the following:

1. Battery State (BS); energy level.
2. Operation State (OS); performance level.
3. Security State (SS); collision average.
4. Interaction State (IS); the number of sent or received messages

3.3.2. Summary swarm architecture

AMEB is an architecture designed for managing swarm robots. Many robotic swarming algorithms have a variable that balances exploration and exploitation. Often, this is managed by time, iterations or some mathematical (decreasing) function. AMEB enables robots to change their individual behaviour (exploration and exploitation) by introducing an emotional state. Robots decide their behaviour based on their battery state, operation state, security state and interaction state. Additionally, collective knowledge can be built by this framework. In Swarm Robotics (SR) individual behaviour should result in collective behaviour. This architecture is a good SR framework for any decentralized robotics swarm where collective knowledge is as important as individual knowledge.

3.4. Communication architecture

Effective communication is critical for Swarm Robotics (SR). The communication architecture is a broad concept, and starts from hardware and goes until the application of the data. In this chapter, the basics of data are explained (section 3.4.1). Also short and long range communication methods (sections 3.4.2 and 3.4.3) are given some attention. Next, Mobile Ad-hoc Network (MANET)s (section 3.4.4) are explained. Then the way by which nodes between robots are created (network routing, section 3.4.5) are researched, and finally how communication structures between robots are maintained (network topologies, section 3.4.6) is reviewed. These can all have a significant impact on the overall behaviour and performance of the swarm and this is therefore taken into account in the review.

3.4.1. ISO/OSI

To ensure standardised worldwide data communication, the International organization of Standardization (ISO) has come up with the Open System Interconnection (OSI) model. This model exists of 7 hierarchical levels represented in fig. 3.11.

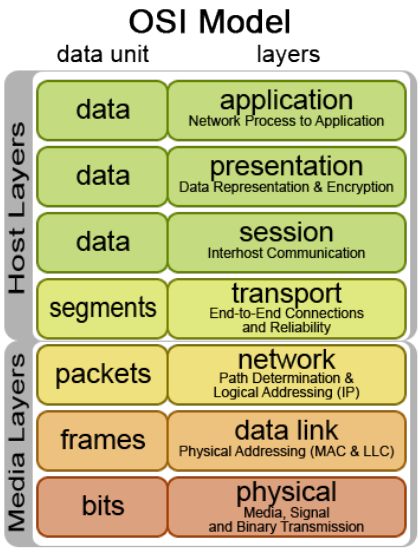


Figure 3.11: Caption

Physical layer

The lowest layer in the OSI framework is the physical layer. This layer is responsible to activate, maintain and deactivate physical connections. For example, it regulates voltages and data rates and converts digital/analogue *bits* into electrical, radio or optical signals. Here, it must be resolved how a single bit is to be transmitted. It also transmissions the reception of unstructured raw data between a device and a physical transmission medium. Devices and components associated are antennas, amplifiers and network sockets. Specifications of this layer define many aspects, including voltage levels, data rates, the timing of resetting, material usage, transmitting and receiving, pin layout, and more. Popular technologies used are CAN bus and USB.

Data link layer

This layer provides a synchronized link between two directly connected nodes (in local area network (LAN)) or adjacent network nodes (wide area network (WAN)) and detects and corrects errors that may occur. Sequentially transmitting and receiving data *frames* is therefore managed by this layer. The layer is split into how devices gain access to the network (MAC) and logical link control (LLC), which comprises networking protocols, with error checking and synchronisation. Protocol examples on the logical link control part of this layer are I2C, BUS, IEE 802.3 Ethernet, and IEEE 802.11 Wireless LAN. A popular architecture used in robotics is the Low Rate Wireless Personal Area Network (LR-WPAN) used by Zigbee/ WirelessHART and MiWi.

Network layer
The network layer transfers the variable data sequences (called *packets*) from one node to another using different channels that are connected in the network. It manages Subnet traffic. This layer also divides outgoing messages into packets while incoming packets are assembled into messages for higher levels. If data packets are too large, splitting packets can ensure they are still routed to their destination. Headers are attached to packets containing content, source and destination. An often-used networking protocol is IPv4/IPv6.

Tranport Layer
The transportation layer breaks variable data sequences into smaller units so that they are processed more efficiently by the network layer. The reliability of a link is controlled through flow control, segmentation/desegmentation and error control. In addition, *segments* are kept track of and retransmitted if delivery is failed. Commonly used protocols are the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).

Session Layer
The session layer is in control of inter-process communication between two systems by managing the connection between local and remote applications. Services like authentication, authorization and session restoration are employed. Full-duplex/half-duplex or simplex operations are possible. In addition, procedures can be suspended and restarted and sessions can be terminated.

Presentation layer
The presentation layer makes sure the receiver understands and is able to use the data. When two systems use different syntaxes this layer can be used as a translator. Data compression/encryption and conversion are also done here. Converting complex data structures into a stream of flat byte strings using mechanisms such as TLV or XML is the key functionality of the presentation layer.

Application layer
The application layer interacts with software applications, and mainly holds application programs to act upon received and sent data. Examples are Modbus and HTTP.

3.4.2. Short range wireless communication

For Swarm Robotics (SR) purposes, only communication protocols that use wireless technologies are researched. The methods discussed in this chapter are common methods for robot-to-robot communication. The methods that are compared are; ZigBee, Bluetooth and Wi-fi. Each with varying costs and capabilities. It becomes clear that there are different trade-offs in using one system over another. Wifi would be a good choice for fast and long(er) distance nodes, however, it has a high power consumption. If low power consumption is important, then ZigBee would be a better option. The different OSI layers are displayed in fig. 3.12. This shows that every protocol fits differently into the OSI model.

Protocol	Name	Nominal range (m)	Frequency bands (GHz)	Data Rate (Mbps)	Power Consumtion
IEEE 802.15.1	Bluetooth	50-150 ∼ often 10	2.4 to 2.485 ISM band	1 to 24 ∼often 1	Low
IEEE802.11a/b/g/n/ac	Wi-Fi	100-250 ∼often 20-40	2.4 and 5	5-3466 ∼often up to 600	High
IEEE802.15.4	ZigBee	10-100 line of sight	868, 915 MHz & 2.4 GHz	250 Kbps ∼often40-250 Kbps	Very Low

Table 3.1: Comparison of Bluetooth, Wi-Fi and ZigBee

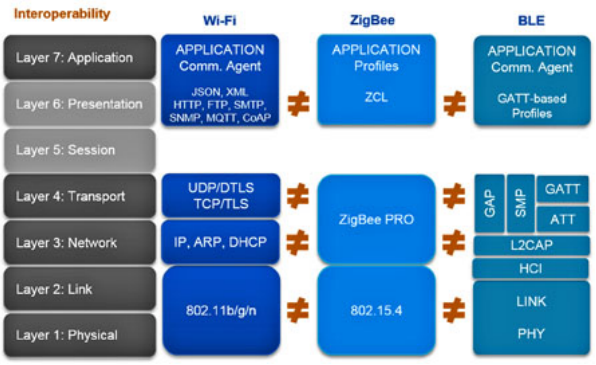


Figure 3.12: Wi-Fi, Bluetooth and Zigbee OSI/OSI stack view [69]

3.4.3. Long range communication

Sometimes a broader network is required. LoRa is a LPWAN technique that allows low-powered wireless communication. The frequencies used are orders of magnitude smaller than methods like Wi-Fi, increasing the transmission range. The low-power requirement of LoRa makes it ideal for battery-powered Internet of Things (IoT) devices. LoRaWAN is an example of a low power wide network based on this LoRa technique. Data rates are usually between 0.3 kbit/s and 27 kbit/s. In fig. 3.13 it is clear that these LPWAN techniques are applicable for Swarm Robotics (SR), when low data rates are sufficient and where long range is desirable. The pro with these LPWAN methods is also that a private network can also be set up, making these systems applicable in rural areas where there is no standard network coverage possible.

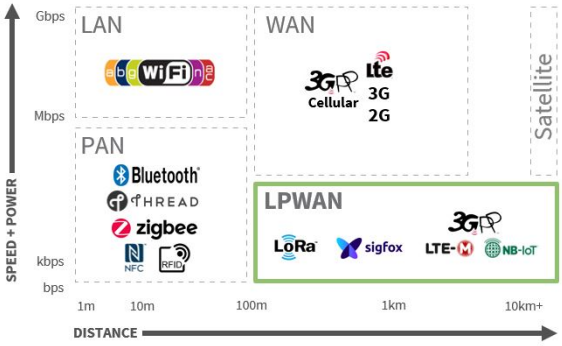


Figure 3.13: LPWAN techniques and their specification [70]

3.4.4. Mobile Ad-hoc Network (MANET)

In Swarm Robotics (SR), it is often desired to introduce a so-called 'neighbourhood', this is the range of a group of robots where they can sense each other, or communicate with each other. To enable robots to communicate within their neighbourhood, often an on-demand Mobile Ad-hoc Network (MANET) is set up [71]. Once a robot detects another robot, communication is initiated. Mobile ad-hoc networks consist of special kinds of wireless mobile nodes that form a temporary network without using any infrastructure or centralized administration. Robots can join or leave these networks at any time, enabling a swarm to scale up and down as needed. There is no fixed infrastructure, nodes are equal and there is no centralized overview or control. The difference between cellular and Ad Hoc networks are shown in table 3.2. Aspects that make this system valuable for swarm are rapid deployment capabilities, dynamic network topologies, the ability to function in hostile environments with noise and losses, adaptability of the network and of course the cost-effectiveness of the system.

