Designing the Concurrent Searching and Pathfinding Algorithm for Swarm Robotics

Rob Steven van den Berg

Delft University of Technology

Designing the Concurrent Searching and Pathfinding Algorithm for Swarm Robotics

Master's Thesis in Embedded Systems

Department of Microelectronics
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

Embedded Networked Systems group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

Rob Steven van den Berg

27th February 2023

Author
Rob Steven van den Berg

Title
Designing the Concurrent Searching and Pathfinding Algorithm for Swarm Robotics

MSc presentation
To be defended publicly on Monday, February 27, 2023, at 15:00
Delft University of Technology
the Netherlands

Graduation Committee

Dr. R.R.V. Prasad	Associate Professor, TU Delft
Dr. C.J.M. Verhoeven	Associate Professor, TU Delft
Dr. R.T. Rajan	Assistant Professor, TU Delft
Dr. A.Y. Majid	(Former) Postdoctoral Researcher, TU Delft

Preface

The Zebro project is a project by the TU Delft to develop a robust robotic swarming platform that can navigate rough terrain. This effort has culminated in a modular, six-legged, mass-producable robot called the Zebro. In this thesis the goal is to develop pathfinding behaviour for this robot, so that the Zebro can perform its first decentral swarming behaviour. The problem of finding a target and forming a path to it is known and is well-explored in the field of swarm robotics. Using swarm intelligence, even swarms of simple robots are capable of solving this problem. However, often these works assume that two robots can perceive each other can physically reach each other, or that robots can detect any target within communication range. These assumptions do not hold for all swarms and environments. This thesis introduces the CSP (Concurrent Searching and Pathfinding) algorithm to make path finding possible in simple swarms without relying on these assumptions. By strategically constructing and reconstructing an ephemeral network of stationary robots and relocating the rest of the robots throughout this network, swarms performing CSP can efficiently find targets with minimal risk of fracturing the network. The effect of packet loss, and swarm size, as well as the the execution time vis-à-vis creation of new anchors are shown. Through this research, not only is the Zebro robot is one step closer to performing productive missions as a swarm, many other robotic swarms and applications can also benefit from CSP, because it depends on very few hardware requirements and makes few assumptions about the environment.

Contents

Chapter 1

Introduction

Co-operation has been happening in nature for millions of years in many different ways. If you look around today there is a good chance that you will see a flock of birds flying over, find a beehive while exploring the woods, or have to deal with a colony of ants in your backyard. It is also not just something that happens in animal species. Mutualistic relationships are present in bacterial species too, and humans have built entire civilisations around the principle of working together and trading. Evidently, there is a lot to be gained in different individuals working together towards a common goal. In the past century an entirely new kind of ”species” has been making its way into the world: robots. Over the years quite some impressive and useful robots have been built and programmed, although these robots usually work alone while attempting to accomplish a single goal. The perhaps obvious but relatively novel next step is to have independent robots work together in a swarm.

The Zebro robot (from now on simply reffered to as *Zebro*), developed by the Zebro team at the Delft University of Technology, is a six-legged robot that is designed specifically with swarming activities in mind. Its relatively cheap manufacturing cost, its modular design and its ability to traverse rugged terrain are all important features to enable various swarming missions. There are already multiple functional Zebro units that can walk around and avoid obstacles. Much of the required hardware is in place to allow for swarming, however, no form of swarming behaviour has been implemented yet.

An ideal demonstration of swarming behaviour should be a sufficiently complex task the Zebros that is similar to an actual use case for a swarming robot like the Zebro. This thesis aims to introduce a pathfinding algorithm that keeps in mind the limitations of the Zebro robot. This specific problem has been chosen because it is a problem that can theoretically be solved by a group more quickly than by an individual (and than by a group of individually acting participants) if proper communication and behavioural rules are enforced. The goal is to have the robots in our pathfinding algorithm make it clear to anyone in a single look that a path has been found, by arranging the robots' bodies along the found path.

Although the problem of searching has been well studied for swarm robotics, often assumptions are made that would make the solution not compatible with the Zebro robots' limitated capabilities (see section 3.2). This thesis designs and implements the CSP (Concurrent Searching and Pathfinding) algorithm to achieve the desired behaviour.

Fundamental constraints.

1. This work does not assume that a communication link being present between two robots means that there is a physically traversable path between the robots. As a consequence, our method does not need to rely on line-of-sight communication.
2. The distance from which robots can sense the target is significantly smaller than the range in which robots can sense/communicate with each other.
3. The robots are sent to an unfamiliar environment without any idea of terrain or obstacles.

Assumptions.

1. The robots in this work can move forward with a speed of 10cm/s and turn with a differential drive system. Obstacles can be avoided using ultrasound detectors.
2. For communication purposes, they rely on a wireless communication module with a range of 3m.
3. It is assumed that the robots can localize themselves relative to other robots without a pre-existing setup in the environment, for example through single anchor localization[1].
4. The target is a virtual point in space which can only be detected *only* when a robot is within 25cm.

Method. Rather than having the swarm start in the searching state upon being turned on, this work instead uses a leader election approach that dynamically allows the swarm to collectively switch to a task. The swarm starts in an idle state, and any robot may spontaneously decide to start the searching behaviour by becoming the nest. The rest of the swarm will then also switch to the searching task, which starts the collective searching mission from this nest to the target. If the target is found, the swarm will demonstrate the existence of the path from the nest to the target by all positioning themselves along this path.

1.1 Contributions

1. This work introduces the Concurrent Searching and Pathfinding (CSP) algorithm, which finds the object of interest with the stringent constraints listed above. An implementation of this method has been implemented in C++, built upon the ARGoS simulator [2].
2. CSP has been designed to dynamically switch to searching behaviour and decentrally decide upon the nest without any centralized control. CSP provides the path that has been found using the robots themselves.
3. An investigation into the efficiency of CSP for different searching parameters is provided.
4. An investigation into the robustness of CSP against packet loss and node failures is provided.
5. Results of the efficiency of the swarm for different swarm sizes and environments are provided.

1.2 Thesis outline

The thesis is seperated into several chapters as follows:
First, chapter 2 will provide the necessary background information for this thesis: first some definitions will be given, followed by an analysis of swarming behaviour in nature and an overview of related work in the field of swarming robotics. This chapter will conclude with a formal statement of the research question for this thesis. Chapter 3 will give an introduction of the Zebro project: its origins, history and the capabilities of the Zebro robot will be assessed. After that, chapter 4 will describe how the resulting CSP algorithm came to be by first giving a problem analysis (accompanied by concrete targets), and then goes into the considerations and decisions that were made in the process of implementing the swarming behaviour before describing the inner workings of CSP. How well CSP performs is then assessed in chapter 5. Finally, 6 will draw conclusions and discuss possible future work.

Chapter 2

Background

This chapter will give an overview of the background information for this thesis. First, some definitions will be given, such as what a robotic swarm *is*. After these definitions, a summary of swarming behaviour in nature and its takeaways will be given in section 2.2, followed by a summary of related work in search swarming algorithms in section 2.3. Finally, a description of the problem this thesis aims to solve is given, which is expressed as a research question in section 2.4.

2.1 Definitions

Here, definitions and neccesary information will be given that are needed to understand the rest of this document.

2.1.1 Robotic swarm

This thesis uses E. Şahin's definition for robotic swarms [3]: *Swarm robotics is the study of how large number of relatively simple physically embodied agents can be designed such that a desired collective behavior emerges from the local interactions among agents and between the agents and the environment.*

The main characteristics of a swarm robotics system are [4]:

- The robots in the swarm are autonomous
- The robots are situated in an environment and can interact with it
- The robots' sensing and communication capabilities are local
- The robots do not have access to centralized control and/or to global knowledge
- The robots cooperate to tackle a given task

This means that not every group of robots is a robotic swarm, even if they work together in to achieve a common goal. The simplest way to have a group of robots perform a single task is to have a central point of intelligence - a sort of master that knows the entire state of the group of the swarm and sends out commands that the group carries out. An implementation like this is not a robotic swarm as it violates the characteristic that robots in swarms do not have access to centralized control. Shortly put, robotic

swarms are *decentral*: the knowledge and decision making of the swarm is scattered across the swarm.

Swarms are often defined to have at least 100 members. However, Şahin reasons that although studies focusing on just a group of several robots that does not take scalability in mind should not be considered robotic swarms, a lower bound would be difficult to justify and that groups of 10-20 robots are acceptable as swarms. Because huge swarms are costly to produce, testing with large groups of robots can be an obstacle. Therefore it is deemed important by Şahin not to exclude studies that are carried out with a small number of robots, but have the vision or promise of scalability in sight.

2.1.2 Swarm intelligence

Swarm intelligence is the act through which a swarm can produce highly structured collective behaviours and complex tactics of the swarm as a result of simple interactions between the individuals of the swarm and the environment. Section 2.2 gives some concrete examples of how swarm intelligence is present in nature in for example the flocking of birds (subsection 2.2.1), where the birds only employ simple rules to keep a certain distance from their direct neighbours, and in the foraging behaviour of ants (subsection 2.2.2), where an external spatial memory is kept through the use of pheromones that ants leave behind. Even though the individuals in these swarms are behaving in a fairly simple way, they are able to accomplish a complex task as a collective. This is why employing swarm intelligence can be such an effective strategy: instead of having one or a few costly individuals perform a complex task, costs can be lowered and risks can often be reduced by having these tasks handled by a large group of simple, low-cost individuals instead. This is deeply rooted in the field of swarming robotics, to the point of it being implied even in the definition of swarming robotics (see the definition of swarm intelligence in the previous subsection): swarm intelligence is the main ingredient that can give robotic swarms an advantage over a single complex robot and master-slave implementations.

2.1.3 Argos

ARGoS is a *multi-physics robot simulator* that runs under Linux and MacOSX. The simulator focuses on efficiency as well as flexibility, so that it allows for complex simulations that have multiple robots in customizable environments. This makes ARGoS a viable platform for simulations of robotic swarms. ARGoS is modular in nature which means the physics engine(s), controllers, sensors, actuators, loop functions and visualisations can individually be implemented and altered. Simulations are run in a customizable 3D environment and can be paused, their speed can be increased and decreased, and they allow for easy access to the virtual robots' debug logs at any point. ARGoS comes pre-packaged with several virtual sensors, actuators, robot types and robot controllers, that can be altered or used as-is to simplify the process of building a desirable swarm simulation. ARGoS is used to verify the functionality of CSP and to test it in various scenario's and with varying parameters.

2.1.4 Nest

The starting place of the searching mission is often referred to as the nest in works that focus on searching behaviour for swarm robotics. The nest can either be a seperate robot or construction, or one of the robots

in the swarm that has become stationary and functions as the nest. In the solution proposed by this thesis, the latter is the case. In this work, the nest also functions as an anchor.

2.1.5 Anchor

An anchor or anchoring robot is a robot that has become stationary and functions similar to pheromone (see section 2.2.2): the robot is a recognition point for other robots. Being an anchor is a role that any robot can take on. Anchors can also stop being an anchor, with exception of the nest.

2.1.6 Backbone

Anchoring robots are often used to build up chains and tree like structures that mimic pheromone trails. In this work, the network of anchors is referred to as the backbone.

2.1.7 Physically traversable path

A path between two robots is deemed a physically traversable path if the space between the robots is walkable by a robot using simple obstacle avoidance behaviour with reasonable certainty. That means that there are no major obstructions such as walls or cliffs separating the robots that make travelling from one robot to the other directly impossible.

2.2 Swarm behaviour in nature

Nature is often taken as an example in swarm robotics. Swarms of animals have had millions of years to evolve and perfect their swarming behaviour, which is why these swarming behaviours in nature are often used as inspiration for swarm behaviour implementations in robots. In this section, two interesting, widely present forms of swarming behaviour in nature will be explored: flocking and swarming. The observations made from these forms of swarming behaviour will then be used to give more insight into the key principles behind swarming in section 2.2.3.

2.2.1 Flocking

A well-known and well-studied occurrence of swarming behaviour is flocking. Schools of fish, flocks of birds, as well as several other species can move in such an organized and integrated manner that they appear to be behaving as one coherent entity, even though the group may change shape and/or direction along the way. Several studies have used computer simulations to show that such collective behaviour can be demonstrated by groups without the need for group leadership, hierarchical control or global information.

The simplest of these distributed flocking models only assume that individuals in a group move at a constant velocity and that they take the average direction of motion of their peers within a local neighbourhood as their own direction [5, 6]. Although the minimalism of these models is very useful for analysis, it comes at the cost of realism, because the individuals in such simulations do not avoid collisions and do not exhibit attraction or repulsion towards others. These shortcomings prevent the forming

of self-bound groups (such as a school of fish or a flock of birds) when the individuals make any kind of error in the decision making, which makes these models incapable of fully explaining the swarming phenomenon that is being exhibited in nature.

Several authors have developed models where the individuals exhibit local repulsion, alignment, and attraction based on the position and orientation of their neighbours, resulting in flocking behaviour in various possible configurations [7, 8, 9], still without the need of global information. Following their approach, Couzin *et al.* developed a model in 3D space for swarming behaviour, where individuals have behavioral "zones" that determine how they respond to each other [10]. Summarized, the model consists of two rules:

1. Individuals in the swarm try to keep a minimum distance between themselves and others at all times to avoid collisions. This rule has the highest priority.

2. When individuals are not in the process of avoiding a collision (rule 1), then they will be attracted to their neighbours and align themselves with their neighbours.

