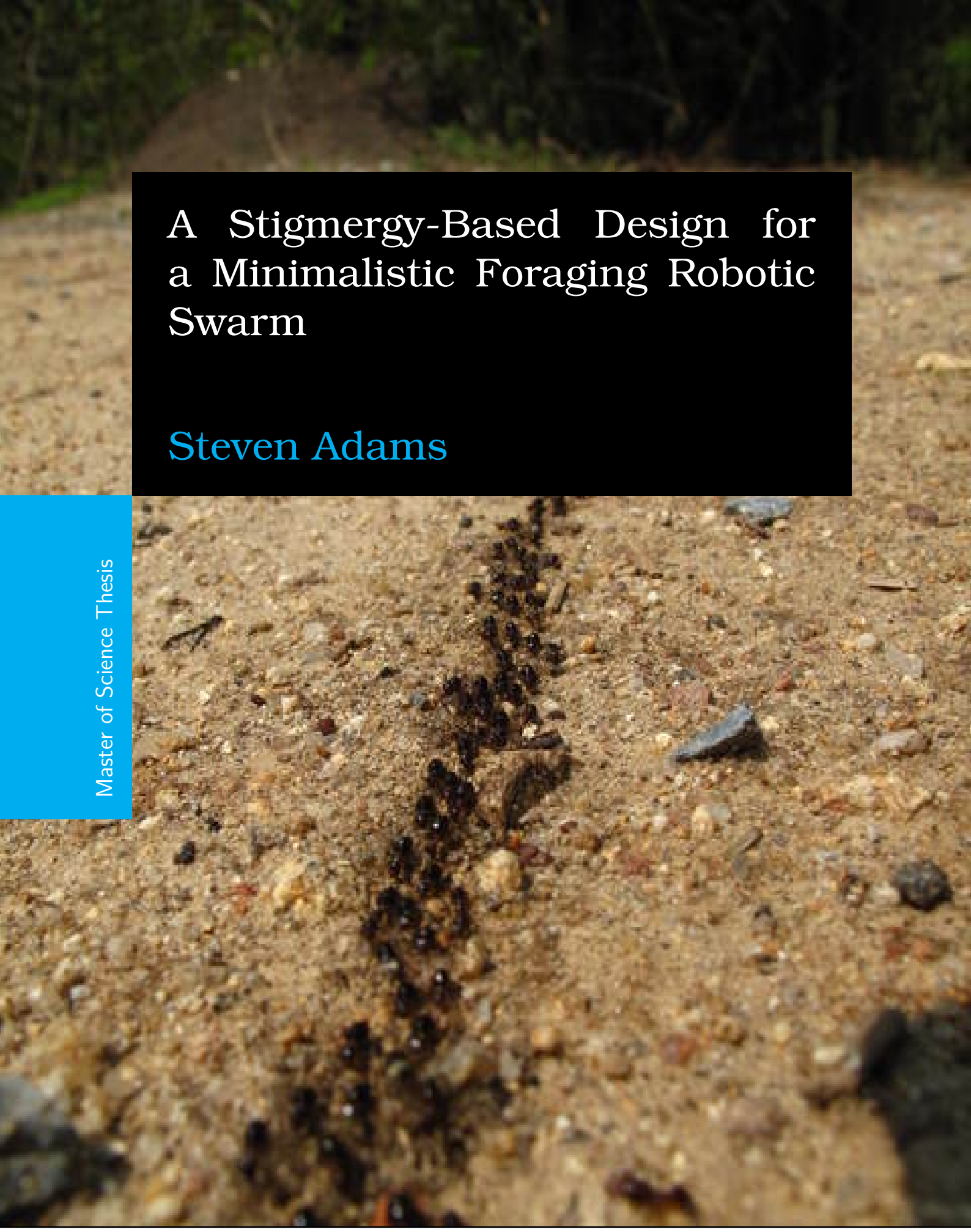


A Stigmergy-Based Design for a Minimalistic Foraging Robotic Swarm

Steven Adams

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

A Stigmergy-Based Design for a Minimalistic Foraging Robotic Swarm

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Steven Adams

June 29, 2021

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Copyright © Delft Center for Systems and Control (DCSC)
All rights reserved.

Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

A Stigmergy-Based Design for a Minimalistic Foraging Robotic Swarm

by

Steven Adams

in partial fulfillment of the requirements for the degree of

Master of Science Systems and Control

Dated: June 29, 2021

Supervisors:

Ir. D. Jarne Ornia

Dr. ir. M. Mazo Espinosa

Reader:

Dr. ir. J. Alonso Mora

Abstract

Over the last years advantages in autonomous agent systems and technology have created many potential applications for large numbers of collaborating robots in the field of surveillance, mapping, mining, farming and (space) exploration. The underlying principle that enables robots to collectively solve complex tasks is that of minimal interference: the basis of swarm robotics. In nature, swarms of insects use stigmergy, communication through environment marking, to connect individual and collective behavior. Many have tried to implement this stigmergic principle in swarm robotics, though it remains a challenge to implement stigmergy in robotic systems suited for real applications.

In this thesis, we present a biologically inspired minimalistic design for a foraging robotic swarm based on stigmergy. Our self-guiding swarm requires only very restricted robot capabilities: Robots do not require global or relative position measurements; the swarm is fully decentralized; and the robots need no infrastructure in place. Additionally, the system only requires one-hop communication over the robot network, we do not make any assumptions about the connectivity of the communication graph and the transmission of information and computation is scalable versus the number of agents. All this is realized by letting the agents in the swarm act as foragers or as guiding (beacons).