Cellular	Ad Hoc Wireless Network
Infrastructure networks	Infrastrucureless networks
Fixed, prelocated cell sites and base station	No base station, and rapid deployment
Static backbone network topology	Highly dynamic network topologies with multihop
Relatively caring environment and stable connectivity	Hostile environment (noise, losses) and irregular connectivity
Detailed planning before base station can be installed	Ad hoc network automatically forms and adapts to changes
High setup costs and large setup time	Cost-effective and short setup time

Table 3.2: Ad Hoc Networks versus cellular networks [72]

3.4.5. Routing methods

[73] proposed routing methods for Multi-Robot Systems (MRS) that use Mobile Ad-hoc Network (MANET)s. Routing methods are important for Swarm Robotics (SR). They refer to the process of transmitting data and communication between devices in the network. For example, selecting the path or route that the data will follow based on distance, availability and reliability.

Reactive routing

The first method explained by [73] that is deemed fit for SR is reactive routing. A path between the source and the destination is only discovered on an as-needed base, rather than maintaining a pre-computed routing table. All transmission between source and destination is maintained until the path is no longer valid. This can be the result of links breaking. This method is useful where large numbers of nodes are required, such as in search and rescue, and environmental monitoring. Examples of these protocols are Dynamic source routing (DSR), ad hoc on-demand distance vector (AODV) and temporally ordered routing algorithm (TORA) [73]. Another example of such a method is the Ant-Colony based Routing Algorithm (ARA). Data can be forwarded from robot to robot using Ant Colony Optimization (ACO) methods [72]. The method is highly adaptive, efficient and stable and is found to be optimal for routing data between robots using mobile multihop ad hoc networks. Link existence is not guaranteed and links change often. Additionally, there are also some drawbacks to reactive routing. One potential downside is that it can introduce delays in communication, as the route discovery process can take some time. Reactive routing can also generate more traffic in the network, as nodes may need to perform multiple route discoveries to establish a connection.

Hybrid routing

In hybrid routing, both benefits of a fixed and a dynamic topology are used in separate conditions. Robots (nodes) are divided into clusters and cluster heads are assigned. Inside the cluster, proactive routing is used. This means a routing table is periodically maintained by the nodes. Here nodes are likely to have frequent exchanges and stricter movement strategies and this is therefore deemed necessary. The cluster head is in charge of transmitting and receiving information between the clusters by communicating with other cluster heads while using reactive routing, which is explained above. This approach is perfect for networks where clusters are formed based on the proximity between robots and between-cluster communication is less important, but still deemed necessary.

Location or geographic routing

In this type of routing, the location of each node is used to determine the best route for data and communication to be transmitted between nodes. Location/geographic routing is a method where nodes do not have to take notice of the topology of the system. The only important measure is the closest distance available for the next hop to the destination. [72] mentioned that this method would be useful in a place where MRS stretches larger areas where the network topology is in constant change. Protocols are for example Location-Aided Routing in mobile ad hoc networks (LAR), Energy-Efficient Location-Aided Routing (EELAR), greedy location-aided routing protocol (GLAR), and Location-Based Efficient Routing Protocol (ALERT).

3.4.6. Network topologies

In SR, network topologies refer to the arrangement and interconnection of robots in a swarm. The choice of network topology can have a significant impact on the communication, coordination, and overall behaviour of the swarm. Popular topologies are mesh, star, bus and ring topologies. Each of these topologies has its own characteristics and benefits, and the choice of topology will depend on the specific needs and goals of the swarm. Having the possibility to communicate with all members of the swarm does not mean they all need to be communicating with each other at all times. Zou et al. [39] compared three different communication topologies: ring, fully connected and adaptive random. Visualisation can be found in fig. 3.14. Points can be seen as connected devices (robots), and lines as connections between the robots. They found that there was no distinguishable difference or improvement among these models. This is because the number of robots is a much more influential variable, making the choice of topology negligible.

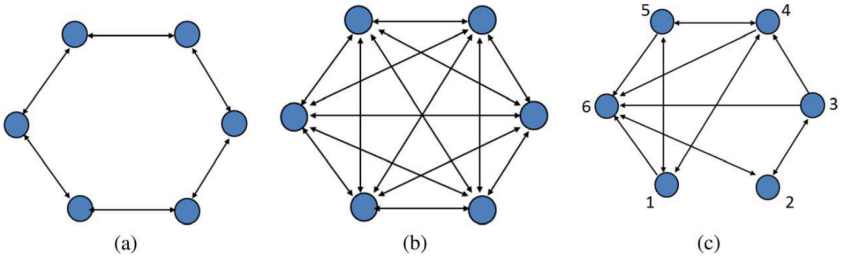


Figure 3.14: Graphs of topologies. (a) Ring. (b) Fully connected. (c) Adaptive random [39].

When several robots search for a single target, or if they are in each other's proximity, sub-swarms are often created. They are able to communicate information with each other. The three communication topologies mentioned in fig. 3.14 can then be applied for communication inside the swarm but also between swarms. [43] used this to their advantage, initiating sub-swarms by electing elite robots as cluster centres (robots with personal historical best positions that are equal to the best position of their cluster $p_b(n) = q_b(n)$) using a ring topology. Robots compare their fitness with their neighbours. Then, relocating the remaining robots choose their newly allocated cluster by using a star topology, e.g., all robots are connected and can share information with each other quickly. This way different communication methods can be utilized to their level of necessity. Another example has been given by [43], which used the immediate left and right neighbours to create a ring topology. The historical best position $qbest_i(t)$ at time t of robot i is updated using the best position between these neighbours, as seen in 3.3. Here $f(\overleftarrow{x}), f(\overrightarrow{x})$ are the immediate left and right fitness of the neighbouring particle or robot.

$$qbest_i(t) = argmax(f(x_i(t)), f(\overleftarrow{x}), f(\overrightarrow{x})) \tag{3.3}$$

An interesting communication architecture especially for target information truncation has been introduced by [7], the authors use self-organization and multi-task division between individuals of sub-swarms. Each target is seen as a sub-task, and every robot can only do one sub-task. A robot information card is constructed by dividing targets as $classI$ and $classII$, where $classI$ targets are targets directly classified by the robot in question. $classII$ targets are targets that are obtained by communicating with other robots in range. The robot will then have a target set while giving priority to $classI$ tasks. Sub-swarms are created by two or more robots having the same sub-task, these will also have a sub-swarm information card which is used to facilitate the acquisition of information within sub-swarms and the interaction between the sub-swarms.

Multi-Hop and Daisy chain

Multi-hop communication is an important concept in the field of Swarm Robotics (SR), as it allows robots in a swarm to communicate and coordinate their actions over long distances even without having the actual hardware to enable this. In multi-hop communication, a node uses another node to transmit data or communication through intermediate nodes, or ”hops,” rather than directly between the two nodes. This allows nodes to communicate with each other even when they are not within direct communication range. This chain of intermediate nodes is often also called a daisy chain. In the ring topology presented in fig. 3.15 (a) . A chain of communication is formed enabling data to hop from robot 3 (r3) to the base station (BS).

Rendezvousing

Rendezvousing refers to the process of two or more robots physically meeting at a predetermined location to transfer or exchange information. In rendezvousing, robots go from and to a pre-determined point and meet with exploration robots to communicate information, as seen in fig. 3.15. This is different then multi-hop, as robots are not in continuous communication with each other at all times.

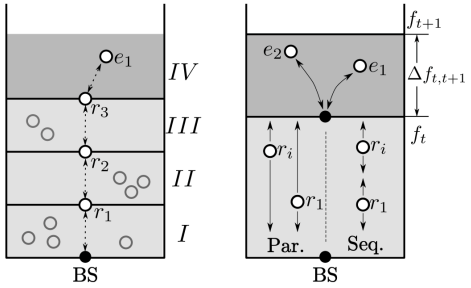


Figure 3.15: Multi-hop (left) and rendezvousing (right) of robots r_n and exploring robots e_n [74]

3.4.7. Summary communication architecture

A concise communication architecture has been standardized by International organization of Standardization (ISO) in something called the Open System Interconnection (OSI) model. Information is divided into 7 different layers. In robotics, a lot of plug-and-play solutions exist to maintain communication. The foundation of every method however defines its capabilities. Frequently used techniques like Wi-fi, ZigBee and Bluetooth have been compared with each other. The communication range, data bandwidth, data rate, power consumption and even price are considerations that decide which method is most fit for the use case.

In Swarm Robotics (SR) communication is often set up in something called a Mobile Ad-hoc Network (MANET) network. This enables robots to communicate within their neighbourhood, in a dynamic way. Robots can join or leave these networks at any time, enabling a swarm to scale up and down as needed. There are however many ways to make such a Mobile Ad-hoc Network (MANET) network. Reactive routing for example discovers communication paths on an as-needed base, requiring more network traffic but at the same time offering stability and adaptability. Another routing method is hybrid routing. Here, communication clusters are used. Every cluster has a cluster head that is in charge of exchanging information with other cluster heads and thus with other clusters. It is also possible to route information based on the position of a node. This removes the need to understand the communication topology of the system, basing this on a closest-distance-to-target measure.

Within (clusters of) a swarm, there are different communication methods possible. These are also known as network topologies. Most prominent are the ring/fully connected and adaptive random methods. The topology decides the way neighbours communicate with each other. This choice can be based on communication capabilities and the amount of communication necessary. To be able to truncate this information to a base station, or to unreachable members of the swarm, multi-hopping can be used. Here, robots transmit data through intermediate nodes or 'hops'. Another method to get the information trough is by using rendezvousing, here robots meet a pre-determined point, to communicate information. The robot is tasked with retrieving the information to the base and then moves back with this newly acquired information.

3.5. Obstacle and collision avoidance

In this chapter, different obstacle avoidance and collision avoidance methods that can potentially be used in multi-target swarm search in unknown environments are studied. Simpler avoidance methods are preferred over more complex ones, however, in unknown environments, too simple can also be dangerous. The algorithms that are described are the Bug algorithm (section 3.5.1), arc method (section 3.5.2), Particle Swarm Optimization (PSO) avoidance algorithm (section 3.5.3), fuzzy avoidance (section 3.5.4), braitenberg avoidance (section 3.5.5), virtual force method (section 3.5.6), vicinity checking (section 3.5.7), and trajectory checking algorithm (section 3.5.8).