This behaviour was achieved by having a three non-overlapping zones (see figure 2.1). Individuals in the swarm always try to move away from neighbours in their *zor* (zone of repulsion). When not performing an avoidance maneuver, the individuals align themselves with their neighbours in the *zoo* (zone of orientation), as well as being attracted to neighbours in the *zoa* (zone of attraction).

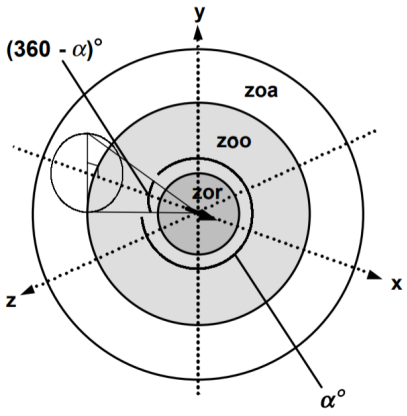


Figure 2.1: Representation of an individual in the grouping model centered at the origin, travelling in direction x. *zor* = zone of repulsion, *zoo* = zone of orientation, *zoa* = zone of attraction. The possible "blind volume" behind an individual is also shown. α = field of perception. From [10], reprinted with permission.

The results of running this model show that there are four fundamental grouping types that the flock can assume. Groups change rapidly between these states because the intermediate group types are unstable. The results also show that relatively small changes in individual responses can lead to large variations in the group's properties.

Another important principle that was discovered is that the history of the structure of the group has an effect on the collective behaviour that is exhibited as individual interactions change. Because none of

the individuals have explicit knowledge of what this history is, a form of ”collective memory” is present, which is an example of swarm intelligence (see section 2.1.2).

These studies showed, among other things, that the fundamentals of group formation in a swarm do not rely on external stimuli. However, such stimuli are present in the real world and can play an important part in explaining the formation of the shape of the group. When one participant of the flock turns because of a stimulus, the alignment tendency of its neighbours allows this change to be propagated through the swarm, which can lead to a change in the shape and behaviour of the group. This enables participants of a swarm to perform an avoidance maneuver around an attacker, without directly detecting the attacker itself.

Simulations where a predator and a grouping swarm can detect each other and respond accordingly exhibit characteristic collective patterns that correspond with those seen in nature, such as ”flash expansion”, where individuals rapidly move away from the predator as it strikes, ”vacuolation”, where a cavity forms in the swarm around the attacker, as well as the ”split effect”, where a group becomes fragmented [11]. Another important observation is that the size of the volume in which individuals respond to others has big implications for the collective avoidance behaviour. The smaller this volume is, the more the individual will behave like an independent to the others. This increases the chance for individuals to become nonaligned and for the group to fragment. As the size of the volume increases, individuals will respond to more of their neighbours. On one hand, this leads to a higher quantity of information to which individuals have access, and an increase in the speed of information through the group, because individuals can now respond not only to their nearest neighbours, but also those that are farther away. On the other hand, as the volume gets bigger, the quality of information individuals perceive about the other may also decrease, because the position and orientation of individuals that are farther away can be less important. This corresponds with the claims of proponents of self-organization theory that animals do not need long-range information to coordinate group behaviour [12]: localizing information can allow for better sensitive responses to predators and environmental obstacles.

2.2.2 Foraging

Foraging (the act of searching for wild food resources) is widely prevalent in nature. Specifically the foraging behaviour of social bees and ants have been well studied, partly because of their experimental convenience. Bees [13] and ants [14, 15] both live in communal nests and often travel across learned routes between important places. For example, for bees these foraging tasks involve extracting nectar from flowers in various locations and returning to their nest.

Ants and bees are able to learn complicated visually-guided routes and forage far from their nests and still find their way back. This is accomplished by comparing their current sensory input against a memory to generate an *output vector*, which encodes the direction towards their destination [16]. There appear to be three distinct types of mechanisms involved in the memory based guidance of insects [17]:

Alignment image matching
The simplest of these two mechanisms, alignment image-matching, employed by for example ants, relies on snapshots, which are encodings of visual information [18]. The mechanism has the ant align itself

in such a way that the visual image it sees closely matches an earlier experienced memory of the same view. As long as the ant is in the correct place and uses a matching memory, the mechanism ensures that the ant will then travel in the correct direction. [19]

Positional image matching
Like alignment image matching, positional image-matching also relies on snapshot memories. Computer simulations [20] have demonstrated that it is possible to compute a direction towards a goal by comparing the current visual image against a snapshot memory taken from the location of the goal. This can be done as long as the set of elements that is visible in both images is sufficient. All positions from where this is the case together form the *catchment area* [21]. Usually, the more open the environment, the bigger the catchment area [22].

Path integration
The third type of mechanism is path integration. It is quite different from the other two mechanisms, because it does not require on visual input or familiarity with the environment. That is why this mechanism enables insects to return to their nest after exploring new terrains. The mechanism uses a variety of cues to keep track of an individuals own direction and the distance it has travelled. Directional cues can be derived from the insects' celestial compass, where for example the position of the sun can be used [23]. Distance, on the other hand, can be estimated by counting the insect's own steps [24].

Although path integration is a useful tool to allow insects to explore new environments, it is vulnerable to errors caused by passive displacement. For example, the wind could blow an ant away which would then invalidate the path integration calculated route home. This is why insects often use a combination of several techniques to increase the reliability of their navigational capacities [25].

An interesting thing to consider is how insects employ communication in their foraging efforts. When one individual in a group finds a place of interest such as a food source, it is beneficial to have a way to let other foragers know the location of that food source. Different types of insects have evolved different methods to accomplish this communication. First up, there are straight-forward techniques where insects directly communicate to each other where the food source is. For example, the honeybees have developed a communication process called the *waggle dance*. This dance, performed by an individual, encodes the information needed to navigate to the new location, which the other honeybees can interpret and then use to navigate to the new location, despite never having been there before. Apparently, honeybees can under certain circumstances even use this location information to directly fly between two locations, without having to make a detour to the nest first [26]. It appears ants have developed an entirely different technique to accomplish the communication of an important location, perhaps because ants cannot fly over obstacles and are therefore required to take more complex routes. Ants are known to leave behind pheromone trails from their newfound profitable food source to the nest instead. Other ants are then able to follow this trail to end up at the food source. When they also find food there, they will in turn also leave behind pheromones, which will make the original trail stronger. This makes the pheromone tactic a feedback mechanism, because as long as there is food at the site, the pheromone trail gets stronger, which attracts more ants, but when there is no more food at the site, the pheromone trail won't get reinforced with more pheromones, which causes the trail to naturally degrade over time [27].

2.2.3 Swarming principles

Four key insights can be derived from the previous examples of how flocking and foraging behaviour appears in nature, that apply to swarming behaviour in general:

Simple individuals can form complex collective behaviour
The first important observation is that in both of these examples, all of the participants of the swarm perform very simple behaviour as an individual. However, as stated by [12], these rather simple interactions among simple individuals together produce highly structured collective behaviours. This is one of the reasons why swarming can such can effective strategy: it allows a group of simple individuals to perform feats that the participants of the group would struggle to do, or be completely incapable of performing, on their own. By working together, new tactics (such as external memories and communal navigation) become possible. These forms of more complex intelligence that become possible only through the cooperation of simple individuals are *swarm intelligence*.

Use communication sparingly
The second key insight is that most swarms seem to barely use direct communication, but instead rely heavily on observation and indirect communication. Save for the waggle dance, the tactics employed in flocking and foraging barely use communication but instead mostly rely on observation and external memories. Birds do not tell each other where to go, but instead use their observations of their neighbours' position and orientation and their observations of the world around them to determine where to position themselves in the swarm. By not relying on direct communication, many problems such as misinterpretation and transmission errors are less prominent or even completely absent. On top of that, animals do not have to develop the complex mechanisms to enable efficient direct communication in the first place if they do not require it, which allows them to remain simpler.

Decentrality
The third key insight is that none of the individuals in these swarms know the complete state of the group. Although it would seem intuitive to assume that a flock of birds knows exactly what shape they're forming, the reality is that none of them know the complete shape. There is no leader or central point of intelligence either; they are instead fully *decentral*, only making decisions for themselves based on their local situation. The collective swarming behaviour that forms is merely a consequence of a large number of these local interactions and decisions.
In a central environment there is a leader that makes decisions for the whole group. In order to be able to do this, the leader would need high computational power. This in turn means that all participants of the swarm are required to have high computational power, since all members of the swarm have the potential to become the leader. This means that all individuals in the swarm need to be relatively complex, which means they have a high energetic production cost. Instead, in a decentral group, all individuals can be simpler and therefore have a lower energetic production cost, which means they can also take high risks without a high cost for the group. Also, Conradt & Roper [28] suggest that there are several other advantages to decentral decision making, such as that democratic decisions are more beneficial because they tend to produce less extreme decisions than centrally made decisions.

Self-balancing

The last key insight is that rather than relying on perfect execution of the tasks, the natural swarms seem to be great at stabilizing themselves. This can be seen in the way insects use backup mechanisms for navigation, but the best example is flocking once again. Section 2.2.1 mentions how the simplest models that mimic flocking worked decentrally, but did not allow for any error margin, or for variable speed of the individuals. However, in reality errors in decision making are widely prevalent, and small variations in speed and orientation happen all the time. The reason why the flock does not always immediately break apart when this happens is because of the self-stabilizing nature of the flocking behaviour. The simple rules of keeping a minimum distance from others, moving closer when you are too far away from the others, and taking on the average orientation of your neighbours, and performing these corrections constantly, makes it so that the flock can stay together in imperfect conditions. A small gust of wind might leave a few individuals lagging behind a little bit, but they will then catch up on their own. Even when the flock does fragment because of an obstacle or another unforeseen circumstance, they have the capability to regroup again. The unpredictable nature of the real world forced swarms to evolve to take on this more efficient strategy of self-balancing.

These insights explain why collective behaviour can be a desirable tactic, and why it can therefore also be better to have a robotic swarm solve some problems, rather than one complex, expensive robot. Why put an expensive robot in a risky situation when a horde of simple, cheap robots can accomplish the same objective? In many situations, a swarm can also simply be quicker or more efficient at performing a task than an individual. Section 2.3 will talk about a few examples where robotic swarms are already used to solve problems. Whatever the case may be, when designing a robotic swarm, it will be important to strive to implement the key insights listed in this section.

2.3 Related work

There are quite a few examples of integrated robotic systems that work together, some of which exhibit traits that are similar to robotic swarms. However, more often than not these systems still rely on a central point of intelligence that commands the inidividual robots, which means they do not fall into the category of robotic swarms. Two examples of such systems are:

1. Robotic warehouse systems: In 2018 Alibaba's logistics affiliate Cainiao Network opened China's largest robotic warehouse in Wuxi, equipped with close to 700 automated vehicles that can autonomously pick up and drive around shelves in the warehouse [29]. These automated vehicles communicate with each other to avoid collisions and autonomously drive to a charging dock when their battery gets low. However, these vehicles receive specific orders for which shelves to pick up and where to deliver them and simply carry out these instructions, which excludes them from the field of swarming robotics.

2. Drone swarm formations: A large group of drones can be coordinated to take on certain formations and fly a pre-determined route. A notable example of this is at the Olympic Winter Games PyeongChang 2018, where intel broke the Guinness World record for "most unmanned aerial vehicles airborne simultaneously" with their drone light show featuring 1218 drones that formed patterns

in the sky such as the olympic rings [30]. Alhough these drones are a large group of unmanned robots that together carry out the singular task of drawing patterns in the sky, the fact that they are controlled by one computer and one drone pilot excludes them from the field of swarming robotics.

Designing actually decentral robotic swarm systems is challenging because the design requirements are specified on the collective level while the implementation is done on the individual robot's level and the desired behavior should emerge from the interactions between the robots. Brambilla et al. [4] give a comprehensive overview of the state of the field. According to Brambilla et al., there exist two types of approaches for designing a robotic swarming system: the behavior-based (or manual) approach and the automatic approach [4]. This work resides in the field of behaviour-based swarming systems. A behavior-based control algorithm consists of a set of modules called behaviors. These behaviors enable a robot to react to changes in its surrounding. For example, while a robot is approaching a desired target using go-to-goal behavior, it senses the presence of another robot on the way. To avoid a potential collision, it switches its current behavior to avoid the obstacle until the path to the target becomes clear again [31]. The design and tuning of these behaviors are done manually as a trial-and-error process until the desired collective behavior is obtained. This section will list go over several behaviour-based implementations of searching that are actually decentral and thus reside in the field of swarm robotics. It is important to note that not all of these have been tested in the real world.

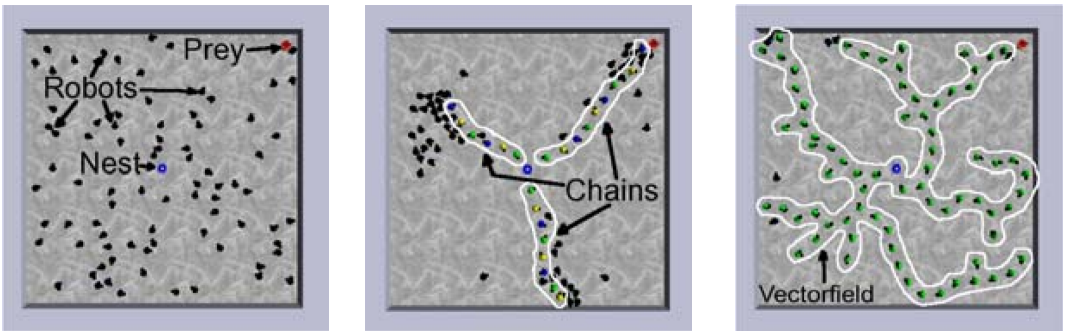
Often robots' physical bodies are used as markers to mimic pheromone trails [14]. In this thesis, such a robot is referred to as an *anchoring robot*, or simply an *anchor*. An anchoring robot is generally stationary. It can be sensed by other robots and may in some cases even send out signals to mimic the strength of that anchoring robot's virtual pheromone [32]. Werger and Matarić [33] used this concept and proved that foraging behaviour was possible even for very simple robots that only relied on physical contact-range sensing. The robots in this study were able to form a physical chain from the nest by zig-zagging along the existing chain and then backing into the end of the chain. All robots in the chain are in physical contact to one another and can send simple messages up and down the chain by tapping with the use of microswitches and break-beams. The robot at the end of the chain can decide to leave the chain, which allows the chain to be broken down dynamically. Prey is searched by the non-stationary robots in the swarm by making circular excursions near the existing chain. Together, from this swarm of simple robots following just a set of simple rules a collective foraging behaviour successfully emerged. Although the implementation sometimes led to unrecoverable errors which indicate there was room for work on robustness, this study refuted the assumed need for complicated sensors, positioning systems, or processing to achieve global position-dependent problems.