We analyse the characteristics and performance of our swarm by our own developed so called Particle Simulator and the realistic Webots Simulator using a swarm of Elisa-3 robots. We show how the swarm self-organizes to solve a foraging problem over an unknown environment and converges to trajectories around the shortest path. In addition, we study directions of future improvements of the swarm design regarding the minimisation of resources, optimality of the created paths and the convergence speed. At last, we derive formal results regarding the reachability, coverage time and hitting times of the swarm.

Contents

1 Introduction

1.1. Motivation and Concepts

Over the last decades the advancements of processing power, sensor accuracy and battery sizes have enabled the use of multi-agent techniques to solve complex robotic tasks. Within multi-agent control, swarm robotics is a growing field that emphasizes the cooperation and the collectivity of a robot group. Rather than equipping an individual robot with a control mechanism that enables it to solve a complex task on its own, individual robots are usually controlled by simple strategies, and complex behaviours are obtained at the colony level by exploiting the interactions among the robots, as well as between the robots and the environment. When designing swarm robotics control algorithms, complex strategies are generally avoided, instead principles such as locality of sensing and communication, homogeneity and distributeness, are followed. The main benefits that one seeks when pursuing a swarm robotics approach are scalability with the number of robots, robustness with respect to noisy conditions, and fault tolerance in case of individual failure. These characteristics can be observed in social insects such as ants, bees, or termites, which therefore often serve as source of inspiration. Over the years many biological inspired swarm robotic systems have been proposed, but the application of them in real-life is still sparse.

In this thesis we focus on the foraging problem, in which a number of agents start at a given point in space and must find a target in an unknown environment, while converging to cycle trajectories between the initialization and target region. The goal is to travel as efficiently as possible between the two regions. The foraging problem includes online path planning as well as exploration, both interesting problems when designing very large robotic systems. Moreover, the combination of exploration and exploitation is nowadays a point of interest for AI related techniques.

In particular the foraging problem has been addressed by robotic ant-inspired swarms that use indirect communication through some ”pheromone”. Many practical methods have been explored to implement pheromone based robotic navigation. However, often complexities explode when designing very large multi-robot swarms, or systems include implicit assumptions that in practice prevent them from being applied to large scenarios or real situations. Our goal in this thesis is to design a minimalistic swarm system capable of solving the foraging task in real-life.

1.2. Goals and Expectations

The main goal of this thesis is to design a minimalistic swarm system capable of solving the foraging problem by using a form of pheromone-inspired communication, with the following restrictions:

- Minimal assumptions on the robot capabilities. All agents are supposed to have equal characteristics (homogeneous system), and do not have knowledge of relative (or global) positions with respect to other agents, and only need an orientation measure (a compass).
- The system relies on one-hop communication only with limited range, and does not require direction or distance information on signals, nor line-of-sight connectivity.
- The system is fully distributed and needs no infrastructure in place.
- Does so with robustness versus communication events or single agent failures

The process of achieving our main goal is split into the following sub-goals:

- Orientate on the characteristics of swarm robotics and its general advantages over other robotic control approaches.
- Study the proposed methods to implement stigmergy in swarm robotics, and decide on which approach is best to use for the foraging problem.
- Study which swarm robotics solutions have been proposed for the foraging problem, classify the solutions and relate the implied robot capabilities to the characteristics of the classes of solutions.
- Based on the insights of literature, propose a minimalistic swarm design.
- Derive formal results on the expected behaviour of the designed swarm.
- Setup an experimental framework and derive performance metrics allowing to experimentally analyze the designed swarms behaviour.
- Experimentally analyze the parametric foraging performance of the designed swarm.
- Experimentally analyze the scalability and robustness of the designed swarm.
- Given the shortcomings of the designed swarm, propose possible extensions, adding complexity but potentially increasing the foraging performance.

2

Background and Problem Description

In this chapter, we orientate on the characteristics of swarm robotics and its general advantages over other robotic control approaches; investigate the proposed methods in literature to implement stigmergy in swarm robotics and decide on which approach is best suited for the foraging problem; and study which swarm robotic solutions have been proposed for the foraging problem and classify these solutions.

First we will introduce in Section 2.1 some interesting biological principles employed in swarm robotics. In Section 2.2, we will discuss the advantages of swarm robotics over classical robotics. In Section 2.3, we will discuss how stigmergy can be implemented in (swarm) robotics, and decide which approach we will take. In Section 2.4 and 2.5, we introduce the foraging problem and present several swarm robotic solutions proposed in literature to tackle the foraging problem, of which we explain the most promising in Section 2.6 in more detail. Since the foraging problem also includes efficient exploration of the environment, and the chosen direction of our potential design involves the challenge of distributing agents optimally over an environment, we will discuss in Section 2.7 some coverage and exploration control approaches. In Section 2.8, we discuss the connection between path planning algorithms and robotic foraging systems. At last in Sections 2.9 and 2.10 we present the preliminaries and define the problem description.

2.1. Self-Organisation, Stigmergy and Path Integration

The collaborative behavior of social insects and their robust performance when fully decentralized solving complex problems, while preserving adaptability and flexibility to changing circumstances, has inspired many engineers to create collaborative artificial systems. The first step in achieving this goal is to understand the biological mechanisms behind collaborative behavior in social insects. We will briefly introduce the most important biological principles we will exploit: self-organisation, stigmergy and path integration.

Bonabeau et al. [7] defined self-organisation (SO) as: ”self-organization is a set of dynamical mechanisms whereby structures appear at the global level of a system from interactions among its lower-level components. The rules specifying the interactions among the system’s constituent units are executed on the basis of purely local information, without reference to the global pattern, which is an emergent property of the system rather than a property imposed upon the system by an external ordering influence.” The authors define four basic elements of self-organisation: positive and negative feedback, amplification of fluctuations and interaction.