3.5.1. Bug algorithm obstacle avoidance

The bug algorithm is a simple method of obstacle avoidance used by robots When a robot using the bug algorithm approaches an obstacle, it enters an obstacle avoidance state [75]. In the original bug-1 algorithm, the robot moves around the obstacle once and then continues towards the target location. The bug-2 algorithm improves bug-1 by allowing the robot to move directly to the target location without having to move completely around the obstacle. However, the bug algorithm is not effective at navigating through multiple smaller obstacles. Instead, it is best suited for avoiding large, convex-shaped static obstacles.. A visualisation of the bug-2 algorithm is given in fig. 3.16.

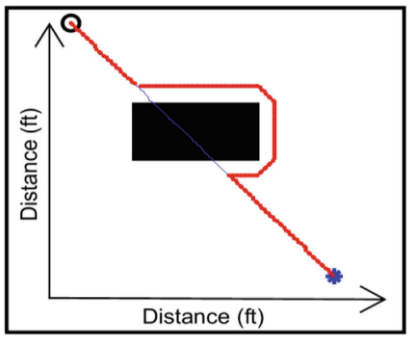


Figure 3.16: Bug-2 algorithm [75]

[39] used a variation of the bug-1 algorithm for static obstacle avoidance in PSO-based source seeking using a swarm of mobile agents. Instead of agents knowing the distance to the target, only the signal strength and their current position were known to the robots. The point with the strongest signal is chosen as the exit point from the obstacle towards the signal/target.

3.5.2. Arc method obstacle avoidance

In [76], an interesting obstacle avoidance method was presented. Whenever the next position of the particle or robot $x_{ij}(t+1)$ is in the boundaries of the object, an obstacle avoidance scheme is started. The centre point of a to-be-formed arc is put on the closest corner of the obstacle. The robot then follows the arc, around the obstacle.

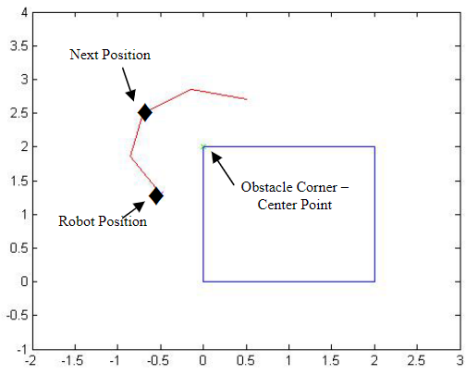


Figure 3.17: Arc function [76]

3.5.3. Particle Swarm Optimization (PSO) obstacle avoidance

When the PSO algorithm is used as a movement algorithm, it is very easy to adjust the algorithm to incorporate obstacle avoidance. [12] does this by adding an obstacle avoidance term $_3 \cdot r_{3j}(op_{ij}(t) - x_{ij}(t))$ to the velocity update equation of PSO (2.4). This changes the formula into eq. (3.4). Here, r_{3j} is a random stochastic variable in the range [0,1] and dimension $j = 1..,n_x$. $op_{ij}(t)$ is an attractive location, suitably located away from the obstacle. It can for example be located where the distance between the obstacle to the robot is maximised.

$$v_{ij}(t+1) = w_i v_{ij}(t) + c_1 \cdot r_{1j}(t) \cdot (pbest_{ij}(t) - x_{ij}(t)) + c_2 \cdot r_{2j} \cdot (gbest_j(t) - x_{ij}(t)) + c_3 \cdot r_{3j}(op_{ij}(t) - x_{ij}(t)) \tag{3.4}$$

c_3 is an obstacle sensitivity coefficient. When no obstacle is sensed, $c_3 = 0$. On the other hand, the closer the robot is to the obstacle, the more c_3 increases. [21] used a similar approach for the bat algorithm. This avoidance method works well for smaller objects [12]. It can also be used to avoid dynamic obstacles as it is continuously updating its velocity to move away from the obstacle.

3.5.4. Fuzzy obstacle and collision avoidance

Fuzzy avoidance is a rule-based approach. A fuzzy controller for obstacle avoidance in swarm robotics was proposed by [45]. Using fuzzy avoidance is often a simple, but effective avoidance method and can improve trajectory smoothness. Here, fuzzy inputs were; the angle of attack α_0, the previous linear velocity v of the robot. This then eventually results in the expected angular velocity w_e, and ve the expected linear velocity to turn away from the object. This process often starts with fuzzification. This means that the inputs are converted into membership degrees. For this specific study, the membership input functions are given in fig. 3.18. The outputs of the membership functions are defined in fig. 3.19.

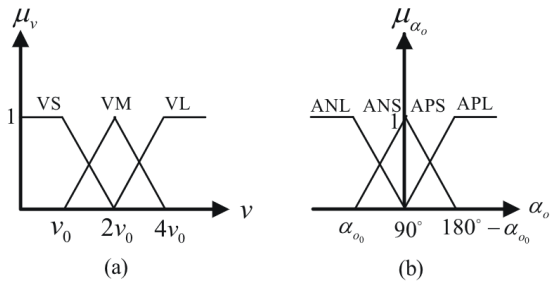


Figure 3.18: Membership functions of inputs μ_v (a), and μ_{α_0} (b), variables are explained in table 3.3 [45]

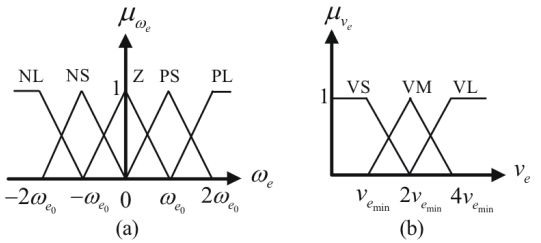


Figure 3.19: Membership functions of outputs μ_{w_e} (a), and μ_{v_e} (b), variables are explained in table 3.3 [45]

The rulebase is a set of if-then rules that correlate fuzzy input with the corresponding fuzzy output. This decision-making logic translates the fuzzy inputs into fuzzy outputs.

IF		THEN	
α_0	v	ω_e	v_e
ANL	VS	Z	VS
ANS	VS	PL	VL
ANS	VM	PL	VM
APS	VS	NL	VL
APS	VM	NL	VL
APL	VM	NS	VM
ANS	VL	PL	VS
ANL	VL	PS	VL
APS	VL	NL	VS
APL	VL	NS	VL

Table 3.3: ANS: Angle Negative Small; ANL:Angle Negative Large; APS: Angle Positive Small; APL: Angle Positive Large; VS: Velocity Small; VM: Velocity Middle; VL: Velocity Large; NL: Negative Large; NS: Negative Small; Z: Zero; PS: Positive Small; PL: Positive Large

Lastly, the defuzzification component turns these fuzzy outputs into non-fuzzy commands for the robot. This approach can deal with uncertain prospects [73] and save computation storage [77]. Because sensory inputs are converted into outputs directly, these avoidance controllers can avoid static and dynamic obstacles.

3.5.5. Braitenberg obstacle and collision avoidance

[17] used Braitenberg obstacle avoidance in multi-target search using a Bee Swarm Optimization-based algorithm. Braitenberg vehicles have sensors, often placed on the left and the right of the robot. Say a robot is moving towards a target position, and strongly senses an obstacle with its left sensor, and less with the right sensor. The sensor difference indicates an object to the left, directly controlling the left motor to turn faster than the right one. This will automatically direct the robot away from the obstacle. A visualisation of this is given in fig. 3.20. This kind of obstacle avoidance is simple and straightforward. Braitenberg obstacle avoidance is able to move away from static and dynamic obstacles. Inputs from the sensor can almost directly be used as input to direct the robot away, which makes it a low computationally demanding algorithm.

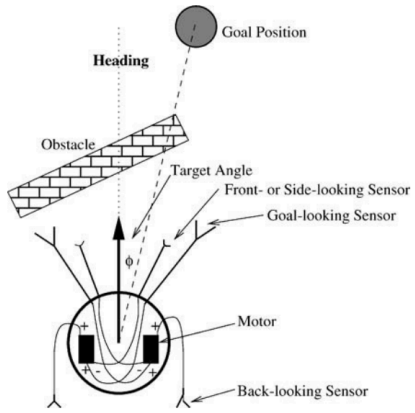


Figure 3.20: Braitenberg obstacle avoidance [78]

3.5.6. Virtual-force potential obstacle and collision avoidance

The virtual-force potential method can be used as a complete swarming algorithm, directing agents to targets, and away from each other and objects. It works by directing robots by adding attractive and repulsive force vectors caused by targets and objects/robots respectively. Only using repulsive vectors makes for a good collision and obstacle avoidance method. The algorithm adds a repulsive vector perpendicular to the velocity of the robot, and away from the (moving) object that is in the collision path, to the current trajectory of the robot. [44] uses a virtual-force repulsive vector for collision avoidance, the vector increases when robots come closer to each other. [18] used a simple but effective virtual-force obstacle avoidance method for multi-target search using Group Explosion Strategy (GES). Each robot checks if it will run into an obstacle with a certain velocity. If so, a small repulsive force perpendicular

to this velocity makes sure the collision will not happen. An example of the influence of the two white circled robots on potential v is given in fig. 3.21. d_1, d_2 is the distance between the two white robots and the current robot. σ is the desired distance of the robots. In this example, robots are desired to stay at a certain distance σ robot 1 repels the grey robot, and robot 2 attracts the grey robot. Together, a total force vector **f** is generated.