Payton et al [34] proposed an approach where robots first disperse in the environment using a gas expansion model: through robot-to-robot distance dependent attraction and repulsion, the robots in the swarm emulate gas particles filling a vacuum. When the robots are uniformly spread throughout the environment, they form a computing grid. A robot that observes a prey sends out a message to all nearby robots reachable by line-of-sight using a set of radially-oriented directional infrared receivers and transmitters. The message contains a hopcount. All receiving robots locally store the direction the message came from, decrement the hopcount and rebroadcast the message in all directions. Robots that receive multiple messages with different hopcounts select the highest hopcount. Together, the robots now form a

graph pointing towards the prey, that can potentially be used for all kinds of purposes, including foraging. Payton et al's approach leans on the connectivity between nodes revealing information about the topology of traversable paths. For this reason, line-of-sight communication between the robots is required, to ensure the path between two robots that perceive each other is physically traversable.
The approach of Payton et al requires no foreknowledge about the environment and no explicit knowledge about the robots' locations. It is quick and effective in environments small enough that the swarm can fill the entire space using gas expansion, but falls short when the space cannot be covered entirely. The main issue, however, is that even if a target was able to be found and the robot that finds the target is connected to the nest through several hops in the swarm, there would be no way to ensure that a physically traversable (see section 2.1.7) path exists along all of these hops. In this work a physically traversable path is defined as a path that can be traversed by a robot, using simple obstacle avoidance behaviour, successfully with reasonable confidence. Since Payton et al. employ line-of-sight communication, they assume that when a communication link exists between robots, a physically traversable path exists, which does not always hold in the real world. A case in point is when a cliff or rough terrain is separating the two robots that cannot be crossed.

Nouyan et al [35] focused on getting a robotic swarm to form a path between two locations in a bounded arena with the constraint that the robots' visual capacities do not allow them to perceive the two locations at the same time. A real robot named the *S-bot* was used for real world verification of the different swarming behaviours. This robot with a diameter of 12cm features a LED Ring with 8 RGB LED's and a vga-camera pointed at a spherical mirror that gives the bots an omnidirectional view. These allow the S-bots to communicate with each other fairly accurately with others up to 30cm away. The bots move at approximately 13cm/s and can make turns in place. The general idea of the searching behaviours is that the robots build up chains using their own bodies by moving from the nest along existing chains and extending the end of a chain with probability P_{in}. When they join a chain, they use their LED Ring to emit signals that inform of the directionality of the chain. Chains are also sometimes broken down: s-bots at the end of a chain have a chance P_{out} each step to leave the chain. Together this results in a behaviour where chains are continuously being built up and broken down. Four different chaining strategies were studied (see figure 2.2):

1. Static chains: The robots simply form (and break down) the chains described above.
2. Aligning chains: same as static chains, but here the s-bots align themselves with their closest two neighbours if their angle is smaller than 120 degrees. This creates relatively straight chains.
3. Moving chains: in this scenario, the s-bot located at the end of a chain will move perpendicular to the chain. Because the others in the chain respond by aligning, this results in the chain rotating around the nest.
4. Vectorfield: in this scenario, instead of chains, the s-bots form a tree-like structure with many branches. This causes the s-bots to be more evenly spread out throughout the area. Instead of using different LED colors to indicate the direction towards the nest, the vectorfield implementation has the s-bot "point" a vector in the direction towards the nest using its LEDs.



(a) (b) (c)

Figure 2.2: Simulation snapshots from the initial situation **(a)**, a typical outcome when employing one of the chain strategies **(b)** and the vectorfield strategy **(c)**. 80 S-bots are indicated by small black circles. The task is to form a path between the blue nest in the centre of the area and the red prey in the top right corner. In this case, there are no obstacles. Upon completion of the task, there will be a path between the prey and the nest. From [35], reprinted with permission.

Several tests were performed for each of the strategies, including scalability tests (a varying number of s-bots), difficulty tests (a varying distance from nest to prey) and obstacle tests (various environments with different obstacles). In these tests, generally the success rate was best when parameters P_{in} and P_{out} were in proximity of the line $P_{in} = P_{out}$. The aligning and moving chains strategies outperformed the static chains strategy. In environments without obstacles, they also outperform the vectorfield strategy for groups of up to 40 s-bots, but after that the vectorfield strategy outperforms. Also, for environments with many obstacles, the vectorfield strategy was the best performer.
The results suggest that for most swarming robotics purposes, the vectorfield strategy would be most desirable. When a robot finds the target, it has physically travelled from the nest to the target along a chain of anchoring robots, meaning that a physically traversable path exists. However, here the range in which a robot can sense the target (up to 90 cm away) is similar to (actually even larger than) the distance from which a robot can sense/communicate with each other (up to 60 cm away), this is not true in many real world applications (e.g., Ultra Wideband communication range is far higher). If the target to find is a buried mine, which can only be sensed very locally with a metal detector at the bottom of the robots, then there exists a large disparity between the robots' communication range and the target sensing range.

Obute et al. introduced RepAtt [36], using repulsion and attraction signals similar to Payton et al.'s gas expansion [34] to guide robots to advantageous areas. Robots employ a random walk strategy to search unexplored areas to find targets. The robots in this work were equipped with a downward facing camera to detect targets within visual range. The robots are able to find targets even though the target sensing range is smaller than the inter-robot communication range. When a robot finds a target, it acquires it and then enters the homing state to travel in a direct line to the nest, which is enabled by the nest's homing signal. RepAtt works even when using a realistic and far from ideal communication model, but RepAtt does not find nor demonstrate a physically traversable path and does not consider the case where the homing state fails at bringing a robot back to the nest when an obstacle is in the way.

	Werger et al. [33]	Payton et al. [34]	RepAtt [36]	Nouyan et al. [35]	CSP (This work)
Non-line-of-sight compatible	✗	✗	✗	✓	✓
Finds traversable path	✓	✗	✗	✓	✓
Short target sensing range	✓	✗	✓	✗	✓
Smart searcher distribution	✗	✓	✓	✗	✓
Decentral nest selection	✗	✗	✗	✗	✓
Resistant against node failures	✗	✓	✓	✓	✓
Resistant against nest failure	✗	✗	✗	✗	✗
Risk of network fracturing	H	M	M	L	L

Table 2.1: Feature comparison between CSP and related swarm search algorithms. H = High, M = Moderate, L = Low

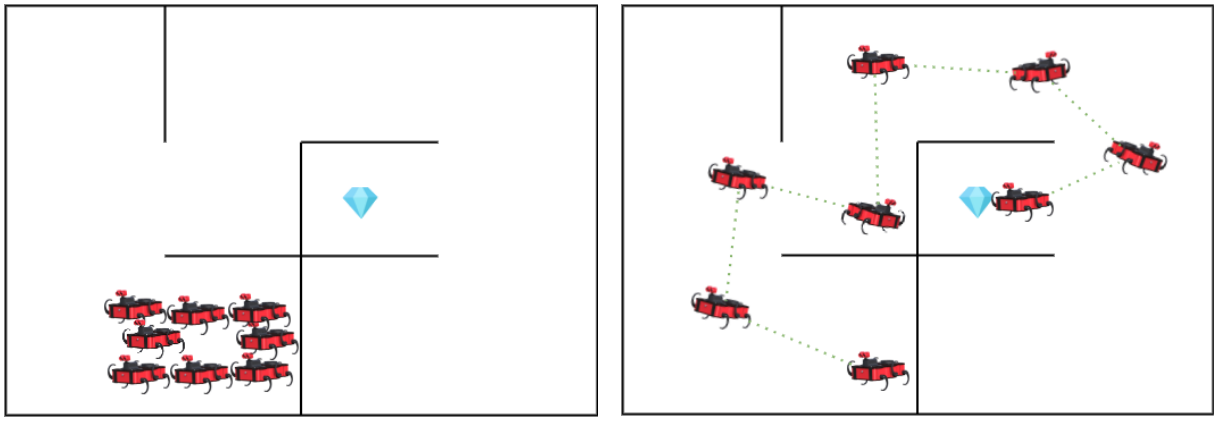
A feature comparison between the searching algorithms mentioned in this section and CSP can be found in Table.2.1.

2.4 Research question

The question this work aims to answer is:

How to make a swarm of Zebro robots search for a target in an unknown environment, and then have them form a path between the starting point and the found target using their own bodies?

Figure 2.3 visualizes the problem. This specific implementation of search behaviour could be used in *Search and Resque*-esque situations. One example would be that a swarm of Zebros is released in a calamity site with the mission to find persons. Once a person has been found, the path that the Zebros form with their bodies can be followed by a rescue worker to find the person in danger.



(a) (b)

Figure 2.3: (a) A room with walls and obstacles. The swarm's objective is to find a path to the target (diamond). (b) The desired final state: a singular path consisting of the robots' bodies between the starting point of the swarm and the found target.

Chapter 3

The Zebro Project

This thesis is part of the Zebro project, carried out by the Zebro team at the Delft University of Technology. This chapter will go over the history of how the Zebro came to be in section 3.1, after which it will focus on the Swarm Zebro, which is the robot that forms the inspiration of this thesis, in section 3.2.

3.1 Origins

The founding of the Zebro team was a result of a combination of factors. In the early 2000s, Boston Dynamics was developing a rough terrain scout robot named RHEX (Figure 3.1) for DARPA [37]. The focus was on making it very robust to be viable in military context, which would make production of the robot very expensive. Although this project was discontinued, it was picked up by several universities across the United States. One notable project is the Kod*lab RHex [38] from the University of Pennsylvania, which redesigned the original RHEX to be more modular and cost-effective, although the project has not seen cost-effective series production. Gabriel Lopes was involved in this project and later became an assistant professor at the TU Delft in 2010. He brought along his idea of animal-like walking robots and built a six-legged walking robot, which was named Zebro (see Figure 3.2).

When the TU Delft decided to perform research in the upcoming field of swarming robotics, the Zebro was picked as the favoured platform to test swarming on, because its insect-like design suited the the swarming robotics field's tendency to take inspiration from insects. So, the Zebro team was founded, which was initially a group of students supervised by Gabriel Lopes with the goal of improving the Zebro. Their efforts resulted in the *Zebro Light* (see Figure 3.3), which not only cut the original Zebro's weight by over 50%, but was also capable of performing a cockroach-like walking gait. With the completion of this prototype, the goal of the Zebro team was stated:
To design, plan and build self-deploying, fault-tolerant, inexpensive and extremely miniaturized robust autonomous roving robots to cooperate in swarms, capable of functioning on a wide spectrum of topology and environment that can quickly provide continuous desired information with the help of distributed sensor systems and carry and support payloads suitable for a wide range of missions.

From then on, multiple initiatives have been started by the Zebro team. This work focuses on the De-

Figure 3.1: The RHEX 1.1. Source: https://www.rhex.web.tr/

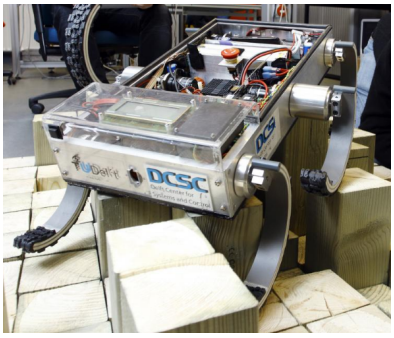


Figure 3.2: The original Zebro. Source: https://www.delta.tudelft.nl/article/lab-robotics

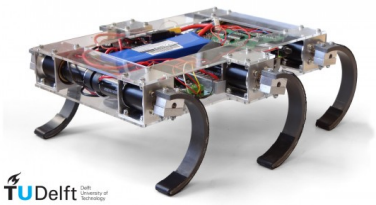


Figure 3.3: The Zebro light. Source: http://robotsquare.com/2013/07/17/zebro-light/

ciZebro. Its specifications will be given in section 3.2. However, some other noteworthy Zebro models are:

1. **The KiloZebro**
   As the name implies, the KiloZebro (see Figure 3.4) is a six-legged robot like the DeciZebro, except it is very large. Because of the heavy build of this Zebro, the motors that power the six legs are quite a bit more powerful than those on the DeciZebro. A big kill switch was placed on the front to ensure this Zebro could be stopped if it was about to collide with anything in its path after it was turned on.

2. **The PicoZebro**
   On the exact opposite part of the spectrum, there is the PicoZebro (see Figure 3.5): a six-legged robot like the DeciZebro and the KiloZebro, but much smaller. Its small size and cheap manufacturing costs make it ideal for demonstration purposes, although it misses functionality to be useful in most real-life situations.