Eighty years ago, the French entomologist Pierre-Paul Grassé [31] showed that some termite species react to, as what he calls, ’significant stimuli’. He noticed that these reactions on their turn created new stimuli for the insect itself and its fellow insects of the colony. Grassé called this form of communication, in which agents are stimulated based on their achieved performance, ’stigmergy’ [32]. Stigmergy differs from other types of communication by the local indirect transfer of information through the environment.

Stigmergy fulfils a crucial role in the complex mechanism behind individuals coordinating complex activities: it connects individual and collective behavior by generic mechanism. Individual modifications of the environment effects the behavior of the collective, the other individuals.

While exploring the literature on stigmergy in biology, we came across another biological phenomenon that is often used as inspiration in swarm robotics: Path Integration (PI), used by honeybees to navigate. PI is the process of continuous updating and integration all distance covered and all angles steered [13]. The PI vector captures all the knowledge of an insect about its current location relatively to some orientation point. To approximate the PI vector a honeybee makes use of the sun as orientation point [13]. Honeybees always know the PI vector from their current location to the nest, and remember the vector from the nest to the food source. Despite bees can fly most of the time over obstructions, the bees sometimes cannot follow the path of their PI vector. As a work-around the bees divide their PI vector into sub-vectors. Every sub-vector directs the bee a part of the route. The bee connects every sub-vector to some landmark to 'know' when to apply which vector [11, 14]. This extension of the PI method creates flexibility and robustness against obstructions and changes to the environment. If the environment changes, probably not the whole path is useless, by dividing the PI vector in sub-vectors, parts of it can preserved.

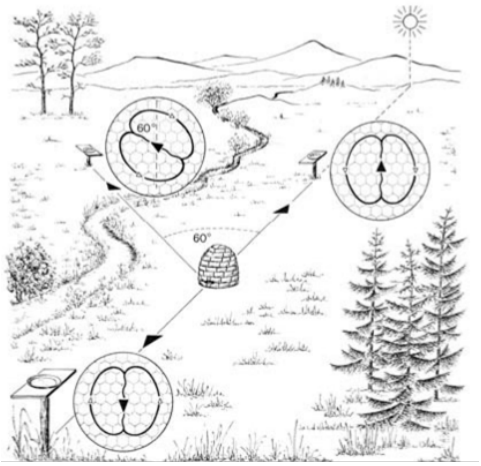


Figure 2.1: Illustration of recruitment dance performed by Honey Bees. The configuration of the dance pattern on the hive relative to the direction of the sun encodes the direction of the food sources [4].

2.2. Classical vs. Swarm Robotics

What are in general the reasons to prefer swarm robotics over single- or (small size) multi-robotic approaches? In the previous section we explained that social insects are able to solve complex problems because of their flexibility, robustness, decentralization and self-organisation. Similar to biology, some engineering problems my be too complex to solve by one or a small group of robots. Additionally, building, using and designing a large amount of simple robots may be cheaper, easier and more robust than several complex powerful robots [88]. Cheaper, since the simpler a robot, the more flexible it is: the less adaptions needed to handle different situations or tasks. Easier, since simple robots are equipped with simpler actuators and sensors. More robust, since simple robots are interchangeable, such that failing robots will not effect task performance. Moreover, Random behavior of individuals can increase a system's ability to explore new solutions. And, self-organisation, decentralization and indirect, stigmergic, communication reduce the required communication among robots dramatically. Classical direct robot-to-robot communication has turned out to be untenable when increasing the number of robots. Also central control is not robust, since failing of the controller direct results in failing of the complete system.

Swarm robotics has disadvantages too. For example, if a system lacks global knowledge it can stagnate: the robots end up at a point where they cannot progress anymore, a deadlock. Or, the inherent (small) heterogeneity of a swarm can result in complexities. The advantages of simple robots will disappear as soon as their assumed uniformness disappears: robots respond different to the same signals.

In this thesis we are interested in swarms based on stigmergy. As it turns out, implementing stigmergy in real-life comes with a lot of difficulties as well.

Let us concertize the general advantages of swarm robotics for the navigation task, the foundation of a wide range of problems considered in the robotic domain. In classical robotics, researchers often equip robots with an explicit map-like representation of their environment [24, 58]. Such a representation may be given a priori, mainly leaving a robot with the non-trivial task of self-localization, or the map may be constructed by the robot itself while moving in the environment. While this is already difficult in a static environment with a single robot, it becomes increasingly complex in dynamic environments, and in particular when multiple robot are considered. Although solutions to such problems have been proposed, complex navigation strategies do not naturally scale with the number of robots, and require careful engineering of the controller. When designing swarm robotics control algorithms for navigation tasks, complex strategies are in general avoided, and instead principles such as locality of sensing and communication, homogeneity and distribudness, are followed.

2.3. Stigmergy in Swarm Robotics

This section provides an overview of the methods proposed in engineering to implement robotic multi-agent systems based on stigmergy. One can distinguish three types of approaches to implement stigmergy:

1. **'Real' Pheromones:** Aims to replicate biological pheromones.
2. **Smart Environment:** Equips the environment to simulate pheromones.
3. **Beacons**: Deploys beacons in the environment as pheromone depositing points.

Figure 2.2 provides an overview of all the methods to implement stigmergy, structured by their implementation characteristics.

In a first attempt to replicate the biological pheromones, one tried to clone pheromones as realistic as possible. One can distinguish four different approaches of attempts to replicate 'real' implementation of pheromones: The use of chemicals tracks [9, 25–27, 34, 70, 78, 80], thermal tracks [77, 79], visual tracks [2, 5, 57, 72, 86] and RFID tags [50–53, 94].