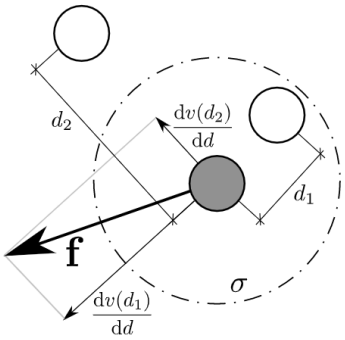


Figure 3.21: Virtual physics-based [79]

Virtual force obstacle and collision avoidance methods are able to calculate a merged avoidance scheme, taking care of multiple static and dynamic obstacles and other robots. Consequently, it is more complex than other avoidance schemes.

3.5.7. Vicinity obstacle and collision avoidance

In [40], robots must maintain a safe distance from each other. Warning zones are set as a circle around every robot and obstacle as seen in fig. 3.22 and fig. 3.23. Whenever a robot enters another warning zone, it initiates the obstacle avoidance scheme by rotating itself randomly. After several attempts, if the robot can still not find a suitable collision-free path, it then stays at its current position till the next iteration. This is a simple algorithm. This solution seems sufficient in a random walk task. However, when a clear target location is known, it may be redundant because the turning can move robots away from the target location.

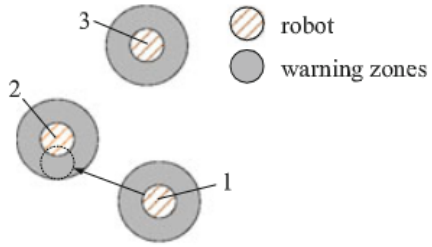


Figure 3.22: Warning zone robots [40]

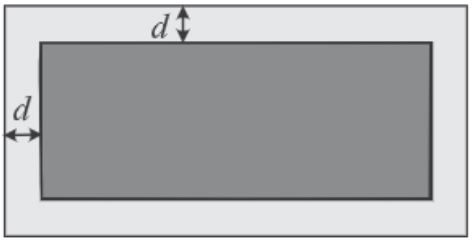


Figure 3.23: Warning zone obstacles [40]

3.5.8. Trajectory collision avoidance

An easy way for robots to avoid each other is to check if there are other robots on the path from p_n to p_{n+1} [6]. When a collision is detected, a random direction is given to the robot and the process is repeated. The algorithm is similar to section 3.5.7, however, it checks the actual path for collision and not around the robot. This enables robots to use fewer sensors than vicinity checking. It is a simple collision avoidance method which could also be translated to be used for obstacle avoidance.

3.5.9. Summary obstacle and collision avoidance

There are many different obstacle avoidance methods for robots moving in a 2-dimensional plane. The bug-algorithms (section 3.5.1) for example have a target in mind and avoid every object in their way. On the contrary, the arc method (section 3.5.2) just considers a local target and moves around it. These are more deliberative methods, requiring sensing, planning and acting. More reactive methods that rely on sensors are fuzzy obstacle avoidance (section 3.5.4) and Braitenberg obstacle avoidance section 3.5.5). Here, sensory information drives the robot away from objects. In a situation where robots are randomly searching, where it doesn't really matter where they are heading, vicinity obstacle and collision avoidance (section 3.5.7) and trajectory avoidance (section 3.5.8) could be an option. Robots randomly turn in a direction to avoid obstacles. On the contrary, where robots need to navigate terrain with many static and dynamic obstacles, a more intelligent avoidance system like virtual force obstacle (section 3.5.6) and collision avoidance would be the better choice. For pso-based algorithms (Particle Swarm Optimization (PSO), Bee Swarm Optimization (BSO), Bat Algorithm (BA)), it would be best to simply modify the velocity update equation in PSO to include obstacle avoidance (explained in section 3.5.3).

3.6. Simulation

Swarm Robotics (SR) requires realistic simulators to test and develop systems without the need for large investments. This is essential for speeding up the development of new algorithms. These simulators must model interactions between robots as well as the interaction of these robots with their environment in a realistic way [80]. It is often also desired that these simulators have some degree of compatibility with ROS [81]. ROS is an open-source robotics middleware, that provides low-level device control and messaging. Integrating simulators with ROS allows the evaluation of the many state-of-the-art robotic algorithms made by the ROS community while providing debugging tools and visualization. A review on physics simulators used for robotics has been done by [82]. The citation count per simulator in the range from 2016 to 2020 that they found is illustrated in fig. 3.24. The 5 biggest simulators that have been used are Gazebo, MuJoCo, CooppellaSim, CARLA, and Webots.

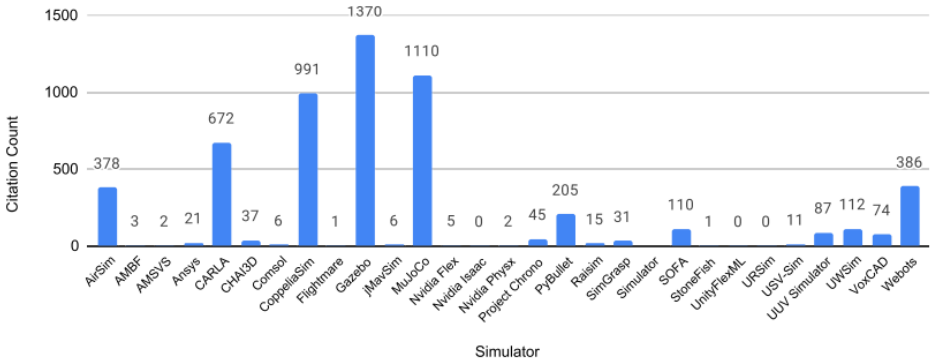


Figure 3.24: Citation count from 2016 to 2020 of papers found per simulator by [82]

The simulators mentioned above cover the entire robotic domain. Accordingly, a more concise search must be done on simulators capable of simulating mobile ground robots. Such comparison is made by [82] and displayed in fig. 3.25.

Simulator	GPS	Tracks	Wheels	Legs	Mecanum / Omni Wheels	Heightmap Import	OpenDrive / OpenStreetMap	Pathplanning	ROS Support	RGBD	LiDAR	Realistic Rendering
Gazebo	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗
CoppeliaSim	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗
Raisim	✗	✗	✓	✓	✗	✓	✗	✗	✗	✓	✓	✓, Unity
Webots	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗
PyBullet	✗	✗	✓	✓	✗	✓	✗	✓	✗	✓	✓	✗
CARLA	✓	✗	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓, Unreal
Project Chrono	✓	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓	✓, POV-Ray

Figure 3.25: Comparison of features between popular robotics simulator used for Mobile Ground Robots [82]

From this selection, the Raisim, PyBullet and Project Chrono simulators are not taken into further account because they do not enable robots to use path planning and GPS. [83] reviewed which simulators are best used for swarming purposes. These are displayed in fig. 3.26. Stage, Bio-PEPA, TeamBots, Swarm-bots, Gazebo, USARSim and ARGoS are fit to be used as simulation software for SR.

Software/Feature	Last Update	Free	Open-Source	Linux	MacOS	Windows	Swarm	Source (Accessed: 11 May 2022)
Stage	2020	✓	✓	✓	✓		✓	Repository: https://github.com/rtv/Stage
Bio-PEPA	2010	✓		✓	✓	✓	✓	Website: https://homepages.inf.ed.ac.uk/jeh/Bio-PEPA/biopepa.html
TeamBots	2000	✓	✓	✓	✓	✓	✓	Website: https://www.cs.cmu.edu/~trb/TeamBots/
Swarm-bots	2014			✓	✓		✓	Website: https://www.swarm-bots.org/
ODE	2022	✓	✓	✓	✓			Website: https://www.ode.org/
Gazebo	2019	✓	✓	✓	✓		✓	Website: http://gazebosim.org/
Webots	2021	✓	✓	✓	✓	✓		Website: https://cyberbotics.com/
MSRS	2015	✓				✓		Not available
USARSim	2013	✓	✓	✓	✓	✓	✓	Repository: https://sourceforge.net/projects/usarsim/
ARGoS	2022	✓	✓	✓	✓	✓	✓	Website: https://www.argos-sim.info/
CoppeliaSim	2022	✓	✓	✓	✓	✓		Website: https://www.coppeliarobotics.com/

Figure 3.26: Features of often used simulation software [83]

Based on the reviewed simulators in figs. 3.24 to 3.26, The Player project (Player/stage/Gazebo), CoppeliaSim, Webots, Teambots, Swarm-bots, and ARGoS are reviewed as potential swarm simulator for multi-target search in unknown domains by SR. Some simulators (CARLA, Bio-PEPA, USARSim) are not taken into further account because after some additional research, they were not applicable, or because they are not widely used making it hard to get into.

3.6.1. Player project simulator

The Player project is an open-source and free software platform for simulating and interpreting robotics and sensory systems. It consists of the Player-networking system, Stage- and Gazebo- robot simulators. Player is a network server middleware (like ROS) used for robotic control and is therefore not taken into further account.

Stage

Stage can simulate a population of mobile robots, sensors and objects that can orientate themselves in a 2(.5)D space. Stage is designed to help develop multi-agent autonomous systems in a simple, computationally cheap method by providing simple models of devices instead of trying to emulate them with great fidelity. Stage can be used It can be used in three ways:

1. As a standalone robot simulation program that loads robotic control based on a library that can be provided.
2. A stage plugin for player provides a population of virtual robots
3. Writing a simulator using C++.

Stage is fast and scalable. Player officially supports C++, C, and Python, however, third-player libraries enable using other clients ranging from Java to MATLAB. Robot controllers are compiled and loaded in run-time. ROS and Player can both be used as middleware.