3. **The Lunar Zebro**
   The Lunar Zebro (see Figure 3.6) is an ongoing project by the Lunar Zebro team to put a swarm of Zebro's on the moon. Because the regular DeciZebro would not be able to withstand the environmental hazards of the moon such as radiation, vacuum and lunar dust, the Lunar Zebro has been designed to take all these environmental conditions into consideration. The Lunar Zebro is planned to be launched to the moon no earlier than 2025.

3.2 The DeciZebro

The Swarm Zebro is intended to become a robotic platform that can autonomously traverse rough terrain and be used to carry out a wide variety of tasks in a decentral swarm. To achieve this goal, all iterations of

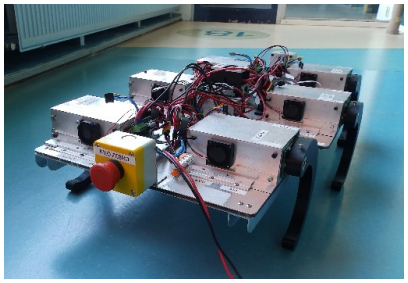


Figure 3.4: The KiloZebro

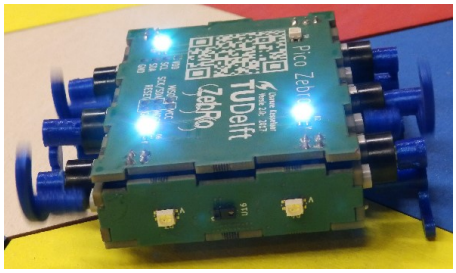


Figure 3.5: The PicoZebro

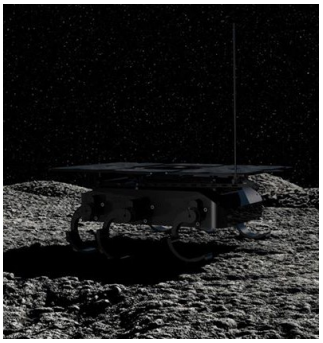


Figure 3.6: The Lunar Zebro

the design of the Swarm Zebro have focused on minimizing the production cost of the robot and making steps toward making sure the robot can be produced in series. The DeciZebro is the most recent iteration of the Swarm Zebro, designed by Mattijs Otten in 2017 [39] (see Figure 3.7). It is a modular design that allows for mass-production. Since this thesis focuses on the DeciZebro model, whenever the term Zebro is used in this thesis, it is referring to the DeciZebro. An aerial view of the Zebro can be seen in Figure 3.8. This section gives an overview of the Zebro's important components.

Figure 3.7: First generation DeciZebro

3.2.1 Modularity

One of the most prominent features of the Zebro design is the extent of its modularity. The philosophy behind the Zebro design is to allow for easy extensions and replacements on the robot in the future. Each of the six legs of the Zebro has its own PCB that takes care of the actuation of its respective leg. All of these leg PCB's are connected through a I^2C [40] bus to a Raspberry Pi Zero, the "brain" of the Zebro.

Any new components can be added onto the Zebro and connected through the same I^2C connection, which will make it easy to adapt the Zebro's hardware to any application's needs.

The modular legs are a good example that show that the modularity is not only prevalent in the hardware, but also in the software that has been developed so far. There is a very clear distinction in which piece of software is located for what purpose, and even on which chip that software runs. On the main computer in the Zebro, the Raspberry Pi Zero, there is (among others) the software that is referred to as the TLC (Top Level Controller). The TLC reads out data from any connected components and makes all the high level decisions, after which commands are sent to all the components in the Zebro. Most of the software that handles the components is located in or near the components themselves, such as in the case of the legs. Any code that is not strictly in the TLC, but that is still located on the Raspberry Pi Zero (for example various background tasks) are seperated in the software's architecture. The Zebro project strives to keep this modularity in place wherever possible in extensions and alterations to keep the Zebro robotic platform as agile as possible.

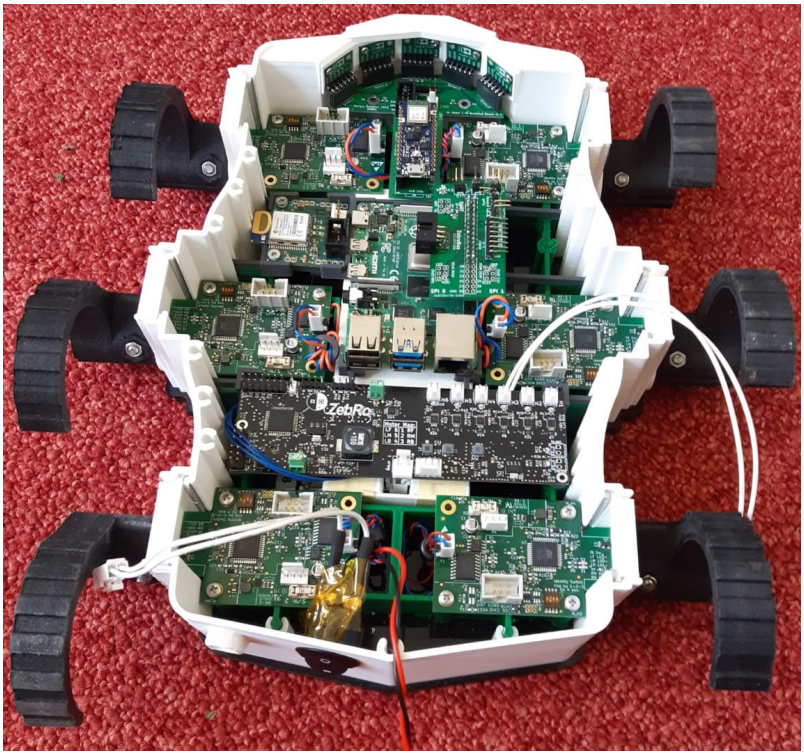


Figure 3.8: Aerial view of the insides of a Zebro

3.2.2 Six-legged design

To enable movement, the Zebro robot has six leg-shaped extremities, each connected to a servomotor. The main reason for choosing legs over wheels is that legs allow for easier traversal of rugged terrain. Where wheels can easily get stuck in grass, behind ledges and in other common outdoor situations, the six legs allow the Zebro to step onto – and over – obstacles and uneven patches in the terrain with relative ease.
The biggest downside of the six legs in the design is that it increases the complexity of moving around. Wheels can simply be turned forwards or backwards to move, but if you would try to apply the same tactic to the six legs, the robot would flop around, constantly crashing into the ground and not moving efficiently at all. That is why the Zebro needs proper coordination of the legs to ensure the body maintains its balance while moving. This legged locomotion has been researched and implemented by William Suriana [41]. The resulting implementation uses a trajectory-following SMPL (Switching Max-Plus Linear) system to allow the Zebro to move forwards, backwards and turn in both directions by alternating the rotation of the legs of the Zebro with specific timings so that there are always enough legs on the ground that the Zebro will not fall over. Using the SMPL system, the Zebro turns in a similar fashion as a differential-drive mobile robot, by altering the lift-off and touchdown times of the legs in such a manner that there is a difference between the angular velocity of right-sided and left-sided legs. With this difference in linear velocity between the right and left side, the Zebro, like wheeled robots, will produce turning movement.

3.2.3 Ranging sensors

The Zebro is equipped with five VL53L1X Time-of-Flight laser-ranging sensors [42] by default. All of them are located in the front of the Zebro as seen in Figure 3.9: one of them faces directly forward, two of them are placed at a slight horizontal angle (both directions) and the last two are placed at an even larger horizontal angle (both directions). Each of these sensors can fairly accurately measure distances of up to 400 cm, regardless of the color or reflectance of the object in front of it. These sensors can allow the Zebro to avoid collisions by turning away from obstacles in front of it. All five sensors are connected to the Raspberry Pi Zero using the I^2C connection, so that the TLC can make use of the sensors' readings to make decisions in the Zebro's behaviour.

Figure 3.9: Front view of the ranging sensors in the Zebro

3.2.4 Localisation and Communication

As of starting this project, there is a wireless communication and localisation system in development for the Zebro. It is based on UWB (ultra wideband). This chip, as the name implies, allows multiple individuals in the swarm to send messages to each other, but also to localize themselves relatively to each other. Because this system is still in development, in this thesis it is assumed that at some point the Zebro will be able to measure their own relative position to another Zebro within communication range. Although UWB provides a communication range of up to 25m, the actual quality of communication and localisation for the Zebro is unknown as of now.

3.2.5 Battery and Power Management

The Zebro has a battery pack consisting of four 18650 batteries, connected in series. Together these batteries deliver a voltage between 12V and 16.8V. There is a BMS (Battery Management System) that measures the temperature and the cell voltage of each battery to balance the output of each battery. There is also a PMS (Power Management System) in place to perform power conversion and furthermore, each motor is fused to as a security measure. The battery pack can currently be charged through a laptop charging port located on the backside of the Zebro, although multiple charging options such as through a solar panel or through a wireless charging pad are planned for a future version of the Zebro.

Chapter 4

The Concurrent Searching and Pathfinding algorithm

This chapter describes how CSP works and gives the rationale behind the decisions that led to its design. First, the problem will be analyzed and some constraints and goals will be set that the solution should adhere to in section 4.1.

4.1 Problem analysis

As described in section 2.4, the research question is:
How to make a swarm of Zebro robots search for a target in an unknown environment, and then have them form a path between the starting point and the found target using their own bodies?

The desired behaviour is a variation of foraging behaviour. This behaviour has been described in detail in section 2.2.2. It is also a problem that has been examined in the field of swarming robotics many times. In section 2.3 several studies have been listed that implemented variations of the foraging behaviour in real robotics. Those solutions can function as a good example to aid in designing similar behaviour for a Zebro-like robot. For this reason, frequently used terms in those studies will also be introduced here: the starting location of the swarm will be called the *nest* and the target location will be called the *target*.

4.1.1 Targets

There are some targets to be set that are deemed important for the implementation of the algorithm. This section will go over all the targets that the algorithm should adhere to.

Fault tolerance
Fault tolerance is a priority. The final algorithm should be resistant to different types of errors and faults. When robotic swarms walk around in the real world, all kinds of unexpected things can happen. The kinds of errors that the final algorithm should at least be able to cope with are:

- **Dying nodes**
  In the real world, a robot could simply stop working all together at some point because of a soft-

ware crash, a hardware problem, because the robot fell from a height, or simply because the battery died. Whatever the cause, one or multiple robots that stop working altogether at some point should not cause the whole swarm to be permanently in disorder. Even if almost the whole swarm dies, the remaining robots should be able to still function as a group.

- **Getting stuck** Although the Zebro's legs are designed to be able to step over small obstacles and traverse rugged terrain (see section 3.2.2), it cannot be assumed that the Zebro, or any other kind of robot for that matter, will never get stuck. A robot might fall into a pit that it cannot get out of on its own or get stuck in grass that is too high to move through, or one or multiple legs might even break off which prevents locomotion altogether. Even on optimal terrain, a hardware or software failure could occur that prevents a robot from walking any longer. How it happens is irrelevant; fact is that there can be situations in which one or multiple robots in the swarm are still functional, but cannot move anymore.

- **Being moved**
  Although ideally the robots would move only if they intend to move, external factors can also cause a robot to move in the world. A robot can get picked up by a human being, blown away, pushed by another robot, et cetera. If this will interfere with the robot's positional tracking and the robot thinks it is in a different position than that it actually is while or after being moved, this can potentially be a short term problem. However, once the external factors stop moving the robot, it should be able to reorient itself after some time and the swarm should be able to keep on functioning.

- **Communication errors**
  Section 3.2.4 mentioned that the Zebro will be equipped with a localization and communication module. In any asynchrous distributed system it cannot be assumed that data arrives correctly to other parties, or if it even arrives at all. Robotic swarms are especialy prone to communication errors because the topology of the network is constantly changing as the robots move around and as the environment changes. This is why the implementation of the algorithm should be very tolerant to transmission errors. Stormont [43] described that it is important to provide redundant communication. No single message should be *required* to be successfully received by the intended receiver for the whole swarm to continue operations. Even if many messages in the swarm get lost or get delivered with bit errors, the swarm should eventually be able to resume operation of the algorithm.

This is a varied group of possible faults that can occur that the swarm should be able to endure while executing the search algorithm. If at any point the algorithm cannot be continued any longer because of a critical error somewhere in the swarm, the remaining robots that are in each other's vicinity should at least be able to regroup and restart or choose to abandon the searching operations.

There are also some possible problems for the successful execution of the algorithm, that will *not* be considered in the scope of this thesis. These problems are:

- **Byzantine failures**
  A byzantine failure is a type of failure where the rest of the swarm can not be sure if the node failed. This kind of failure could occur in the swarm if the software in a robot malfunctions or if one of the robots runs different code from the rest of the swarm (for example an old version of the same algorithm, or malicious code uploaded by an adversary). This type of error is difficult to solve in distributed systems. Solutions to this problem usually increase the complexity of the whole algorithm and only works when a certain percentage of nodes in the network can be considered trustworthy. Because of the complexity byzantine failures are considered out of scope for this thesis.

- **Intruders**
  A similar situation to a byzantine failure. If another machine is communicating within communication range of a robot through the same communication medium, then the robots in the swarm might think that the other machine is also an individual in the swarm and treat is as part of the swarm. This could happen on accident, but also on purpose by malicious parties that could build a machine for the sole purpose of interfering with swarm. Because of the complex nature of this problem, this thesis will also not focus on solving the problem of dealing with intruders.