Another approach to implement stigmergic communication is by a smart environment, mostly applied for experimental purposes. We distinguish three different types of smart environments: Systems that use a RFID grid to store local information [8, 39, 41, 44, 81]; systems that use a graphical projector to project local information on the environment [29, 85]; and systems that apply Augmented Reality (AR), which equips robots with virtual actuators and sensors to sense the (virtual) environment [54, 74, 87]. Since these approaches are generally easily adaptable to varying environments or swarm sizes and of low costs, smart environments are well suited for experiments.

The most popular approach to implement stigmergy in recent literature is the use of beacons to store, communicate and optionally process virtual pheromones. One can distinguish methods that implement autonomously moving beacons from beacons that need to be deployed (heterogeneous beacons). Methods that use heterogeneous beacons can be split into methods that use fixed beacons [43, 50–53], beacons that once deployed can not be moved, or movable beacons [36, 76]. Methods that use autonomous beacons can be categorized in methods that use homogeneous [20, 23, 62, 68, 68] or heterogeneous [21, 22] swarms. In a homogeneous swarm all robots have equal capabilities and dynamics. Beacons are implemented in three different ways in a homogeneous swarm:

- Robots have a fixed state: act as beacon or as worker.
- Robots can switch between two state: act as beacon or as worker.
- Robots act simultaneously as beacon and as worker.

With the worker state corresponding to contribute to the task(s) to be solved. We conclude that:

1. The *'Real' Pheromone*-approach is way too complex to implement in practice.
2. Although the *Smart Environments*-approach is interesting for experimental purposes, it is inherently not suited for real applications as it requires central control and environmental modifications.
3. In general the *Beacons*-approach is the most interesting approach. Moreover, as heterogeneous beacons require a beacon dropping and moving mechanism for the robots, it is most efficient to use autonomous robots as beacons. In addition, as one wants to be able to optimize the beacon infrastructure, agents should be able to switch between a beacon and 'worker' state.

So it can be concluded that the most promising approach to implement stigmergy in swarm robotics is by a homogeneous swarm in which robots are able to switch between a beacon and a 'worker' state.

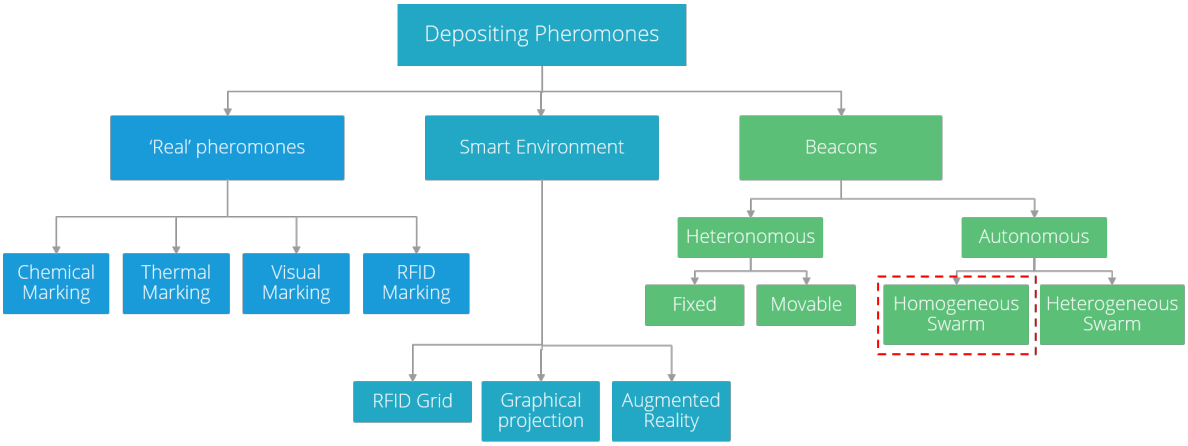


Figure 2.2: Methods to implement stigmergy in Swarm Robotics, structured by their implementation characteristics. Red marked the category we will focus on.

2.4. The Foraging Problem

The most famous example of stigmergy in nature is that of foraging ants: the process of ants searching and returning food to their nest. Ants deposit a trail of chemicals, called pheromones, while moving from the food source location towards the nest. Other foraging ants, equipped with pheromone concentration sensors [90], are attracted to these trails. The ants prefer to follow the trail with the highest pheromone concentration [12]. Ants start exploring randomly the space in search for the food. As paths are created, more ants will choose a trail with the highest concentrations pheromone, resulting in an even higher concentration for this branch. This autocatalysis principle, trail reinforcement by exploiting positive feedback, in combination with the evaporation of the pheromones over time, triggers convergence to the shortest path. The phenomenon of foraging ants is extensively studied in literature.

From the natural pheromone of foraging ants, the abstract foraging problem is derived. The foraging problem is defined as the dual problem of exploration/exploitation, where a number of agents start at a given point in space and must find a target in an unknown (possible time) varying environment, while converging to cycle trajectories that enable them to exploit the found resources as efficiently as possible. Foraging has been extensively studied but it is still interesting when designing very large robotic systems since it combines exploration and on-line path planning, and the duality of exploration vs. exploitation is nowadays extremely relevant in Reinforcement Learning and other AI related fields [37, 61, 89].