Gazebo

Gazebo is also part of the Player project, adding 3D capabilities. It integrates ODE physics, OpenGL rendering, and support code for sensory stimulation and actuator control. It uses multiple physics engines such as ODE. Realistic environmental rendering of lightning, shadows and textures makes the simulation look real. The sensors can then 'see' this simulated world. The controllers written for stage can be used for Gazebo and vice-versa. The rest of the infrastructure is similar to Stage. Mesh manipulation is not available. Scene objects can be added and interacted with during simulation. Also, there exists a lot of support because of the large community that Gazebo has. According to [84], the UI usability is not that good. During simulations, Gazebo was able to simulate 10 robots and a large scene faster than in real time. Whenever this amount was increased to 50 robots in this large scene, the simulation ratio decreased to 1 second to 0.57 simulated seconds. Which is still good.

3.6.2. CoppeliaSim

CoppeliaSim, also known as V-REP, is a robotics simulator based on a distributed control architecture. This means every object or model can be individually controlled by a script, plugin, ROS node, remote API client or any other custom solution. This makes it ideal for multi-robot applications. The controllers can be programmed in C/C++, Python, Java, Lua, Matlab or Octave. Similarly to Gazebo, CoppeliaSim is a 3D simulation program. Simulating multiple robots can be computationally expensive. CoppeliaSim is ROS-compatible. The simulator is not fit to simulate many robots, as modeless are too detailed and it takes way to long [84].

3.6.3. Webots simulator

The Webots simulator is an open-source three-dimensional open-source mobile robot simulator. It is fit for multi-agent research. The platform enables users to model, program, simulate and eventually transfer a model to a robot. Features include wheeled, legged and flying robots.

1. Library of sensors and actuators
2. Programming through C++, Java, Python and Matlab using APIs
3. Transfers controllers to real mobile robots.

4. ODE library for accurate physics simulation
5. Enables multi-agent systems to be simulated with global and local communication facilities.

Similarly to Gazebo and CoppeliaSim, Webots is a 3D simulator.

3.6.4. ARGoS

ARGoS is a multi-physics robotic simulator. It is capable of simulating large-scale swarms of robots. Different environments (e.g. orbit, water flow, wind, rough terrain, bloodstream), robots (e.g., mechatronics, sensors, actuators) and communication methods can be simulated (e.g., wifi, vision, stigmergy, etc.) [85]. ARGoS is programmed in C++. The difference between ARGoS and Webots/Gazebo/Stage is that the physics engine is not the simulated space. Here, the physics engine and simulated space are distinct concepts. For instance, flying robots can be assigned to a 3D dynamics engine and wheeled robots to a 2D kinematics engine. Another example is where different parts of a room receive different physics engines. It is possible to develop the code in MATLAB/Simulink using the mobile robot package and use a third-party application to automatically create C++ code. It also easily runs 10+ robots in a large scene up to 1.59 times real time. Unfortunatly, ARGoS libraries only include a few robots, none which are interesting for outdoor mobile robot purposes.

3.6.5. TeamBots

TeamBots is based on a Java collection of applications and packages for multi-agent mobile robotics research. Execution on mobile robots sometimes requires C, Java is primarily used for higher-level functions. At the moment, TeamBots will run and simulate Nomadic Technologies' Nomad 150 robot. The simulator would be great for simulating that particular robot, however, for any other robot, it would be more straightforward to look for another simulator.

3.6.6. Swarm-bots

The swarm-bots simulator is made to be a full develop- and test-programming environment in which to design a robot control. It is a continuous 3D dynamics simulator of a multi-agent system. It can predict the dynamics and kinematics of a single s-bot (swarm bot developed especially for this simulator) in 3D. Distributed control algorithms can be simulated. Physical connections between two robots are created dynamically during a simulation and can be deactivated when the robots separate. Swarmbot3D was developed on top of Vortex. Similarly to TeamBots, the project would be great for simulating the s-bot, however, for any other robot, it would be easier to look into other simulators.

3.6.7. A selection of current robots

table A.1 shows a collection of mobile robots. These are compared in the following topics: average use run-time, the processor that is used, robot weight and price. Because the field of professional mobile robots is not as saturated as for example the drone markets, price and quality vary heavily. Scalability is a key concept in SR. Robots must therefore be affordable. 5 interesting robots are displayed in figure table 3.4.

	Price	Proccesor	Run-time	Speed	Weight
ROSbot 2.0 PRO (fig. A.6)	$3.199	Intel Atom x5 Z8350	3 hrs	1-1.25 m/s	2.85
Open Rover zero 3 (fig. A.16)	€3.343	Jetson NX Orin	4 hrs	2.5 m/s	11
Clearpath Jackal UGV (fig. A.7)	€17.493	Jetson AGX Xavier	4 hrs	2 m/s	17
Leo Rover (fig. A.17)	$4.799	RaspberryPi 4B 2GB	4 hrs	0.4 m/s	6.5
Panther (fig. A.11)	£16.034	Raspberry Pi 4B	7 hrs	2 m/s	55

Table 3.4: Selection of robots that are up to the task

3.6.8. Summary Simulation

According to comparisons made by [82, 83] and some further research, Stage, Gazebo, Cop- peliaSim, Webots, Teambots, Swarm-bots, and ARGoS are all fit for multi-target searching swarm simulation. There are many different simulators available that have pre-programmed models of various robots (Stage, Gazebo, CoppeliaSim, ARGoS, Webots) ready to be used. Other simulators are focused on one specific robot (Swarm-bots, Teambots). These are seen as unfit for this purpose as it limits the project. [84] researched the trade-offs between Gazebo, CoppeliaSim (V-REP) and ARGoS and concluded that ARGoS is suitable for swarm robotics tasks like collective foraging, flocking, or area coverage. It does unfortunately lack a variaty of mobile robots and is therefore not handy for fast development. Gazebo is more suitable for large swarm robotics experiments as it outperforms ARGoS in large environments, the simulator also has many robot models, plugins, modules and documentation that can be used. Webots is also an easy-to-use, well-documented simulator for developing a new swarming algorithm. [86] compared CoppeliaSim, Gazebo, MORSE and Webots and concluded that Webots should be used whenever simulation efficiency is very important and the simulation accuracy must be high. [87, 88] also used Webots to perform multi-robot simulation. Concluding, Stage, Gazebo, ARGoS and Webots are the go-to simulators for multi-robot, multi-target simulations in large scenes.

4

Problem Formulation and Research Plan

The world is still a vastly unexplored place. Many ancient civilisations have lived and perished, sometimes leaving traces of what once was. The job of an archaeologist is to study past people and cultures by excavating and examining remains. Some of these areas are however large and hard to reach. For example, Huari, which is one of the most significant archaeological sites in western South America. Huari is situated at an elevation that ranges between 2500-2900 meters above sea level, which makes it hard to breathe. Its enormous size also makes it hard to research [89].

Figure 4.1: Hilltop of Huaqanmarka in Huari-Peru [89]

In situations where the topography is extreme and sites are likely to only be found by searching accessible land, unsystematic field survey methods can be used. This involves archaeologists wandering through the survey area at will. Above-surface finds often are found by walking through the field, placing flags, and reporting finds to fellow archaeologists. Examples of artefacts are (shards of) jars, pots and coins. Surface artefacts may contain a great deal of information about the sub-surface and may indicate interesting areas that require more research. These initial findings can direct the search to smaller, more manageable and artefact-dense areas. As with other survey methods, further research is always needed to substantiate preliminary findings. After this initial survey, more accurate methods are often used to map the area, systematically field walk the area, do sub-surface measurements or even perform destructive excavation methods.

The problem boils down to the non-existence of a scalable solution that can identify anomalies/artefacts for mid-to-large areas while remaining affordable. The implication of the solution is in the initial part of the whole archaeological process and can range from finding the site in the first place to searching for areas of interest within a pre-defined space. A robotic swarm can possibly help by elevating archaeologists from this initial field survey phase, directing their time and resources onto areas of interest.

The problem statement can be formulated as follows: **'To develop an autonomous multi-target searching robotic swarming algorithm to alleviate archaeologists of field walking by identifying and mapping artefacts in an unknown environment'**. The task is a massive one, and the thesis will therefore only focus on the algorithmic part of finding multiple targets in such an environment. The research will not pay any further attention to manoeuvrability, complex obstacle avoidance methods, complex communication protocols, complex robotic control methods and not any other problems that arise with the topic. The following sub-objectives are formulated within the thesis table 4.1.

1. Design or choose a robotic swarming algorithm that is focused on solving the multi-target searching explained in the problem statement that archaeologist face.
2. Design an environment with targets that accurately reflect the problem described.
3. Implement the algorithm on a mobile robot while incorporating physical limitations (like battery level). How can the algorithm's efficiency be exploited?
4. Benchmark the algorithm against other well-known algorithms.

Task	**Week**	**Weeks needed**
Literature survey+ Algorithm design + Environment design	4-8	4
Inserting targets in environment	9	1
Adding the robots to the simulation	10-13	4
Implement the swarming algorithm	14-19	6
Optimize the algorithm	20	1
Benchmarking	21-22	2
Thesis writing	23-24	2
Green light	24	1
Thesis defense	26	1
Total	-	22

Table 4.1: Research plan

References

[1] D Guzzoni, A Cheyer, L Julia, and K Konolige. Many robots make short work: Report of the SRI International mobile robot team. *ojs.aaai.org*, 18(I), 1997.

[2] R Cassinis, G Bianco, A Cavagnini, and P Ransenigo. Strategies for navigation of robot swarms to be used in landmines detection. Technical report, IEEE, 1999.

[3] Vignesh Kumar and Ferat Sahin. Cognitive Maps in Swarm Robots for the Mine Detection Application*. Technical report, Rochester Institute of Technology, New York, 2003.

[4] Wisnu Jatmiko, Kosuke Sekiyama, and Toshio Fukuda. A PSO-based Mobile Sensor Network for Odor Source Localization in Dynamic Environment: Theory, Simulation and Measurement. Technical report, 2006.

[5] Jie Li and Ying Tan. A Probabilistic Finite State Machine based Strategy for Multi-Target Search Using Swarm Robotics. Technical report, 2019.