- **Duplicate identities**
  This thesis assumes that all robots have some form of identification that is unique to them. This can be used in the algorithm to direct messages towards a certain robot or to recognize which of the other robots is sending a message. This thesis will not focus on what will happen if two robots happen to have the same identifier.

- **Battery management**
  In nature, the participants in a swarm need to make sure that they acquire energy (through eating), because otherwise they will die and not be helpful to the swarm anymore. Robots can also run out of energy and they will need to recharge if their battery gets low. Battery management will not be considered in the scope of this thesis.

Decentrality
Because the swarm must be as fault-tolerant as possible, there should not be a single point of authority. Any robot should be able to stop completely or partially working at any point, and the swarm must be able to continue. All robots will get exactly the same code, and none of the robots will know the state of the entire swarm at any given point. Robots may have different roles at different moments in the algorithm, but these roles should be dynamically appointed or assumed, and those roles should not grant them authority over other robots or allow them to know the global state of the swarm. Pradhan et al. [44] showed that signalling is generally a more efficient means of communication for a foraging swarm than using stigmergy. One of the targets in designing the searching behaviour solution is to use communication only in cases where it has a substantial advantage over using local observation.

Scalability
Because this is a swarm algorithm, the algorithm should be scalable to a large amount of robots. There should be no mechanisms in the algorithm that impose limitations on the amount of robots in the swarm.

Modularity
Although this thesis focuses on finding a searching behaviour solution that suits the Zebro robots' capabilities, the aim is to provide a general solution for searching under the constraints that are mentioned in the introduction. Ideally, the resulting searching algorithm should be suitable to many types of robots and for many situations. Moreover, this thesis only tests the algorithm working in the ARGoS [2] simulator on a virtual robot. Writing the code to be modular would is helpful for portability so that the pieces of code that are hardware-dependent can easily be swapped out for code that works on the targeted hardware.

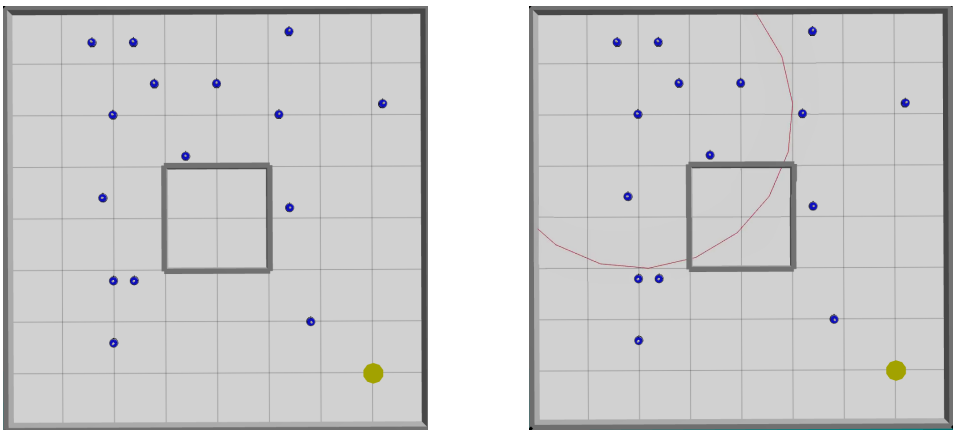
4.1.2 Hardware constraints

Although the algorithm is implemented in a virtual simulator only, the virtual robots are assumed to have similar capabilities as the Zebro robot. That means that the robots are assumed to be practically blind, as the Zebro has no incorporated camera system, instead merely relying on ranging sensors in the front side to enable obstacle avoidance. Since the Zebro is planned to have an ultra wide-band based localisation and communication module that allows Zebro's to communicate and determine their relative position to each other, the virtual robots in this work are assumed to be capable of doing this too, within a communication range of 3m. Crucially, this also means that the robots in this work are not equipped with a line-of-sight communication or sensing system, which means that two robots that sense each other can not assume that the space seperating them is physically traversable. The virtual robots in this work are 17cm in diameter and are able to move forwards at a speed of 10cm/s and make turns using a differential drive system. The robots are assumed to be able to detect the target within a range of 25cm, which is a factor of twelve smaller than the robots' communication range of 3m.

4.2 Algorithm concept

In a general setting, the swarm will be located in an unknown area with no knowledge of the location of the target. When the robots in the swarm are being "turned on", they are idle. The swarm must decentrally start the searching mission. The desired end state is that the robots collectively form a singular path between the starting point of the mission (the nest) and the target. To achieve this, CSP has been designed, which works in three phases:

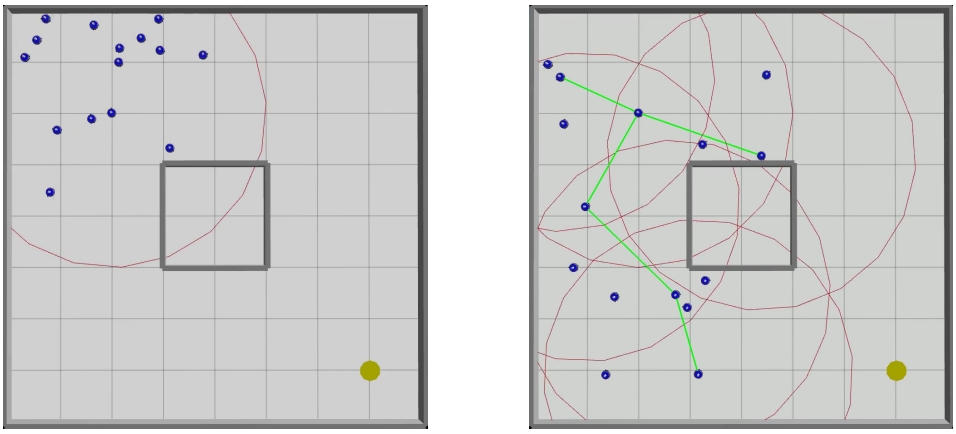
Phase 1: Nest selection
The swarm starts in an idle state where every bot is doing nothing (Figure 4.1a). A robot may spontaneously decide to initiate the searching behaviour. When this happens, this robot tries to become the *nest* through a leader election process. During the leader election process, several types of messages will be propagated through the swarm, which will make the swarm agree on who the nest is and also make any idle robot join the searching behaviour, taking on the searcher role. All the recruited searcher robots will first travel towards the nest to ensure they are within communication range from the nest (Figure 4.1b).



(a) (b)

Figure 4.1: (a) Starting position in an arena with an obstacle in the middle. The yellow dot is the target. (b) A leader has been elected to take on the role of the nest. The red circle shows the communication range of the nest (3.0m).

Phase 2: Searching While anchors stand still as a mark, all searchers walk through the environment in random directions, while always attempting to stay within communication range from an anchor (Figure 4.2a). Occasionally, a searcher turns into an anchor and other times an anchor can turn back into a searcher (with the exception of the nest). This causes the backbone to be built up and get constantly rearranged (Figure 4.2b). The searchers will cover more and more ground until, finally, one of the searchers finds the target.

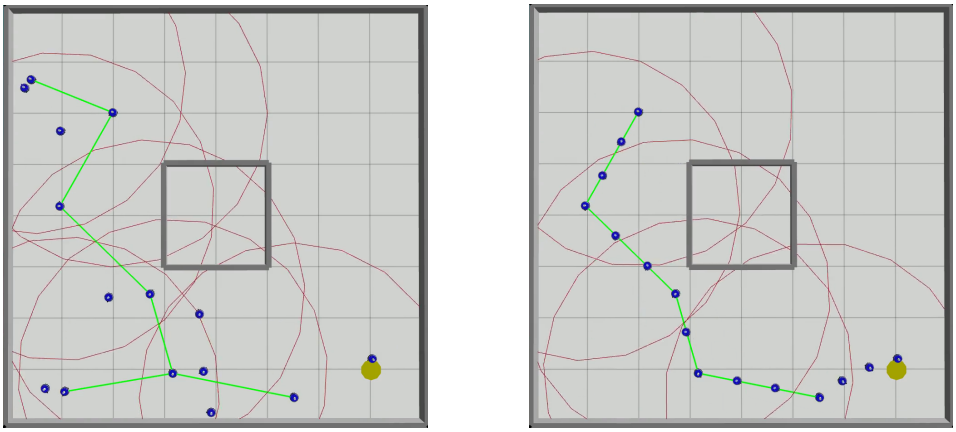


(a) (b)

Figure 4.2: (a) All robots have moved within communication range from the nest. (b) A network of anchors is being built up. The green lines connecting the anchors visualize the backbone.

Phase 3: Path formation When a robot finds the target (Figure 4.3a), it broadcasts a message indicat-

ing the target has been found. A procedure starts in the swarm that will eventually cause all bots that are *not* a part of the chain of anchors from the nest to the target to drop their searcher or anchor role. They will spread themselves evenly on the path from the nest, through all the anchors, to the target. The result is that all the bots in the swarm are forming a path that starts at the nest and ends at the target, around all the obstacles in the environment (Figure 4.3b).



(a) (b)

Figure 4.3: (a) A robot finds the target. (b) The robots have formed a singular path between the nest and the target.

4.3 Implementation

This section describes the specifics and considerations of CSP in more detail.

4.3.1 Nest selection

The searching task should be a behaviour that can be started by the swarm at will. In this work, when the swarm is turned on, all robots are idle. The swarm should be able to dynamically switch from this idling behaviour to searching behaviour, for example as a response to an impulse one or more of the robots receive. When the searching task gets initiated by one or more robots in the swarm, all idle robots should join in on the searching task and the entire swarm should agree on which robot should take on the nest role.

This problem is a variation on what is called the leader election problem in the field of distributed systems: all participants in the distributed system need to agree on *one* leader. Although this is a prominent problem that has many solutions, most of these solutions are for networks with static topologies. However, a robotic swarm is more complex, because they form a mobile ad-hoc network (MANET): a network of wirelessly connected mobile components that does not rely on any pre-existing infrastructure. The topology of the network of robots is highly dynamic. The robots can walk, so the possible connections between robots can change at any point. There is not even any guarantee that the network will be

connected at all times. Also, one or multiple robots can also simply stop working entirely during any operation. On top of that, the highly dynamic nature of the swarm's network is not the only difficulty; there is also no guarantee for a message to be received by the other robots, even if they are within communication range. There have been several proposed solutions to the leader election problem in mobile ad-hoc networks.

Derhab & Badache [45] state that although the classic property of the leader election problem in a distributed system with a fixed number of nodes is that *eventually there is a unique leader*, this can not be guaranteed with the characteristics of an ad-hoc network.

Although there are several viable solutions to the leader election problem, not all of them are applicable to CSP. Some of the solutions are quite robust, but are not suitable because the pose extra requirements on the hardware of the swarm. An example is the algorithm proposed by Derhab & Badache [45], which requires all nodes to have synchronized clocks. Malpani et al. [46] propose an algorithm based on TORA (Temporally Ordered Routing Algorithm) that ensures that eventually all network partitions have a unique leader. It aims detect partitions in the network, and to elect a new leader when the old leader of a network partition becomes unreachable. However, in the case of searching behaviour this does not make sense: the "leader" is just the nest. If the swarm were to re-elect another random robot in the swarm halfway through the searching task, then the new nest would then be in a completely different location from the original nest. Even if the rest of the execution of the searching task would be successful, the result would be undesirable: the formed path would now be between the target and the new nest, which is in an arbitrary position in the environment, instead of the original starting position. That is why the nest will be treated as a special case in this thesis: the nest will be required to keep functioning properly for the algorithm to execute successfully. In the case that part of the swarm loses connection to the nest, instead of electing a new one, an effort should be made to reconnect to the old one, by for example moving closer to it. In the case of an actual failure of the nest, the searching behaviour of the swarm should terminate. In the future works section (section 6.2) several alternatives to deal with failures regarding the nest will be discussed.

A desirable solution for this specific leader election problem should be simple and self-stabilizing, make sure that eventually there is not more than one leader (nest) in connected parts of the swarm network. To make implementation simple and meet specifically just these needs, the implementation of leader election is a relatively simple algorithm inspired by the asynchrous leader election algorithm for complete networks by Afek & Gafni [47], but applied to connected networks rather than complete networks, among other changes.

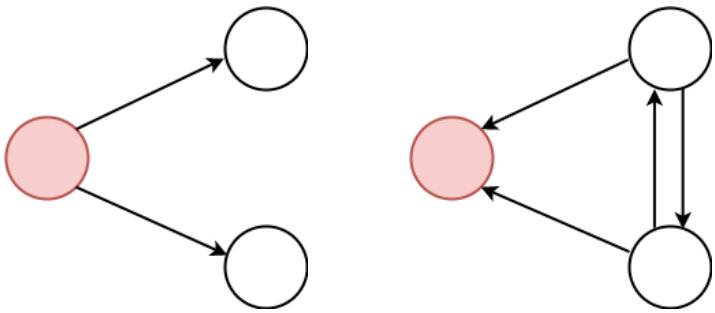
Afek and Gafni's election for leader election in asynchronous it taken as a starting point. To make it work in a robotic swarm, three key parts have to be changed because of the nature of the swarm network. First of all, the swarm network is not a complete network, but a connected network (unconnected individuals are not considered to be part of the swarm). Secondly, the swarm's network topology will be highly dynamic instead of static, as the robots will be moving around. Lastly, there is no guarantee messages will arrive even if two individuals are within communication range in the real world, due to transmission errors.