2.5. Solutions to the Foraging Problem

Over the years many solutions for the foraging problem have been proposed, most of which inspired on ants and applying the stigmergic principle. Only a few of these solutions are tested in realistic experiments, because many of these systems include implicit assumptions that in practice will prevent them of being applied to large scenarios or real situations. These may be related to sensor range, memory

storage in the agents, computational capacity or reliability of communications, among others. In addition, for many of these solutions the complexity will increase dramatically when designing very large multi-robot swarms, be it in terms of required number of computations, data transmissions or dynamic couplings. Moreover, the previous work has largely been ad-hoc: it assumes a single ant pheromone to help set up a gradient to the food source, plus some arbitrary a priori mechanism to return to the nest (a light source located at the nest, a gradient produced by the nest itself, etc).

In this section we discuss the most promising proposals for decentralized robot foraging systems. Note that these solutions do not necessarily have to rely on stigmergy or ant-inspired mechanisms. We are interested in all possible decentralized multi-agent solutions. Let us classify the proposed solutions into the following categories:

- **Communication Network**:
  Authors in [20, 23] use an ant-inspired swarm to solve a foraging problem on a 2D space by assuming a connected line-of-sight communication network, and having agents flood this network with relative positions information every time step. The robots can measure their relative position to neighbouring agents and every agents keeps a navigation table containing their relative positions to other agents. Every time step all agents broadcasts their navigation table and update their table using its own measures and the tables received by others.

- **Beacon Chains**:
  Authors in [68] propose a multi-robot system that uses IR to communicate a pheromone based counter signal between robots in a cascade. Robots transmit and compute intensity and direction from multiple signals from neighbours and cascade the information through the network, such that agents find an unknown target in space and create trajectories back and forth. These trajectories could be used by other robots to travel between the target and nest. The authors of [62] propose a similar system as [68] but use visual signals (LEDS) to communicate

- **Stigmergic Pheromone Reinforcement**:
  In [36, 76], authors use a combination of agents and beacon devices to guide navigation and store pheromone values. Authors treat pheromones as utility estimates for environmental states, and use utility update and state transitions functions inspired on reinforcement learning. The beacons use line-of-sight communication to detect other beacons and transmit weight values to the robots, The authors demonstrate how agents manage to find trajectories that go back and forth to the target, with and without obstacles, having robots store, deploy (and move or delete) beacons to ensure the feasibility. In section 2.6 we discuss the algorithm developed in [36] in more detail.

- **Landmarks:**
  In [51, 53], authors propose bee-inspired path integration algorithms on a computational set-up, where agents use landmarks to store pheromone-based information when a change in direction is required. Agents explore until they find the goal in a 2D space, and use path integration in combination with the landmarks to integrate their trajectory and find an efficient way back.

In Table 2.1 the most important system and agent characteristics are summarized. Note that, while orientating on to existing concepts and designing of our concept, we pay extra attention to the assumptions on communication capabilities. Because, it is important not to underestimate the difficulties and limitations of implementing the assumed communication protocols, or as [66] states: Unfortunately, learning is a challenging issue in itself, and difficulties associated with it often resulted in a simplified approach to communication, usually neglecting costs of communication with other agents." In Table 2.1 we also classify the type of learning, where we distinguishes team and concurrent learning. Team learning means that a single learner is applied to search for behaviours for the entire team. Concurrent learning means that for each team member a learner is employed.

As Table 2.1 shows, all the *Communication Network*-, *Beacon Chains*- and *Stigmergic Pheromone Reinforcement*-concepts assume agents to communicate directly with other agents, while the *Stigmergic Pheromone Reinforcement*- and *Landmark*-concepts decouple this and propose an environment-based interactions where agents only write and read data into locally reachable beacons. Except for the *Landmark*-concept all reviewed concepts represent collaborative methods and require robots to

have some form of relative position measure. Additionally, most concepts present strong requirements on communication of information in the swarm, either requiring line-of-sight communication or even multi-hop routing of tables of agent data through the network.

Notice that for the *Communication Network*-concepts all agents can contribute to the foraging task. The *Beacon Chains*-concept requires a fixed number of agents to create a beacon infrastructure and the *Stigmergic Pheromone Reinforcement*- and *Landmark*-concepts require large amounts of deployable beacons. We conclude that the theoretical efficiency, i.e. what part of the required resources can contribute directly to the foraging task, of the *Communication Network*-concepts are higher than the other concepts. At last, remark that all concepts require line-of-sight communication and assume that direct communication between agents implies an obstacle free path between these agents. One can think of environments with small corridors or obstacles only blocking parts of the possible paths between agents, in which this assumption is problematic,

	System		Agents					
			Measurement		Communication			
	Collaborative	Learning	Absolute Orientation	Relative Positions	Type	Line of Sight	Identification	Routing
Communication Network [20, 23]	Yes	Team	Yes	Yes	Direct	Yes	Yes, global	Multi-Hop
Beacon Chains [62, 68]	Yes	Team	No	Yes	Direct	Yes	No	Multi-Hop
Stigmergic Pheromone Reinforcement [36, 76]	Yes	Concurrent	No	Yes	Indirect	Yes	Yes, local	One-Hop
Landmark [51, 53]	No	Concurrent	Yes	No	Indirect	Yes	Yes, local	One-Hop
Proposal	**Yes**	**Concurrent**	**Yes**	**No**	Direct	**No**	**No**	**One-Hop**

Table 2.1: Overview of system and agent characteristics of a selection of multi-agent foraging solutions in literature. Bold marked, the characteristics of our proposed concept. The red markings indicate if a characteristics is reasonably assumed to be restrictive and have a negative effect on the performance.

2.6. Stigmergic Foraging Algorithm

Panait and Luke [65] propose a model that handles pheromones as utility measures for local environment states. The proposed robot pheromone dropping and navigation functions feature some similarities from state transition and utility update functions in reinforcement learning. One of the most important distinctive aspects compared to 'ordinary' foraging algorithms is the use of multiple pheromones to create more complex behavior. The researchers see the robots as mobile Automata, that react to and update external states in the environment, and base their choice to update or use a certain pheromone type on their internal state. We will refer to the model of [65] as the *utility model*.