[6] Kurt Derr and Milos Manic. Multi-robot, multi-target particle swarm optimization search in noisy wireless environments. In *Proceedings - 2009 2nd Conference on Human System Interactions, HSI '09*, pages 81–86, 2009.

[7] You Zhou, Anhua Chen, Hongqiang Zhang, Xin Zhang, and Shaowu Zhou. Multitarget Search of Swarm Robots in Unknown Complex Environments. *Complexity*, 2020, 2020.

[8] Áron Kisdi and Adrian R.L. Tatnall. Future robotic exploration using honeybee search strategy: Example search for caves on Mars. *Acta Astronautica*, 68(11-12):1790–1799, 6 2011.

[9] Qing Hao Meng, Wei Xing Yang, Yang Wang, Fei Li, and Ming Zeng. Adapting an Ant Colony Metaphor for Multi-Robot Chemical Plume Tracing. *Sensors 2012, Vol. 12, Pages 4737-4763*, 12(4):4737–4763, 4 2012.

[10] Yoan Purbolingga, Achmad Jazidie, and Rusdhianto Effendi. Modified Ant Colony Algorithm For Swarm Multi Agent Exploration on Target Searching in Unknown Environment. In Yoan Purbolingga, Achmad Jazidie, and Rusdhianto Effendi, editors, *Modified Ant Colony Algorithm For Swarm Multi Agent Exploration on Target Searching in Unknown Environment*, Surbaya, 2019. IEEEXPLORE.

[11] Zedadra Ouarda, Jouandeau Nicolas, Seridi Hamid, and Giancarlo Fortino. Exploring unknown environments with multi-modal locomotion swarm. *Studies in Computational Intelligence*, 678:131–140, 2017.

[12] Hongwei Tang, Wei Sun, Hongshan Yu, Anping Lin, Min Xue, and Yuxue Song. A novel hybrid algorithm based on PSO and FOA for target searching in unknown environments. *Applied Intelligence*, 49(7):2603–2622, 7 2019.

[13] Xiaofang Yuan, Xiangshan Dai, Jingyi Zhao, and Qian He. On a novel multi-swarm fruit fly optimization algorithm and its application. *Applied Mathematics and Computation*, 233:260–271, 5 2014.

[14] K. N. Krishnanand and Debasish Ghose. Glowworm swarm based optimization algorithm for multimodal functions with collective robotics applications. *Multiagent and Grid Systems*, 2(3):209–222, 1 2006.

[15] K. N. Krishnanand and D. Ghose. A glowworm swarm optimization based multi-robot system for signal source localization. *Studies in Computational Intelligence*, 177:49–68, 2009.

[16] Reza Akbari, Alireza Mohammadi, and Koorush Ziarati. A novel bee swarm optimization algorithm for numerical function optimization. 2009.

[17] Hannaneh Najd Ataei, Koorush Ziarati, and Mohammad Eghtesad. A BSO-based algorithm for multi-robot and multi-target search. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7906 LNAI:312–321, 2013.

[18] Z Zheng and Tan Ying. Group explosion strategy for searching multiple targets using swarm robotic. *ieeexplore.ieee.org*, 2013.

[19] Xin-She Yang. A New Metaheuristic Bat-Inspired Algorithm, in: Nature Inspired Coop-erative Strategies for Optimization. 284:65–74, 2010.

[20] Guannan Li, Hongli Xu, and Yang Lin. Application of Bat Algorithm Based Time Optimal Control in Multi-robots Formation Reconfiguration. *Journal of Bionic Engineering*, 15:126–138, 2018.

[21] Hongwei Tang, Wei Sun, Hongshan Yu, Anping Lin, and Min Xue. A multirobot target searching method based on bat algorithm in unknown environments. *Expert Systems with Applications*, 141, 3 2020.

[22] Micael S. Couceiro, Patricia A. Vargas, Rui P. Rocha, and Nuno M.F. Ferreira. Benchmark of swarm robotics distributed techniques in a search task. *Robotics and Autonomous Systems*, 62(2):200–213, 2 2014.

[23] Peter Drewett. *Field Archaeology An Introduction*. Routledge, 2 edition, 2011.

[24] Erol Sahin. Swarm Robotics: From Sources of Inspiration to Domains of Application. Swarm robotics View project. *Conference Paper in Lecture Notes in Computer Science*, 2004.

[25] Manuele Brambilla, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo. Swarm robotics: A review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41, 3 2013.

[26] Ying Tan and Zhong yang Zheng. Research Advance in Swarm Robotics. *Defence Technology*, 9(1):18–39, 3 2013.

[27] Levent Bayindir Erol Şahin and Levent Bayindir. A Review of Studies in Swarm Robotics. Technical Report 2, 2007.

[28] Yixin Huang, Shufan Wu, Zhongcheng Mu, and Sunhao Chu. A Multi-agent Reinforcement Learning Method for Swarm Robots in Space Collaborative Exploration. 2020.

[29] Kumar Gaurav, Ajay Kumar, and Ramanpreet Singh. Single and multiple odor source localization using hybrid nature-inspired algorithm. 2020.

[30] Yara Khaluf, Eliseo Ferrante, Pieter Simoens, and Cristián Huepe. Scale invariance in natural and artificial collective systems: a review. *royalsocietypublishing*, 2017.

[31] Yara Khaluf, Stef Van Havermaet, and Pieter Simoens. Collective lévy walk for efficient exploration in unknown environments. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11089 LNAI, pages 260–264. Springer Verlag, 2018.

[32] A. Einstein. Über die von der molekularkinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilchen. *AnP*, 322(8):549–560, 1905.

[33] Bao Pang, Yong Song, Chengjin Zhang, Hongling Wang, and Runtao Yang. A Swarm Robotic Exploration Strategy Based on an Improved Random Walk Method. *Journal of Robotics*, 2019, 2019.

[34] Craig W Reynolds. Flocks, Herds, and Schools: A Distributed Behavioral Model. *Computer Graphics*, 21(4), 1987.

[35] R Eberhart and J Kennedy. Particle swarm optimization. In *Proceedings of the IEEE international*, 1995.

[36] Carl Zimmer. A school of bigeye trevally in Malaysia., 11 2007.

[37] Aniket Gupta, Aman Virmani, Parth Mahajan, and Raghava Nallanthigal. A Particle Swarm Optimization-Based Cooperation Method for Multiple-Target Search by Swarm UAVs in Unknown Environments. In *2021 International Conference on Automation, Robotics and Applications, ICARA 2021*, pages 95–100. Institute of Electrical and Electronics Engineers Inc., 2 2021.

[38] Ganesh K Venayagamoorthy, Sheetal Doctor, and Venu G Gudise. Optimal PSO for collective robotic search applications. *ieeexplore.ieee.org*, 2004.

[39] Rui Zou, Vijay Kalivarapu, Eliot Winer, James Oliver, and Sourabh Bhattacharya. Particle Swarm Optimization-Based Source Seeking. *IEEE Transactions on Automation Science and Engineering*, 12(3):865–875, 7 2015.

[40] Qirong Tang, Lu Ding, Fangchao Yu, Yuan Zhang, Yinghao Li, and Haibo Tu. Swarm Robots Search for Multiple Targets Based on an Improved Grouping Strategy. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 15(6):1943–1950, 11 2018.

[41] M Clerc and J Kennedy. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *ieeexplore.ieee.org*, pages 58–73, 2002.

[42] RC Eberhart and Y Shi. Comparing Inertia Weights and Constriction Factors In PSO. IEEE, 2000.

[43] Jun Qi Zhang, Yehao Lu, and Mengchu Zhou. Hybrid Topology-Based Particle Swarm Optimizer for Multi-source Location Problem in Swarm Robots. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 13345 LNCS, pages 17–24. Springer Science and Business Media Deutschland GmbH, 2022.

[44] Upma Jain, Ritu Tiwari, and W. Wilfred Godfrey. Multiple odor source localization using diverse-PSO and group-based strategies in an unknown environment. *Journal of Computational Science*, 34:33–47, 5 2019.

[45] Yifan Cai and Simon X. Yang. A PSO-based approach with fuzzy obstacle avoidance for cooperative multi-robots in unknown environments. In *2013 IEEE International Conference on Information and Automation, ICIA 2013*, pages 1386–1391, 2013.

[46] Yifan Cai and Simon X. Yang. An improved PSO-based approach with dynamic parameter tuning for cooperative target searching of multi-robots. In *World Automation Congress Proceedings*, pages 616–621. IEEE Computer Society, 10 2014.

[47] Yifan Cai and Simon X. Yang. A potential field-based PSO approach for cooperative target searching of multi-robots. In *Proceedings of the World Congress on Intelligent Control and Automation (WCICA)*, volume 2015-March, pages 1029–1034. Institute of Electrical and Electronics Engineers Inc., 3 2015.

[48] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 26(1):29–41, 1996.

[49] Marco Dorigo. *Optimization, Learning and Natural Algorithms.* . PhD thesis, Ph.D. Thesis, Politecnico di Milano, Italian. - References - Scientific Research Publishing, 1992.

[50] J.M. Pasteels, J.L. Deneubourg, and S. Goss. Self-organization mechanisms in ant societies. I. Trail recruitment to newly discovered food sources. *From individual to collective behavior in social insects : les Treilles Workshop / edited by Jacques M. Pasteels, Jean-Louis Deneubourg*, 1987.

[51] Thomas Stützle and Holger H. Hoos. MAX–MIN Ant System. *Future Generation Computer Systems*, 16(8):889–914, 6 2000.

[52] Bernd Bullnheimer, Richard F Hartl, and Christine Strauu. A New Rank Based Version of the Ant System-A Computational Study. 1997.

[53] Natalie J. Lemanski, Chelsea N. Cook, Brian H. Smith, and Noa Pinter-Wollman. A Multiscale Review of Behavioral Variation in Collective Foraging Behavior in Honey Bees. *Insects 2019, Vol. 10, Page 370*, 10(11):370, 10 2019.