Transitioning from a complete network to a connected network

Whereas in a complete network every individual in the network can be reached directly, in a connected network it can only be guaranteed that other individuals can be reached, but this might be through multiple hops. The main idea behind making the algorithm work in a connected network is to make every individual in the network function as a message relay. When you abstract this functionality, the algorithm still works the same in principle, but direct messages are "simulated" in the swarm. The way this simulation of direct messages works is by substituting every message with a broadcast. There are two types of messages, unaddressed messages (aimed at every individual in the swarm) and addressed messages. When an unaddressed message is sent, the message is broadcast with a *hopcount* parameter, which is initially at 0. Every individual in the swarm that receives this message will broadcast the message again, with the *hopcount* increased by 1.

Addressed messages are only sent to nodes that the address (ID) is known of. This means at one point, an unaddressed message has been received from that node. This means, a *hopcount* was part of that message. This *hopcount* is the amount of hops it took for the fastest message to get from the sender to the receiver (not neccesarily the shortest *hopcount*!). Any addressed message will therefore initially be broadcast with *hops_left* at this initial value. Every node that receives the message, but is not the intended receiver, will re-broadcast the message with the *hop_count* decreased by 1. If *hop_count* reaches zero, the message will not be propagated further through the swarm.

This approach comes with the following problem:



(a) (b)

Figure 4.4: (a) The leader (red) broadcasts an unaddressed message, which two nodes receive. (b) The two nodes that received the message broadcast the message again, which everybody receives again. This is the start of an infinite loop.

An infinite loop occurs (see Figure 4.4). Since every node that receives the message re-broadcasts it, the message can end up at nodes that already received that message, and even at the node that initially sent the message. To solve this problem, every message sent by a node will contain an ID that increases every message. Every node keeps a list of messages it has received from each node, and ignores messages that are already present in that list.
Another problem that arises from making this switch to the connected network, is that there is no way for an election candidate to know if it has captured all nodes in the swarm. In Afek and Gafni's ori-

ginal algorithm, every node "knows" all the direct connections it has to other nodes in the network and therefore knows the amount of nodes it has to capture. This does not hold true in the swarm's network because there are is no predefined network and there are no physical links. This means a fundamental change in the algorithm has to be employed: instead of waiting until all individuals in the swarm have been captured before turning from a candidate into a leader role, every candidate will assume to be the leader until they are captured by a bigger leader. This means that there can be multiple active leaders in the swarm at the same time, however, as long as the network remains connected, eventually only one leader will remain.

Transitioning from a static network to a dynamic network and allowing for communication errors

The problems that arise from having a dynamic network instead of a static network, and having to account for communication errors are closely related problems, in that they can both result in messages not being delivered. The main idea behind accommodating for this problem is to continuously re-send certain messages at an interval. Given that the algorithm runs infinitely, this is the only way to guarantee that messages will be delivered.

For example, the capture broadcast will be re-sent every t seconds with a new ID. This means that the entire capture broadcast will be propagated through the swarm again. This ensures that nodes that failed to receive the message previously, will now have a new chance to receive the message and join the search mission. Also, new robots that joined the network (for example because they just got turned on, or because they suddenly walked into communication range of the swarm) will now also be reached by the capture broadcast and join the search mission.
The same principle applies to the capture ack (capture acknowledgement) messages. To ensure the leader knows it has captured a node, that node has to re-send the capture ack at an interval.

Leader election algorithm as a task starter

In related works (see section 2.3) that propose distributed searching algorithms, the "leader" or "nest" is often considered a separate entity from the swarm which is referred to as the nest. Therefore, there is no need for any type of leader election algorithm, and robots in the swarm immediately start performing the searching task upon being turned on. In the real world, it would be desirable to have the swarm be able to shift between different tasks. The leader election algorithm described in this section allows for that to happen. Within the capture broadcasts, a mission parameter can be passed. All nodes captured by a leader will know the value of this mission parameter, and execute the associated behaviour. In this thesis, there are only two types of behaviour, namely idling (this is the default behaviour of the robot upon being turned on) and the searching behaviour. However, this makes it easy to add different types of behaviour depending on the applications of the swarm.

Loss of leader problem

This thesis states that any individual in the swarm should be allowed to fail or crash, without it affecting the swarm. In this thesis, an exception is made for the leader (nest) node, simply because it does not make sense to continue the execution of the algorithm if the nest fails. The nest marks the start of the search path. If the starting point of the searching mission is lost, the task itself, which is defined as finding a path to the target between the starting point, has become unattainable. Theoretically, solutions to this specific problem could be engineered, with another individual navigating to where the original nest was standing and taking its place. However, because of the difficulty of this problem, it is considered out of the scope for this thesis. Instead, a loss of the starting point is considered to render the searching mission a failure by default in this thesis.

4.3.2 Searching

In the previous section, the nest selection phase has been described. One robot has now taken on the nest role, while all other robots have taken on the role of searcher. This section will describe the workings of the searching phase of CSP.

Building up the backbone

In Nouyan et al. [35] Vectorfield, chains of anchoring robots are dynamically built up. In this work the collection of anchors is referred to as the *backbone*. All searchers start moving from nest along the backbone. At any point, a searcher may become a new anchor themselves, expanding the backbone by starting a new branch or making a current branch longer. As this backbone grows, the ground that can be searched increases as the size. Anchors can also decide to stop being an anchor, in which case they will return to the nest and resume as a searcher. This behaviour causes the backbone to dynamically evolve over time.

As mentioned in section 2.3, the work of Nouyan et al. [35] works because the distance from which a robot can detect the target is greater than the distance from which robots can sense each other. In our work, this is not the case. The robots in this work have an increased communication range of 3m, which theoretically allows us to cover much more area with our backbone because the anchors can be spread further apart. However, the target can only be sensed when a robot is within 25cm. This means that it will not suffice for searchers to merely travel from anchor to anchor; the area around anchors has to be more intricately searched. To adhere to the key principles of simplicity in swarm robotics, random walks are used. However, having the searchers do random walks in the entire area that is within communication range from the backbone would be inefficient. New and under-explored areas should be prioritized in the search. This can be achieved through a system where every searcher is designated to one anchor only. Searchers perform a random walk until they near the edge of the communication range (0.1m margin), then they will move back to the anchor in a straight line (while employing simple obstacle avoidance behavior), before starting to random walk once again. When a single searcher performs this type of searcher around its designated anchor, Figure 4.5 shows how much ground around the anchor will be covered as a function of time. It is assumed that every additional searcher speeds this process up proportionally.

An anchor can keep track of how well the area around it has been explored simply by keeping track of how many searchers it has appointed to it over time. This amount of searchers will be called n. The

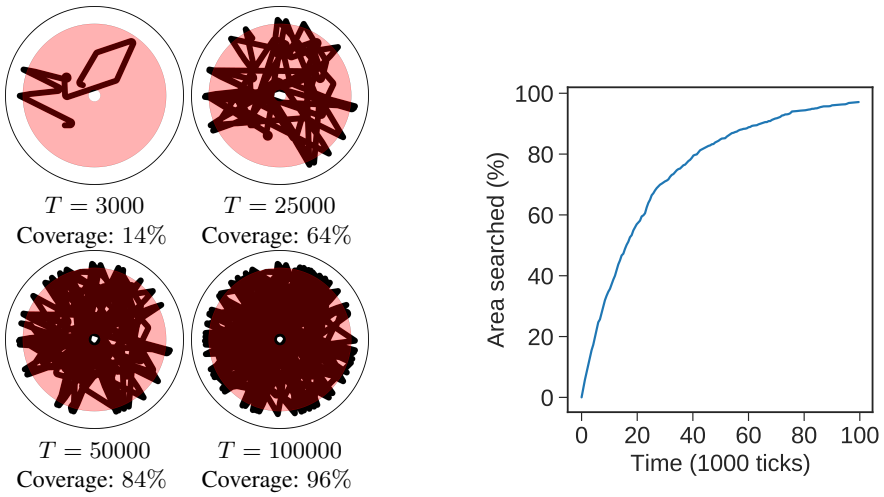


Figure 4.5: How quickly area around an anchor is searched using our implementation of the random walk strategy: (a) Visual representation. The outer black circle shows the communication range of the anchor (3m), and the red circle indicates the measured search area (radius = 2.4m). (b) In graph form. (N=4)

nest keeps track of a variable *groundCovered*. Every 10 ticks (at 10 ticks per second), anchors perform the following simple operation,

$$groundCovered = groundCovered + n$$

The higher the *groundCovered* gets, the better the area around it has been explored. Anchors use this information to make a smart decision about when to relocate a searcher. The higher *groundCovered* gets, the fewer searchers the anchor will keep to itself. It will get rid of surplus searchers by either recruiting a new child anchor or appointing its searchers to exist child anchors (see Algorithm 1). This reallocation of searchers shifts the workforce to areas where it is needed more. All of this is done without any single robot in the swarm having a map of the environment or keeping track of the actual area that has been searched. Instead, it is based on stigmergy and the result of this simple set of rules, making it a form of swarm intelligence.

The process of recruiting a new anchor

An anchor starts the recruitment process by broadcasting a *RECRUIT_NEW_ANCHOR* message. Any searcher that is appointed to this anchor receives this message and joins the recruitment process. When this happens, the searcher stops moving and broadcasts a *PING_ALL_ANCHORS* message. Any anchor that receives this message replies to this message. The searcher is only eligible to become a new anchor if:

1. The closest anchor to this searcher is this searcher's appointed anchor.
2. The searcher is at least 1m away from the appointed anchor.

Algorithm 1: Donating searchers

```
1  groundCovered ← 0                                   ▷ Ground searched around this anchor
2  donationRate                                        ▷ Constant that decides how soon bots will be donated
3  searchers                                           ▷ This anchor's searchers
4  childAnchors                                        ▷ Child anchors assigned by this anchor
5  failedAttempts ← 0                                  ▷ The number of times in a row that creating a child anchor failed
6  satisfied ← false   ▷ Turns to true when this anchor has four child anchors. Never turns back to false, even if child
     anchors disband
7  isNest                                              ▷ True if this anchor is also the nest
8  chooseAnchorOdds   ▷ Chance between 0 and 100 of creating a new anchor over donating searchers when both are
     possible

9        ▷ Code executed at every tick by anchors
10 searchers ← RemoveInactiveSearchers(searchers)   ▷ Forget about all searchers from which a heartbeat
     message has not been detected recently
11 groundCovered += Size(searchers)
12 botsToKeep ← Ceil((10.0 - donationRate * groundCovered)/2) * 2
13 botsAvailableToDonate ← Size(searchers) − botsToKeep
14 if botsAvailableToDonate > 0 then
15 |  DonateSearchers(botsAvailableToDonate)
16 end
17 Function DonateSearchers(amount):
18 |  canCreateAnchor ← (Size(searchers) > 2 and ((!satisfied and failedAttempts <
   |    2) or (isNest and Size(childAnchors) < 4)))
19 |  canRelocateSearchers ← (Size(childAnchors) > 0)
20 |  while amount > 0 do
21 |  |  if canCreateAnchor and (!canRelocateSearchers or Rand(0, 100) < chooseAnchorOdds) then
22 |  |  |  RecruitNewAnchor()                          ▷ Start recruiting a new anchor
23 |  |  |  canCreateAnchor ← false
24 |  |  else if canRelocateSearchers then
25 |  |  |  searcher ← Random(searchers) anchor ← Random(childAnchors)
   |  |  |    RelocateSearcher(searcher, anchor)       ▷ Send a message to searcher to become a searcher of
   |  |  |    anchor
26 |  |  amount − −
27 |  end
```

These constraints are put in place to maximize the reach of the network tree of the anchors. By making sure the new anchors are not positioned too close to other anchors, any new anchor is guaranteed to be in a currently uninhabited area. Figure 4.6 demonstrates the effect of these constraints in an example setting.

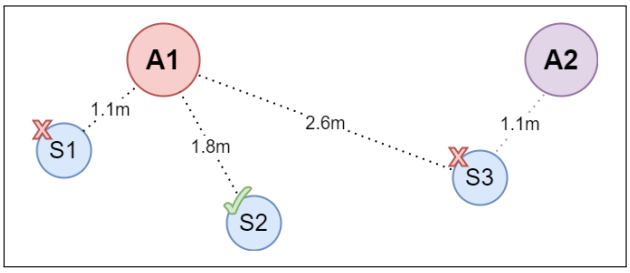


Figure 4.6: Anchor A1 is recruiting a new anchor and has three searchers S1, S2 and S3 within communication range. Searcher S3 is ineligible because it is closer to another anchor (A2) than it is to A1. Between the remaining searchers S1 and S2, searcher S2 wins because it is the farthest away from A1.

There is a predefined time window of 465 ticks in which the anchor waits to get messages back from its searchers. Once this time window has passed, the anchor will decide which searcher, if any, to make a new anchor. If no searcher is eligible, the recruitment process fails and a cool-down period starts before the anchor can retry recruiting a new anchor. During this cool-down period, searchers will move around as usual, which means they might end up in a spot where it's possible to become a new anchor in the next recruitment attempt.

Reappointing a searcher

Anchors remain stationary. Therefore, if a new anchor is recruited, it will need the help of searchers to scout the area around it. A newly appointed anchor will always immediately receive one searcher from its upstream anchor upon being appointed. Furthermore, as already described, it can receive more searchers from its parent at any time.

The process of reappointing a searcher is as follows: the anchor sends a RELOCATE_SEARCHER message to one of its searchers (picked randomly), containing the ID as well as the location of the chosen downstream anchor. When the chosen searcher receives this message, it will immediately internally acknowledge the new anchor as its anchor. The searcher will continue its searcher like usual. Searchers always try to stay within communication range of their anchor, but upon being re-appointed to a new anchor they might now suddenly be outside of the communication range. The searcher will then immediately start heading towards its anchor's location to get back within communication range. In other words, searchers will naturally head towards their new anchor without the need for a new explicit rule instructing them to do so. Since this new anchor always originated from the same anchor, this new anchor should generally be physically reachable by the searcher.