The *utility model* is unique in the stigmergic based foraging literature. As far as we know only [83] also proposed the use of multiple types of pheromones, be it in a biological context trying to model the behavior of desert ants. Since robots generally only use a single pheromone, algorithms rely on ad-hoc mechanisms to return to the nest and implement forms of complex behavior. In fact, most of the contributions in literature even avoid mathematical formalisation of their approaches. The *utility model* does not rely on ad-hoc mechanisms and is inherent mathematical correct formalized. Mainly for this reasons, we will use a similar framework as the *utility model* for the design of our self-guiding swarm. In the remainder of this section we will briefly discuss the main framework of the *utility model*.

Framework

Consider the environment to be captured as a grid, containing one nest location and one or more food sources. The foraging task can be seen as a sequence of two sub-tasks for each robot: the food seeking sub-task; start from the nest and find the food source, and the nest seeking sub-task; start from the food source and find the nest. Which sub-task a robot performs, depends on its internal state. Executing a sub-tasks can be seen as creating a sub-sequence of state transitions: starting at the nest state with the food source as goal state, or vice versa. The robots earn a reward for reaching the goal state.

The algorithm has strong similarities to reinforcement learning; treating the pheromone values as the utility values of specific locations in the environment, the states s. Take the current location of a robot as its external state s. A robot earns a reward, $R(s)$, for reaching the goal state (either the food source or the nest). The utility of a specific state, $U_p(s)$, is equal to the concentration of pheromone type p at location s. Each pheromone is related to a specific sub-task. So in case of the foraging task, which consists of two sub-tasks, each state has two utilities, one per pheromone type. The movements of robots can be seen as state transitions. The state transition policy is simple: move to the adjacent state s with the highest value of the pheromone related to the sub-task the robot is currently performing $U_p(s)$.

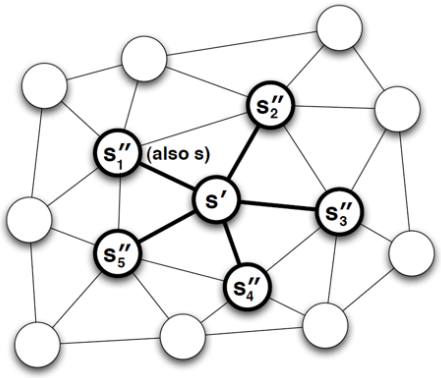


Figure 2.3: Illustration of the state transitions. The circles are beacons, the states. A robot transitioned from s to s', next it can transition to all states s'' [36]

To illustrate the state transitions, consider Figure 2.3, where a robot has recently transitioned from state s to state s' and may transition to any of the s_i'' states. Define the set of all reachable (neighbouring) states from s' as S''. The update rule for each pheromone p is a variation to the Bellman equation in which U_p does not decrease:

$$U_p(s') \leftarrow \max\left(U_p(s'), R_p(s') + \gamma \max_i U_p(s''_i)\right), \quad s''_i \in S'' \tag{2.1}$$

where γ is the learning rate which is a constant between 0 and 1. Next, the robots need to choose an action to perform. With some probability, robots choose a random action. Else, the robot transitions to the state with the highest concentration of the relevant pheromone:

$$s' = \text{argmax}_i \left(U_p(s''_i)\right), \quad s''_i \in S'' \tag{2.2}$$

The authors assume the robots to update all pheromone types parallel, so also the ones not corresponding to their internal state. By this mechanism, while the robot is guided by one pheromone gradient, it builds the gradients of the other pheromones at the same time. This has a significant impact on the efficiency of the gradient building process. Since the robot is updating all the utilities as it moves from the nest to the food (or vice versa), it builds up a gradient in $O(n)$ time, with n the number of state transitions. This is much more efficient than the gradient building efficiency of traditional reinforcement or dynamic programming approaches, whose repeating backups need $O(n^2)$ time. This performance gain is a result of the assumed symmetry of the environment: the transition probability from state s_i to s_j is equal to the transition probability from s_j to s_i. Although an environment is not always symmetric, [65] consider it to be common in many robotics and multi-agent environments. Beside the symmetry assumption, the authors also assume the model to be deterministic: which means that choosing action a in state s_i will always result in an transition to state s_j. This is usually but not necessary the case.

2.7. Coverage & Exploration Control

In this chapter we discuss methods for coverage and exploration control. Part of the foraging problem is to explore the domain efficiently. In Section 2.7.1 we will explain a (single) coverage process named the Probabilistic Coverage (PC)-Process [91] which includes a random walk on a continuous convex space. The authors of [91] show extensive formal results for the PC-Process which we will use in the formal analysis of the exploration behaviour of our swarm design. In Section 2.3 we concluded that the use of beacons is most interesting to implement stigmergy in swarm robotics. This decision brings us to the question how to position the beacon such that an optimal infrastructure of storage points for local information is created. Therefore, we discuss in Section 2.7.2 several approaches to realize dynamic coverage control.

2.7.1. Probabilistic Coverage

Wagner et al. [91] present a random method for exploring a continuous unknown planar domain and derive extensive formal results for its expected performance. The authors show that the expected cover time is proportional to the electrical resistance of a domain, which on its turn is related to geometrical properties of the domain. Hereby they extend similar results regarding the cover time of graphs as derived by [10]. In this subsection we will summarize the main results, which we will use in Section 4 to derive formal results for our swarm design.