[54] V. B. Meyer-Rochow. Glowworms: A review of Arachnocampa spp. and kin. *Luminescence*, 22(3):251–265, 5 2007.

[55] K. N. Krishnanand and D. Ghose. Detection of multiple source locations using a glowworm metaphor with applications to collective robotics. *Proceedings - 2005 IEEE Swarm Intelligence Symposium, SIS 2005*, 2005:84–91, 2005.

[56] Krishnanand N Kaipa, Amruth Puttappa, Guruprasad M Hegde, Sharschchandra V Bidargaddi, and Debasish Ghose. LNCS 4150 - Rendezvous of Glowworm-Inspired Robot Swarms at Multiple Source Locations: A Sound Source Based Real-Robot Implementation. Technical report, 2006.

[57] Wen Tsao Pan. A new Fruit Fly Optimization Algorithm: Taking the financial distress model as an example. *Knowledge-Based Systems*, 26:69–74, 2012.

[58] Yiwen Zhang, Guangming Cui, Jintao Wu, Wen Tsao Pan, and Qiang He. A novel multi-scale cooperative mutation Fruit Fly Optimization Algorithm. *Knowledge-Based Systems*, 114:24–35, 12 2016.

[59] Ying Tan and Yuanchun Zhu. Fireworks Algorithm for Optimization. *Springer-Verslag Berlin Heidelberg*, I:355–364, 2010.

[60] Eliseo Ferrante. A Control Architecture for a Heterogeneous Swarm of Robots. 2008.

[61] D Nakhaeinia, S H Tang, S B Mohd Noor, and O Motlagh. A review of control architectures for autonomous navigation of mobile robots. *International Journal of the Physical Sciences*, 6(2):169–174, 2011.

[62] R Brooks. A robust layered control system for a mobile robot. *ieeexplore.ieee.org*, 3 1986.

[63] Marc Carreras Pérez. A proposal of a behavior-based control architecture with reinforcement learning for an autonomous underwater robot. 2003.

[64] Riccardo Falconi and Il AA Relatore Claudio Melchiorri Claudio Melchiorri. Coordinated Control of Robotic Swarms in Unknown Environments. 2008.

[65] Julio C.Montesdeoca Contreras, D. Herrera, J. M. Toibero, and R. Carelli. Controllers design for differential drive mobile robots based on extended kinematic modeling. *undefined*, 11 2017.

[66] ÁEG Pérez, JLA Castro, RDJR Estrada, and ED González. A Control Architecture for Robot Swarms (AMEB). *Cybernetics and Systems*, 50(3):300–322, 4 2019.

[67] N. Perozo, J. Aguilar, O. Terán, and H. Molina. A Verification Method for MASOES. *IEEE transactions on cybernetics*, 43(1):64–76, 2 2013.

[68] Niriaska Perozo, Oswaldo Terán, and Jose Aguilar. Proposal for a Multiagent Architecture for Self-Organizing Systems (MA-SOS) Autonomic Computing Platforms View project Smart Classroom View project Proposal for a Multiagent Architecture for Self-Organizing Systems (MA-SOS). *LNCS*, 5075:434–439, 2008.

[69] Jim Harrison. IP on Wi-Fi, ZigBee, and Bluetooth: opportunity and risk - Electronic Products, 2016.

[70] Jim Wang. An introduction to wireless technologies in IoT- LPWAN | Allion Labs, 2023.

[71] Floriano De Rango, Nunzia Palmieri, Xin She Yang, and Salvatore Marano. Swarm robotics in wireless distributed protocol design for coordinating robots involved in cooperative tasks. *Soft Computing*, 22(13):4251–4266, 7 2018.

[72] Subir Kumar Sarkar, T.G. Basavaraju, and C. Puttamadappa. Ad Hoc Mobile Wireless Networks : Principles, Protocols and Applications. *Ad Hoc Mobile Wireless Networks*, 10 2007.

[73] Hasan Mehrjerdi, Maarouf Saad, and Jawhar Ghommam. Multi mobile robots formation in presence of obstacles. *ieeexplore.ieee.org*, 2011.

[74] Saeed. Welcome to ARW 2015! 2015.

[75] Muhammad Zohaib, Syed Mustafa Pasha, Nadeem Javaid, and Jamshed Iqbal. Intelligent Bug Algorithm (IBA): A Novel Strategy to Navigate Mobile Robots Autonomously. 11 2013.

[76] Lisa L Smith, Ganesh K Venayagamoorthy, and Phillip G Holloway. Obstacle avoidance in collective robotic search using particle swarm optimization. 2006.

[77] S Kundu and DR Parhi. Behavior-based navigation of multiple robotic agents using hybrid-fuzzy controller. *ieeexplore.ieee.org*, 2010.

[78] X Yang, RV Patel, M Moallem Journal of Intelligent Systems, Robotic, and undefined 2006. A fuzzy–braitenberg navigation strategy for differential drive mobile robots. *Springer*, 47(2):101–124, 10 2006.

[79] Lorenzo Garattoni and Mauro Birattari. Swarm Robotics. *Wiley Encyclopedia of Electrical and Electronics Engineering*, 2016.

[80] Erol Şahin, Sertan Girgin, Levent Bayindir, and Ali Emre Turgut. Swarm Robotics. In *Swarm Intelligence*, pages 87–100. Springer Berlin Heidelberg, 9 2008.

[81] Farzan M. Noori, David Portugal, Rui P. Rocha, and Micael S. Couceiro. On 3D simulators for multi-robot systems in ROS: MORSE or Gazebo? *SSRR 2017 - 15th IEEE International Symposium on Safety, Security and Rescue Robotics, Conference*, pages 19–24, 10 2017.

[82] J Collins, S Chand, A Vanderkop, and D Howard. A review of physics simulators for robotic applications. *ieeexplore.ieee.org*, 2021.

[83] C. Calderón-Arce, R. Solís-Ortega, and T. Bustillos-Lewis. Path planning on static environments based on exploration with a swarm robotics and RRG algorithms. In *Proceedings of the 2018 IEEE 38th Central America and Panama Convention, CONCAPAN 2018*. Institute of Electrical and Electronics Engineers Inc., 12 2018.

[84] Lenka Pitonakova, Manuel Giuliani, Anthony Pipe, and Alan Winfield. Feature and performance comparison of the V-REP, Gazebo and ARGoS robot simulators. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10965 LNAI:357–368, 2018.

[85] Carlo Pinciroli. The ARGoS Website, 2022.

[86] T. Barfoot, J. Burgner-Kahrs, E. Diller, A. Garg, A. Goldenberg, J. Kelly, X. Liu, H. E. Naguib, G. Nejat, A. P. Schoellig, F. Shkurti, H. Siegel, Y. Sun, S. L. Waslander, and . Making Sense of the Robotized Pandemic Response: A Comparison of Global and Canadian Robot Deployments and Success Factors. 9 2020.

[87] Zhe Ban, Junyan Hu, Barry Lennox, and Farshad Arvin. Self-Organised Collision-Free Flocking Mechanism in Heterogeneous Robot Swarms. 2021.

[88] Lucas Hop June. Efficient Swarm Robotic Persistent Surveillance by use of Stigmergy, 2021.

[89] Lidio M. Valdez and J. Ernesto Valdez. From Rural to Urban: Archaeological Research in the Periphery of Huari, Ayacucho Valley, Peru. *Journal of Anthropology*, 2017:1–14, 2 2017.

[90] ROSbot 2.0 PRO | ROS Robots.

[91] ROSbot 2.0 | ROS Robots.

[92] Jackal UGV - Small Weatherproof Robot | Clearpath, 12.

[93] Husky UGV - Outdoor Field Research Robot | Clearpath.

[94] LIMO | Agilex Robotics.

[95] Bulldog mobile robot-outdoor | Fidelrope.

[96] Panther - outdoor AMR | Husarion.

[97] Jaguar 4x4 Mobile Platform | Dr Robot.

[98] Hamster | Cogniteam.

[99] MĪTI | Rover Robotics, Inc.

[100] mini | Rover Robotics, Inc.

[101] 4WD Rover Zero 3 | Rover Robotics, Inc.

[102] Leo Rover - Kell Ideas company.

[103] Copernicus Robot | Botsync.

[104] 0x-ALPHA | Nex Robotics.

[105] 0x DELTA | Nex Robotics.

[106] SUMMIT-XL | Robotnik®.

[107] Seekur Jr | Adept Mobilerobots.

[108] Pioneer 3-AT | Adept Mobilerobots.