Breaking down the backbone

As CSP is being performed, the backbone will dynamically be built up. However, the backbone can also be broken down. There are four situations in which an anchor can disband (stop being an anchor and revert back to being a searcher).

1. The anchor has no downstream anchors, $groundCovered \geq 80$, and the anchor has failed at least twice in a row at creating a new anchor. This situation seems indicative of a dead end: the target cannot be found here and it is hard or impossible to deploy new downstream anchors here to continue the search.

2. The target has been found in the swarm by a searcher and this anchor is not a part of the chain of anchors from the nest to the searcher that found the target.

3. The connection to the upstream anchor is lost.

When an anchor disbands, it will send broadcast a DISBAND message, that will be received by all of its searchers and child anchors. The anchor, as well as its searchers and child anchors will move upstream to an anchor that has not yet disbanded.

Stabilization messages

Many things can go wrong in the swarm. A list of the different failures that this thesis wants to be able to deal with can be found in section 4.1.1. To get around those issues, state messages are repeated on an interval. These are such messages:

- *SHARE_POSITION*: This message is broadcast by anchors to notify their searchers, as well as their upstream anchor and downstream anchors of their own position. It is also an indicator that they are still alive.

- *HEARTBEAT*: This message is send by searchers to their anchor. It serves to let anchors know that this searcher is alive and appointed to them. These messages enable anchors can keep track of their searchers.

4.3.3 Path formation

When a searcher finds the target, the mission of the swarm has succeeded, but there are still some things to be done. The complex structure of bots has to be broken down and rearranged in such a way that only a singular path of bots remains between the starting point and the target. The way the backbone is constructed and the fact that searchers stay within communication range from their anchors ensures that the swarm is connected. There exists a path in the backbone from the nest to the searcher that found the target. All the individual connections in that path should generally be physically traversable.
The searcher that finds the target sends out a *TARGET_FOUND* message to its anchor. This anchor will propagate the message upstream in the backbone until it reaches the nest. Each anchor adds the amount of searchers it has to a parameter $totalSearchers$ in the message, and adds the distance towards the node

they received the message from in parameter $pathLength$. When this message reaches the nest, the nest finally adds its own searchers to $totalSearchers$ and the last distance to $pathLength$. Every anchor (including the nest) now knows the path length that is left from itself to the found target is $distanceLeft$. The nest now initializes its parameter $pathPointsLeft = totalSearchers$. It calculates how many robots should turn into path points between itself and the next anchor ($distanceToNextNode$ away) towards the target using the following formula:

$$myPathPoints = (pathpointsLeft + 1)/distanceLeft \times distanceToNextNode$$

The nest then subtracts $myPathPoints$ from $pathPointsLeft$ and sends this value to the first child anchor on the path towards the target in a *PATHINFO* message. That anchor performs the formula above with the received updated $pathpointsLeft$ variable, with its own values for $distanceLeft$ and $distanceToNextNode$, subtracts its own resulting $myPathPoints$ from $pathPointsLeft$ and passes the *PATHINFO* message with this updated $pathPoinsLeft$ variable on to the next anchor. This continues until the final anchor along the path to the target has been reached. Every anchor now instructs $myPathPoints$ amount of searchers to position themselves uniformly in the distance between itself and the next node. Any surplus searchers are sent either upstream or downstream, depending on where there is a lack of searchers to form the desired path.
Any anchor that was not a part of the final path from the nest to the target, will receive the *PATHINFO* message, conclude they are not part of the resulting path, and immediately disband. This will cause all anchors and searchers that are not part of the final path to return back to the nest as searchers. They will not become path points yet. However, the robot that finds the target sends out the *TARGET_FOUND* message on an interval, so in the next iteration, the nest will include them in the path formation process. The final result as seen from above will then be one single chain consisting of all the robots' bodies, signifying a physically traversable path from the nest to the target (see Figure 4.3b). The whole process happening on an interval stabilizes the path formation process.

Chapter 5

Results

CSP is a complex multi-faceted algorithm. Although swarm systems are implemented at the individual level, collective swarm behavior has to be evaluated. CSP was implemented in C++ and tthe performance of the swarm was tested in several situations in the ARGoS simulator [2]. In these experiments, timing starts when one of the robots has started the leader election process, making itself the nest. The simulation runs at 10 ticks per second. This section presents our results.

5.1 Swarm Size versus Search Time

To determine the impact of the swarm size on the performance of the swarm, an experiment was performed where the number of nodes in the swarm was variable, while the environment was unchanged in each run. Arena 1 (Figure 5.1) was chosen as the environment because it is a neutral environment. Though it looks relatively simplistic with a hurdle in the middle of the map, the robots need to find that there is an obstacles and also find a work around; further, in an unbounded environment, the searching can be unbounded. Figure 5.2 shows that increasing the number of bots in the swarm decreases the search time. However, as the number of bots in the swarm keeps increasing, the time saving will be diminished.

5.2 Packet Loss Effect on the Performance of the Swarm

Efforts have been made to make the swarm resistant to communication failures. Not only do important messages get repeatedly broadcast on an interval, nodes in the swarm also proactively propagate messages in the swarm, which often causes multiple possible paths for messages to be transmitted through the swarm.
A test has been performed to assess the impact of the quality of the communication links on the performance of the swarm. This test was also performed in Arena 1. For several different swarm sizes, simulations were run where the chance that messages fail to deliver has been assessed in the range of 0% up to 90%. The results of this test can be seen in Figure 5.3. The results show that the higher the amount of bots in the swarm is, the less the swarm is affected by dropped messages. The results also show that even for a swarm size of just 10 bots, the swarm is not heavily affected by dropped messages even as the probability reached 0.5. It is only after that point that the execution time starts to worsen very rapidly.

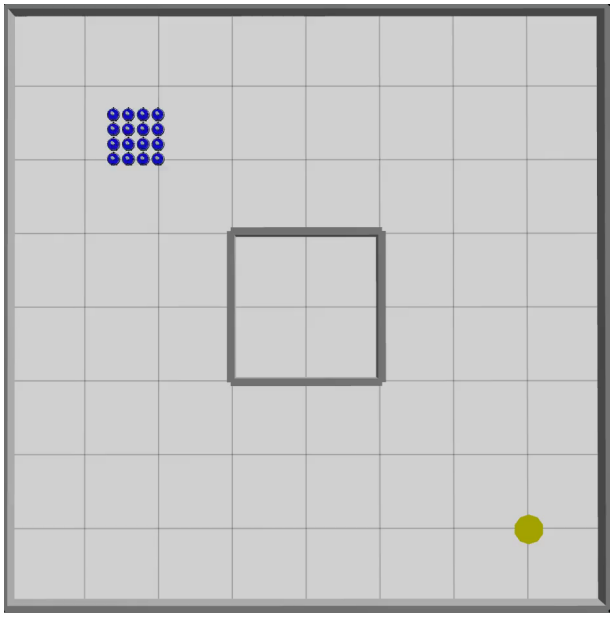


Figure 5.1: Arena 1. The swarm starts in the top left corner and the yellow circle in the bottom right corner is the target. This arena features one obstacle in the center of the map.

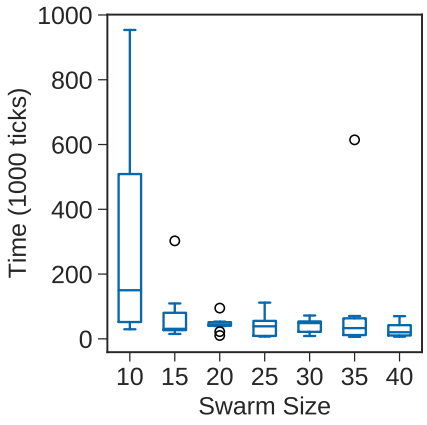


Figure 5.2: Swarm size versus execution time of the algorithm (N = 10 for each bar).

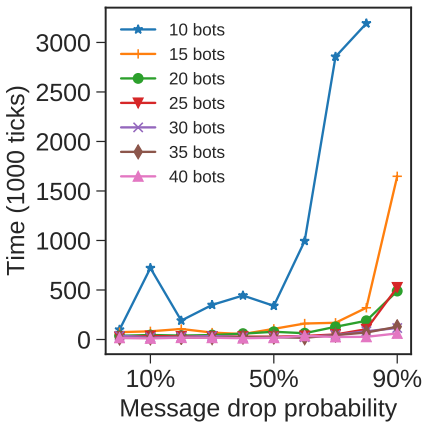


Figure 5.3: The effect of packet loss on the performance of the swarm for different swarm sizes (N = 10 for each data point).

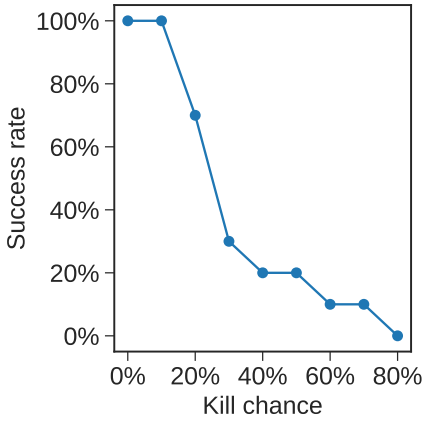


Figure 5.4: The impact of node increasing kill chances on the success rate of the mission (N = 10 for each data point)

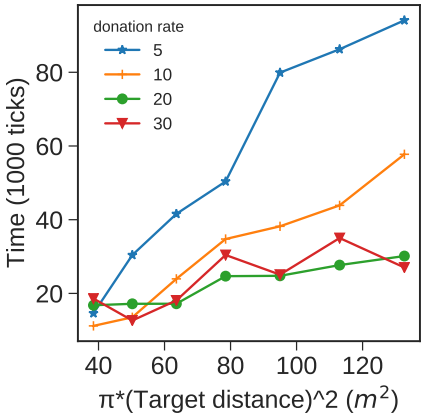


Figure 5.5: The impact of *DonationRate* on the efficiency of the search at different target distances.

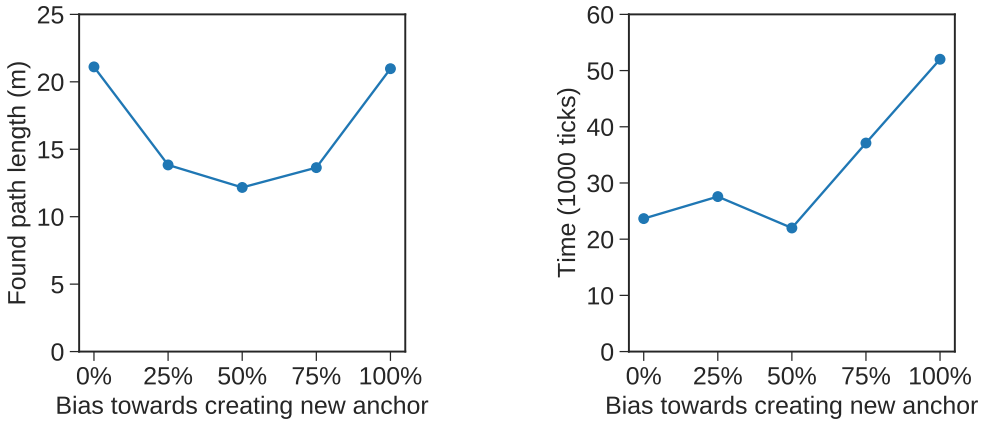
5.3 The Effect of hardware failures on the Performance of the Swarm

To test CSP's resistance against hardware failure, a test has been performed that introduces a chance that a robot dies at every tick. This kill chance does *not* apply to the nest. Whether the search mission succeeds before all robots in the swarm die has been assessed for a range of kill chances for a swarm of 25 bots in Arena 1. If the kill chance is 0, the median search time the swarm takes in this scenario is 27370.5 ticks (based on 10 measurements). The range of kill chances has been chosen such that the chance that any given node is alive at 27370.5 ticks is between 0% and 80%. Increasing the kill chance not only affects the swarm directly, it also makes the search mission take longer, which increases the chance that all bots in the swarm have died before the target can be found. Therefore, even with perfect algorithmic resistance against dying bots in the swarm, a high enough kill chance should always affect the success rate of the swarm. The results of this test can be seen in Figure 5.4.

5.4 Donation Rate and target distance

An anchor's surplus searchers will be used to create new child anchors or will be donated to existing child anchors to expand the backbone. The rate at which this happens is defined by a variable *DonationRate*. The formula that calculates the maximum amount of searchers an anchor is allowed to have at any point is as, $10 - DonationRate * groundCovered$.

An experiment has been set up to test the impact of this variable $DonationRate$ against the performance of the swarm for different target distances in an open arena. The results of this test can be seen in Figure 5.5. As the target distance increases, the area that theoretically should be searched increases as well (πr^2 where r is the target distance). The results show that for farther target distances, higher donation rates start significantly outperforming lower donation rates, showing that it is beneficial to quickly expand the backbone when the target is far away.



(a) (b)

Figure 5.6: The impact of the anchor's donation tactic on (a) the length of the path that is finally formed. (b) the execution time of the algorithm.