First, let us discuss some general results for the online covering problem, which is formally defined as the problem to find a local rule of motion that will cause a robot to follow a space covering curve, such that every point of the given region is in some prespecified r-neighborhood of the robot's trail, r being the covering radius of the robot. Such a rule, if obeyed for a sufficient number of steps, should lead the robot to follow a covering path which is a polygonal curve defined by the points $x_0, x_1, \dots, x_T$, that covers a region $\mathcal{D}$. Let us define $\mathcal{R}_r(x_i)$ as a disk of radius r around x_i, $\mathcal{R}_r(x) \coloneqq \{x' : \|x' - x\|_2 \leq r\}$, and R_r as the r-circle around z, $R_r(x) \coloneqq \{x' : \|x' - x\|_2 = r\}$. The covering path can be formalized as,

$$\mathcal{D} = \bigcup_{k=0}^{T} \mathcal{R}_r(x_k), \tag{2.3}$$

where for all $i, |x_{i+1} - x_i| \leq r$. The shape of $\mathcal{D}$ is not known in advance. The authors of [91] show a lower bound on the length of any covering path, independent of the algorithm used to generate it.

Lemma 1. [91] *The number of points in a covering sequence of r-circles, say $X = x_0, x_1, \dots, z_{T_c}$, such that $|z_{i+1} - z_i| \leq r$, is bounded from below*

$$T_c \geq \left\lceil \frac{6\pi}{4\pi + 3\sqrt{3}} (A/a) - 1 \right\rceil \tag{2.4}$$

where A is the region's area and $a = \pi r^2$ is the area covered by the robot in a single step.

This result is combined with the work of [40], who derived the minimum number of circles to cover a region. Define $N(r)$ as the minimum number of r-circles needed to cover a region of area A. Then

$$\lim_{r \to 0} N(r) = (2\pi\sqrt{3}/9)(A/a) \tag{2.5}$$

and the minimum is attained in the "honeycomb" (hexagonal) arrangement of the circles. This result from [40] implies that, asymptotically, the number of cover steps T_c cannot go below $1.209.. \times (A/a)$ while Lemma 1 implies that for any value of r, $T_c \geq 1.06... \times (A/a)$

Next, we narrow our scope to the problem of covering a μ-size-grid polygon of size n, i.e. a polygon made of a connected set of n unit squares on the grid. Denote the set of μ-size-squares in $\mathcal{D}$ by $S = \{s_1, s_2, \dots, s_n\}$. This partition is not known to the agents, but only serves analysis purposes.

Consider a discrete time dynamics, with time steps $k \in \mathbb{N}$ and take the location of an agent at time step k as $x(k)$. In the case of Probabilistic Coverage the agents move can be described as a random walk with variable step size. We will present the PC-process as introduced by [91] in some more

generic formulation. Define the maximum step size by $h_0 \in \mathbb{R}_+$ and restrict it to be always smaller than the coverage radius: $h_0 \leq r$. Note that, although [91] present their results taking $h_0 = r$, all the formal results are derived only assuming $h_0 < r$. Algorithm 1 presents the generalized PC-algorithm.

Algorithm 1: Generalized PC Process [91]

set $x(0) = x_0$;
for $k = 0$ **to** T **do**
  cover $\mathcal{R}_r(x_k)$;
  set $h = \min\{h_0, \max_{\mathcal{R}_{2h'}(x_k) \subset \mathcal{D}}\{h'\}\}$;
  choose a random point w **from** $R_h(x_k)$;
  set $x_{k+1} = w$;

Let us first introduce several definitions: Take $Q = \{q_1, q_2, \dots, q_n\}$ as a collection of subsets of the set $\mathcal{D}$, then the time for visiting some point of every subset in Q is defined by $T(q_1, q_2, \dots, q_n)$. The time it takes an agent to cover all n square grids of $\mathcal{D}$ is defined as the *cover time*, $T^{TC} \in \mathcal{R}_+$. Define the *hitting time* from a point $x \in \mathcal{D}$ to a square s_j, denoted by $h_j(x) \in \mathbb{R}_+$, as the expected time of a PC-process that starts at x and ends upon first reaching a point in square s_j. Also define *commute time* between squares s_i and s_j, denoted by $C_{i,j} \in \mathbb{R}_+$, as the average time of a round trip from s_i to s_j and back, i.e. $C_{i,j} = C_{j,i} = \max_{x \in s_i, y \in s_j}\{h_j(x) + h_i(y)\}$

Next, observe that the (random) location of an agent at $k+1$ only depends on its previous location x_k, hence the PC-process is strong Markov Process. It was proved in [56] that under such a process $\mathbb{E}[T(q_1, 1_2, \dots, q_n)]$, the expected time for visiting some point of every subset in Q (starting from anywhere in $\mathcal{D}$), is bounded as follows:

$$h_{max} \leq \mathbb{E}[T(q_1, q_2, \dots, q_n)] \leq h_{max} \sum_{i=1}^{n} 1/i \tag{2.6}$$

where $h_{max} = \max_{x \in (\mathcal{D} \setminus Q), 1 \leq i \leq n}\{h_i(x)\}$, and $h_i(x)$ is the expected time to first reach subset q_i upon starting from $x \in \mathcal{D}$. Equation (2.6) implies that the expected *cover time* of $\mathcal{D}$ can be bounded as:

$$\max_{s_i, s_j \in \mathcal{D}}\{C_{i,j}\} \leq \mathbb{E}[T^{PC}] \leq 2(\log n) \max_{s_i, s_j \in \mathcal{D}}\{C_{i,j}\} \tag{2.7}$$

where it is used that using $\sum_{i=1}^{n}(1/i) < 2 \log n$.