A

Appendix

A.1. Taxonomies of Swarm Robotics

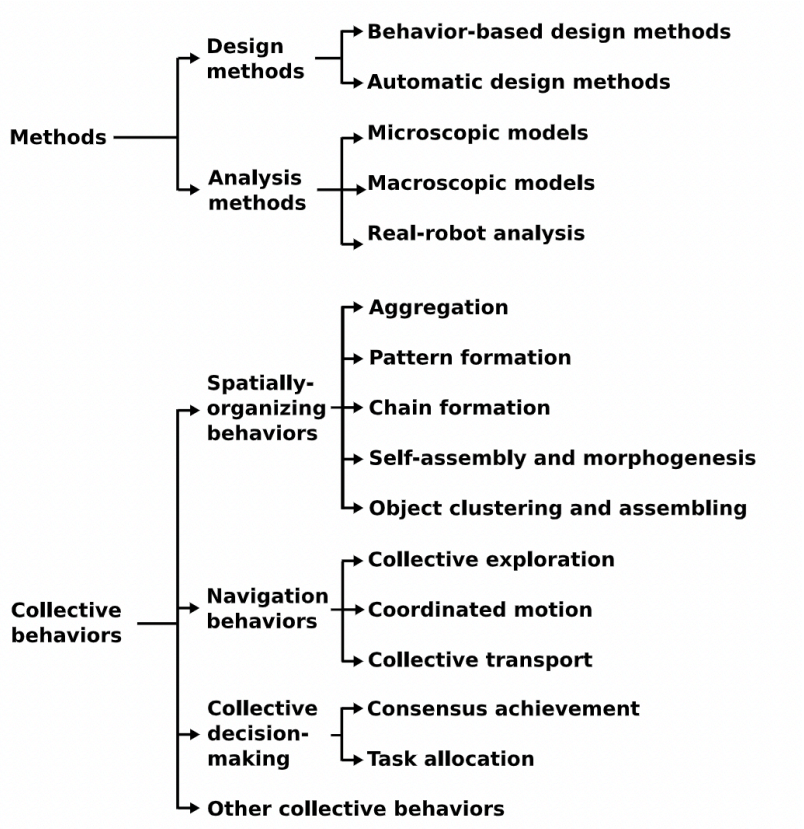


Figure A.1: Taxonomy of swarm behavior and methods by Brambilla et all. (2013) [25]

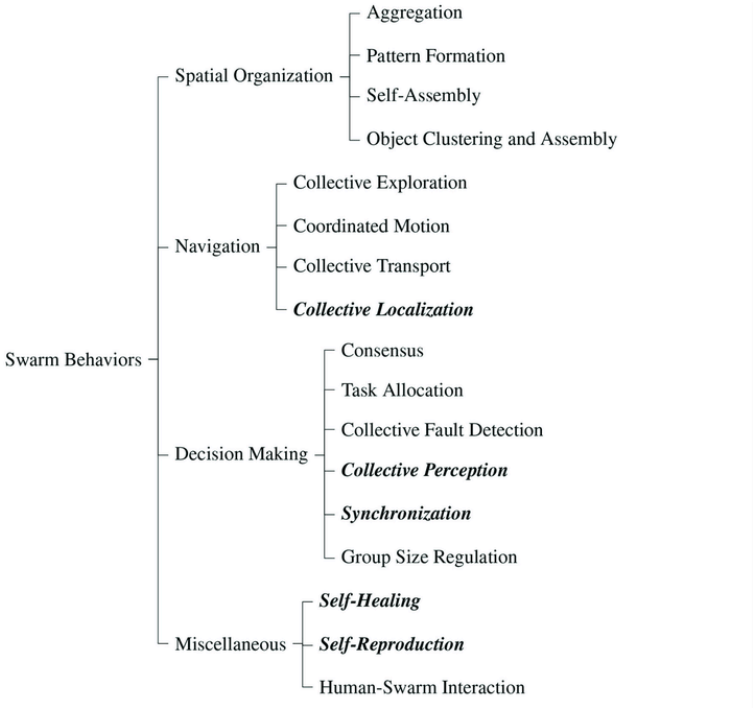


Figure A.2: Taxonomy of swarm behavior by Brambilla et all. (2013) [25]

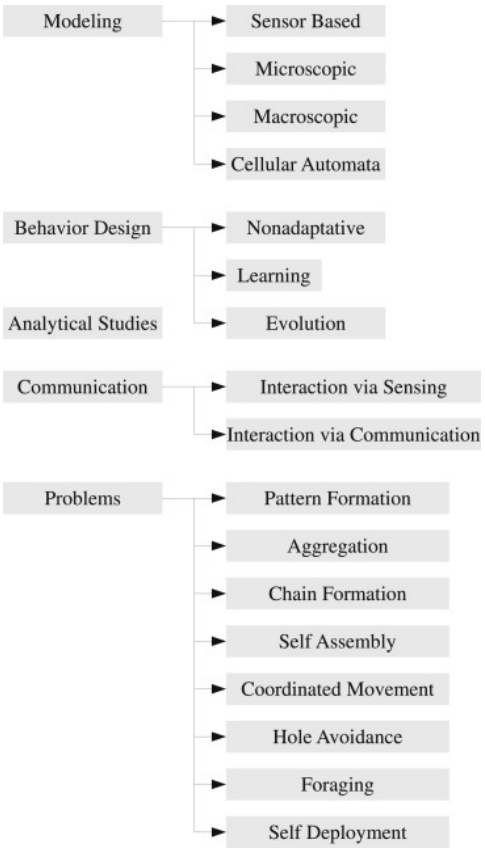


Figure A.3: Taxonomy of swarm systems by Bayindir et all. (2007) [27]

A.2. Subsumption architecture

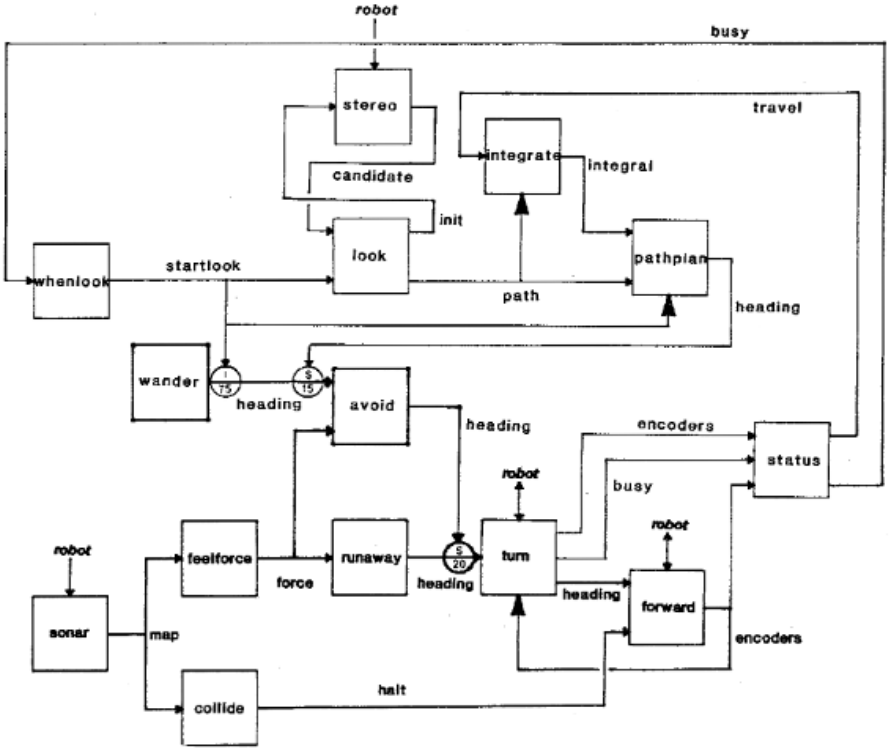


Figure A.4: Elaborate level 2 subsumption architecture model introduced for a mobile robot [62]

A.3. Examples of available robot

	Movement	Processor	Weight (kg)	Run-time (hrs)	Min price
ROSbot 2.0 [90]	Wheels	Raspberry Pi 4	2.85	3	$2.399
ROSbot 2.0 PRO [91]	Wheels	Intel Atom x5 Z8350	2.85	3	$3.199
Clearpath Jackal UGV [92]	Wheels	Jetson AGX Xavier	17	4	€17.493
Clearpath Husky UGV [93]	Wheels	Jetson AGX Xavier	50	3	€25.200
LIMO [94]	Tracks, Wheels	Jetson Nano (4G)	4.2	40 min	$400
Bulldog [95]	Wheels	-	60	3	$58.600
Panther [96]	Wheels	Raspberry Pi 4B	55	8.5	£16.034
Jaguar [97]	Tracks,Wheels	-	25	2	$11.500
Hamster [98]	Wheels	Rapsberry PI (V6)	1.5	90 minutes	$4.900
Open Rover MITI [99]	Wheels	Jetson NX Orin	-	5	€3.433
Open Rover mini [100]	Wheels	Jetson NX Orin	-	4	€2.384
Open Rover zero 3 [101]	Wheels	Intel NUC I3	11	5	€3.343
Leo Rover [102]	Distanced wheels	RaspberryPi 4B 2GB	6.5	4	$4.799
Copernicus Robot [103]	Wheels	Fitlet 2/ Jetson Nano	75	3	-
0x ALPHA [104]	Tracks	Intel Core i3 processor	50	3	-
0x DELTA [105]	Wheels	Intel Core i3 processor	44	4	-
Summit-XL [106]	Distanced wheels	CPU ROS (Intel i7)	65	5	€ 17.622
Seekur Jr [107]	Wheels	Phytec MPC-565/Mamba EBX-37	77	8	-
Pioneer 3-AT [108]	Wheels	Phytec MPC-565/Mamba EBX-37	12	3	$34,495

Table A.1: Robots and their different attributes. All information is estimated. Contact the companies for accurate up-to-date data

Figure A.5: ROSbot 2.0 [90]

Figure A.6: ROSbot 2.0 PRO [91]

Figure A.7: Clearpath Jackal UGV [92]

Figure A.8: Clearpath Husky UGV [93]

Figure A.9: LIMO [94]

Figure A.10: Bulldog [95]

Figure A.11: Panther [96]

Figure A.12: Jaguar [97]

Figure A.13: Hamster [98]

Figure A.14: Open Rover MITI [99]

Figure A.15: Open Rover mini [100]

Figure A.16: Open Rover zero 3 [101]

Figure A.17: Leo Rover [102]

Figure A.18: Copernicus Robot [103]

Figure A.19: 0x ALPHA [104]

Figure A.20: 0x DELTA [105]

Figure A.21: Summit-XL [106]

Figure A.22: Seekur Jr [107]

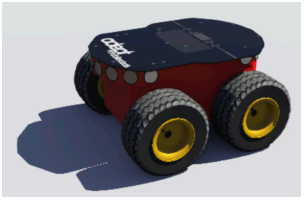


Figure A.23: Pioneer 3-AT [108]