5.5 The impact of branching bias on path length and execution time

As mentioned in the previous subsection, whenever an anchor has surplus searchers, it will either assign new child anchors or assign the surplus searchers to the existing child anchors. Of course, when an anchor has no child anchors yet, the only choice is to create a new child anchor. However, if there already is a child anchor and the maximum of 3 child anchors is not reached yet, both options are possible. The anchor then decides with a certain probability P which of these options is chosen for any surplus searcher. A higher chance P means a higher chance of creating a new anchor, which results in more branches in the anchor network; a lower chance P means searchers are more often donated to existing child anchors which results in long chains in the anchor network. An experiment was performed to assess the effect of this probability P on the length of the found path's length (Figure 5.6a) and on the average execution time (Figure 5.6b). The experiments were performed in the open arena, with a target distance of 5 m and a donation rate of 30. The results show that a donation rate of 50% resulted in the shortest found path lengths and that the swarm was quicker at finding the target when P was 50% or lower.

Chapter 6

Discussion

6.1 Conclusions

CSP is a novel algorithm that enables a robotic swarm to find a target in an unknown environment and construct a physically traversable path to it. CSP achieves this even when the target sensing range is significantly smaller than the robot-to-robot communication range, and does so without assuming that a communication link between two robots being present means that the path between those robots is physically traversable. An approach similar to Nouyan et al's vectorfield [35] was implemented to effectively search unknown areas. Because the targets in this work could only be detected very locally by the robots, CSP put an emphasis on making the robots extensively search the area to increase the chance of a robot bumping into the target. To accomplish this, searchers are assigned to an anchor, around which they will randomly search the area, while staying within communication range. The searchers can be made to travel up or down in the backbone at the request of an anchor, but never between branches. This is because there is no guarantee that there is a physically traversable path to a perceived anchor in a different branch, as our swarm does not rely on line-of-sight communication. To improve the performance of the swarm, the anchors actively manage and relocate the searchers in strategic ways. As the efficiency of searching the area around an anchor decreases with time, it is not surprising that preferring to expand the network quickly is generally more efficient than letting the searchers search every nook and cranny around a new anchor before moving on. The results also suggest that in most cases, a balance between creating long chains and creating multiple branches is ideal.

The swarm does not start out with searching behaviour upon being turned on, but instead is idle. The swarm is able to collectively switch to a task using a leader election algorithm. This is useful in real-world cases where swarms should be able to perform different kinds of tasks and dynamically switch between them without requiring the swarm to be turned off first or reprogrammed. CSP allows any idle robot to spontaneously decide to start a searching task in the swarm. When this happens, that robot assumes the role of the nest and convinces the rest of the swarm to join the searching mission. Multiple robots could decide at the same time to start the searching behaviour and become the nest, but the leader election algorithm makes sure that as long as the swarming network is connected and stabilizes, there will eventually be only one nest.

CSP was shown to not only be scalable in swarm size but also get more resistant to communication failures as the swarm size increases, withstanding up to 70% message drop without detrimentally affecting the performance for swarms consisting of at least 15 robots. CSP is also robust against hardware failures – the search success rate stays at 100% for a node kill chance of 10%, and first reaches 0% for a kill chance of over 70%. Results also show that a clean 50/50 balance between extending existing chains and creating new branches in the backbone optimizes the length of the path that was found as well as the execution time.

Although CSP provides a solution to searching and path formation that works under the given constraints, it is important to note that major restricting decisions have been made in the design to allow for this. For real-world problems, the more known about the specific environment and the swarms' available hardware and capabilities, the better the behavior of the swarm can be tweaked to perform optimally. In many cases, our method may not yield the most optimal result. Instead, our method can serve as a solid base that can function even in conditions where many existing solutions are infeasible.

6.2 Future Work

Any robot in the swarm is allowed to fail at any moment without major consequences for the swarm, with the exception of the robot that functions as the nest. When the nest crashes, the starting point of the searching mission has become lost. Further work is needed to implement a solution to this problem. Since all searchers and child anchors of the nest know the location the nest should be in, a solution could be developed where one of them moves to the location where the nest was and takes on the role of the new nest.

CSP has been designed as part of the Zebro Project. To answer the research question (section 2.4), The Zebro's limitations were carefully considered with the ultimate goal of implementing behaviour that demonstates the swarm co-operating to achieve the concrete problem of finding an object in an unknown environment. CSP has the swarm form a path with the robots' physical bodies to make it clear even to laymen that the robots are collaborating and found this path as a collective. However, although CSP has been tested in virtual environment, CSP has not yet been successfully implemented on the actual Zebro robot that it is aimed at. This would be the next logical step. As a part of this project, CSP's code has already been ported to the Zebro's Raspberry Pi Zero processor and made to interface with the Zebro's hardware. There were several issues with the Zebro swarm were encountered that made real world verification hard, such as there only being four completely functioning Zebro robots in the Zebro lab during this thesis, unreliability of the hardware and limited battery life. However, the largest problem that made actual real-world verification on the Zebro robot impossible within the scope of this thesis was the lack localisation functionality in the localisation and communications module. This work assumes that robots can relatively localize themselves to another robot within communication range, but the module responsible for this could not produce accurate results. In order to make CSP work on the Zebro, either this module needs to be fixed, replaced, or an external positioning system should allow the Zebro to simulate the required functionality.

Bibliography

[1] T. Wang, H. Zhao, and Y. Shen, "An efficient single-anchor localization method using ultra-wide bandwidth systems," *Applied Sciences*, vol. 10, no. 1, p. 57, 2019.

[2] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, M. Birattari, L. M. Gambardella, and M. Dorigo, "ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems," *Swarm Intelligence*, vol. 6, no. 4, pp. 271–295, 2012.

[3] E. Şahin, "Swarm robotics: From sources of inspiration to domains of application," in *International workshop on swarm robotics*. Springer, 2004, pp. 10–20.

[4] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo, "Swarm robotics: a review from the swarm engineering perspective," *Swarm Intelligence*, vol. 7, no. 1, pp. 1–41, 2013.

[5] A. Czirók, H. E. Stanley, and T. Vicsek, "Spontaneously ordered motion of self-propelled particles," *Journal of Physics A: Mathematical and General*, vol. 30, no. 5, p. 1375, 1997.

[6] A. Czirók, M. Vicsek, and T. Vicsek, "Collective motion of organisms in three dimensions," *Physica A: Statistical Mechanics and its Applications*, vol. 264, no. 1-2, pp. 299–304, 1999.

[7] I. Aoki, "A simulation study on the schooling mechanism in fish." *The Japanese Society of Fisheries Science*, vol. 48, no. 8, pp. 1081–1088, 1982.

[8] C. W. Reynolds, *Flocks, herds and schools: A distributed behavioral model*. ACM, 1987, vol. 21, no. 4.

[9] A. Huth and C. Wissel, "The simulation of the movement of fish schools," *Journal of theoretical biology*, vol. 156, no. 3, pp. 365–385, 1992.

[10] I. D. Couzin, J. Krause, R. James, G. D. Ruxton, and N. R. Franks, "Collective memory and spatial sorting in animal groups," *Journal of theoretical biology*, vol. 218, no. 1, pp. 1–11, 2002.

[11] P. J. Slater, C. T. Snowdon, J. S. Rosenblatt, and M. Milinski, *Advances in the study of behavior*. Academic Press, 1997, vol. 26.

[12] E. Bonabeau, G. Theraulaz, J.-L. Deneubourg, S. Aron, and S. Camazine, "Self-organization in social insects," *Trends in ecology & evolution*, vol. 12, no. 5, pp. 188–193, 1997.

[13] D. H. Janzen, "Euglossine bees as long-distance pollinators of tropical plants," *Science*, vol. 171, no. 3967, pp. 203–205, 1971.

[14] M. Kohler and R. Wehner, "Idiosyncratic route-based memories in desert ants, melophorus bagoti: how do they interact with path-integration vectors?" *Neurobiology of learning and memory*, vol. 83, no. 1, pp. 1–12, 2005.

[15] M. Mangan and B. Webb, "Spontaneous formation of multiple routes in individual desert ants (cataglyphis velox)," *Behavioral Ecology*, vol. 23, no. 5, pp. 944–954, 2012.

[16] H. Cruse and R. Wehner, "No need for a cognitive map: decentralized memory for insect navigation," *PLoS computational biology*, vol. 7, no. 3, p. e1002009, 2011.

[17] M. Collett, L. Chittka, and T. S. Collett, "Spatial memory in insect navigation," *Current Biology*, vol. 23, no. 17, pp. R789–R800, 2013.

[18] B. Cartwright and T. S. Collett, "Landmark learning in bees," *Journal of comparative physiology*, vol. 151, no. 4, pp. 521–543, 1983.

[19] R. A. Harris, P. Graham, and T. S. Collett, "Visual cues for the retrieval of landmark memories by navigating wood ants," *Current biology*, vol. 17, no. 2, pp. 93–102, 2007.

[20] R. Möller, "Insect visual homing strategies in a robot with analog processing," *Biological Cybernetics*, vol. 83, no. 3, pp. 231–243, 2000.

[21] B. Cartwright and T. Collett, "Landmark maps for honeybees," *Biological cybernetics*, vol. 57, no. 1-2, pp. 85–93, 1987.

[22] J. Zeil, M. I. Hofmann, and J. S. Chahl, "Catchment areas of panoramic snapshots in outdoor scenes," *JOSA A*, vol. 20, no. 3, pp. 450–469, 2003.

[23] R. Wehner and M. Müller, "The significance of direct sunlight and polarized skylight in the ant's celestial system of navigation," *Proceedings of the National Academy of Sciences*, vol. 103, no. 33, pp. 12 575–12 579, 2006.

[24] M. Wittlinger, R. Wehner, and H. Wolf, "The ant odometer: stepping on stilts and stumps," *science*, vol. 312, no. 5782, pp. 1965–1967, 2006.

[25] F. C. Dyer and J. L. Gould, "Honey bee orientation: a backup system for cloudy days," *Science*, vol. 214, no. 4524, pp. 1041–1042, 1981.

[26] R. Menzel and U. Greggers, "The memory structure of navigation in honeybees," *Journal of Comparative Physiology A*, vol. 201, no. 6, pp. 547–561, 2015.

[27] D. J. Sumpter and M. Beekman, "From nonlinearity to optimality: pheromone trail foraging by ants," *Animal behaviour*, vol. 66, no. 2, pp. 273–280, 2003.

[28] L. Conradt and T. J. Roper, "Group decision-making in animals," *Nature*, vol. 421, no. 6919, p. 155, 2003.

[29] Technode, "Alibaba launches china's biggest robotic warehouse ahead of singles day," https://technode.com/2018/10/30/alibaba-new-robotic-warehouse/, 2018.

[30] Intel, "Intel breaks guinness world records title and lights up the sky at winter olympics with intel shooting star drones," https://newsroom.intel.com/wp-content/uploads/sites/11/2018/02/Intel-Olympics-Drone-Fact-Sheet.pdf, 2017.

[31] R. C. Arkin, R. C. Arkin *et al.*, *Behavior-based robotics*. MIT press, 1998.

[32] N. R. Hoff, A. Sagoff, R. J. Wood, and R. Nagpal, "Two foraging algorithms for robot swarms using only local communication," in *2010 IEEE International Conference on Robotics and Biomimetics*. IEEE, 2010, pp. 123–130.

[33] B. Werger and M. J. Mataric, "Robotic food chains: Externalization of state and program for minimal-agent foraging," *From Animals to Animats*, vol. 4, pp. 625–634, 1996.

[34] D. Payton, M. Daily, R. Estowski, M. Howard, and C. Lee, "Pheromone robotics," *Autonomous Robots*, vol. 11, no. 3, pp. 319–324, 2001.

[35] S. Nouyan, A. Campo, and M. Dorigo, "Path formation in a robot swarm," *Swarm Intelligence*, vol. 2, no. 1, pp. 1–23, 2008.

[36] S. O. Obute, P. Kilby, M. R. Dogar, and J. H. Boyle, "Repatt: Achieving swarm coordination through chemotaxis," in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2020, pp. 1307–1312.

[37] U. Saranli, M. Buehler, and D. E. Koditschek, "Rhex: A simple and highly mobile hexapod robot," *The International Journal of Robotics Research*, vol. 20, no. 7, pp. 616–631, 2001.

[38] KOD*LAB, "Rhex - kod*lab," https://kodlab.seas.upenn.edu/robots/rhex/, 2017.

[39] M. Otten, "Decizebro: the design of a modular bio-inspired robotic swarming platform," Master's thesis, 2017.

[40] NXP Semiconductors, *I^2C-bus specification and user manual*, NXP Semiconductors, 2014, rev. 6. Document Identifier: UM10204.

[41] W. Suriana, "Turning of a legged robot via a switching max-plus linear system," Master's thesis, 2017.

[42] ST, "A new generation, long distance ranging time-of-flight sensor based on st's flightsense™ technology," https://www.st.com/resource/en/datasheet/vl53l1x.pdf, 2018.

[43] D. P. Stormont, "Autonomous rescue robot swarms for first responders," in *CIHSPS 2005. Proceedings of the 2005 IEEE International Conference on Computational Intelligence for Homeland Security and Personal Safety, 2005.* IEEE, 2005, pp. 151–157.

[44] A. Pradhan, M. Boavida, and D. Fontanelli, "A comparative analysis of foraging strategies for swarm robotics using argos simulator," in *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 2020, pp. 30–35.

[45] A. Derhab and N. Badache, "A self-stabilizing leader election algorithm in highly dynamic ad hoc mobile networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 7, pp. 926–939, 2008.

[46] N. Malpani, J. L. Welch, and N. Vaidya, "Leader election algorithms for mobile ad hoc networks," in *Proceedings of the 4th international workshop on Discrete algorithms and methods for mobile computing and communications*. ACM, 2000, pp. 96–103.

[47] Y. Afek and E. Gafni, "Time and message bounds for election in synchronous and asynchronous complete networks," in *PODC*, vol. 85, 1985, pp. 186–195.