In order to find the maximum commute time ($C_{i,j}$) in $\mathcal{D}$, it is shown that the commute time between any square s_i, s_j in $\mathcal{D}$ is proportional to the product of the number of squares in $\mathcal{D}$ and the electrical resistance between s_i and s_j. The following Lemma is derived as a continuous analog to [10].

Lemma 2. [91] *$C_{i,j}$, the commute time between squares s_i and s_j in $\mathcal{D}$, is given by*

$$C_{i,j} = \frac{4n}{r^2} \rho_{i,j} \tag{2.8}$$

where ρ is the electrical resistance of $\mathcal{D}$, assuming a material of unit sheet-resistance. The sheet resistance of a material is defined as the voltage across a square of the material caused by one unit of current (i.e. one Ampere) that is flowing between two parallel edges of the square. The sheet resistance is expressed in units of Ohms per square.

Combining Equation (2.7) and Lemma 2 results in Theorem 1

Theorem 1. [91] *$\frac{4n}{r^2}\rho \leq \mathbb{E}[T^{PC}] \leq \frac{8n}{r^2}\rho \log n$, where n is the size of R and ρ its resistance.*

The resistance $\rho = \rho(\mathcal{D})$ can sometimes be bounded in terms of geometrical properties of the shape, and can always be numerically approximated. For example, for $\mathcal{D}$ an $a \times b$ rectangle with $a < b$ it holds that $\rho = \mathcal{O}(a/b)$.

Corollary 1. [91] *If $\mathcal{D}$ is a square $a \times a$ room, then*

$$c_1 a^2 \log a \leq \mathbb{E}[T^{PC}] \leq c_2 a^2 \log^2 a \tag{2.9}$$

where c_1, c_2 are small constants.

The authors also shown that the variance of the cover time, denoted $\text{Var}[T^{PC}]$, is also bounded from above by:

$$\text{Var}[T^{PC}] \leq 2^{10} \max_{s_i, s_j \in R} \{C_{i,j}\} \leq \frac{2^{12}}{r^2} n\rho \tag{2.10}$$

2.7.2. Dynamic Coverage Control

In Section 2.3 we concluded that the *Beacons*-approach is the most promising approach to implement stigmergy in swarm robotics. In Section 2.6 we discussed the *utility model*, from which we will borrow several aspects for our design, among which the location dependent state and the navigation by two types of pheromone fields. The beacons will store the local utilities, and all beacon combined create the pheromone fields used by foraging agents to navigate. These decisions take us to the following challenge: How do we control the beacons to cover the environment that such they create the optimal infrastructure to the information of the local pheromone fields? In this subsection we will evaluate several methods by their ability to create uniform - and non-uniform coverage based on some desired density function, and the required capabilities of the beacons to achieve such performance, such as restricted local communication and distance measures. In particular, we are interested in methods which can perform non-uniform coverage without complete knowledge of a global density function.

The Stico (Stigmergic coverage approach) method [2, 71–73] is based on very simple rules of motion and uses continuous pheromone trails as source of repulsion instead of attraction. The StiCo approach shows the power of negative feedback and stigmergy based on 'real' pheromones to perform dynamic coverage. The ID-Stico (Intruder Detection StiCo) approach shows that dynamic non-uniform coverage only requires a variable range of territory per beacon, i.e. a variable distance to other agents.

The Potential Fields approach [35, 69] applies a physical metaphor of repellent, attractive and viscous forces created by detected surrounding objects or robot movements, to create uniform dynamic coverage. In principle it is not possible to adjust this approach to achieve non-uniform coverage, while preserving the approach's main pro of agents only requiring local distance information. If we would want to adjust the potential field approach to create non-homogeneous coverage, we would need to connect the forces acting on a robot to its internal state or assume local influences on the forces, which would result in a mechanism almost similar to Voronoi-Partitioning based coverage control.

In the field of dynamic coverage control, density function are used to realise non-uniform coverage. These density functions capture for example the change that an event happens at a sub-region of the environment. The goal is to find a control algorithm to locate the robots such that the change of a swarm to perceive an event is maximized. As it turns out, optimal control laws enforce the robots to create centroidal Voronoi tessellations (CVT), i.e. the optimal control law forces each robot to move towards the center of mass of the Voronoi regions calculated using the density function. Various algorithms that guarantee converge to CVT have been proposed [15, 16, 48, 49, 82]. We will refer to these control method as *Voronoi Partitioning* coverage control. Although some algorithms have been developed that use distributed control actions, *Voronoi Partitioning* coverage control has one major drawback: it requires knowledge of a global density function. In our case the pheromone field created by the pheromone values and stored at the beacons can act as the density function. However, several challenges appear, such as: how do we create a continuous density function based on the pheromone values stored in the Beacons? Linear combinations of Gaussians could do the job, though this would require relative positional knowledge of neighbouring beacons. Another challenge: How robust is the coverage system under fast changing pheromone concentrations? We conclude that if we manage to implement the *Voronoi Partitioning* coverage control by combinations of local density functions this method would perfectly solve the dynamic coverage control task of the beacons of our swarm. However, as this algorithm itself would be a valuable addition to the existing literature, this is very challenging.

2.8. Path Planning Algorithms

The foraging problem consists of the general path planning problem. Ant Colony Algorithms have been widely applied to path planning problems and in general as (discrete) path optimization tool [19, 28, 55], even some of the methods discussed in Section 2.5 have been directly applied as offline path optimization tool [52].

Basic path planning algorithms find their origin in *Dijkstra's* algorithm and a special case of *Dijkstra's* algorithm, the centralized *A** [18, 33]. These algorithms uniformly discretize the search space and afterwards plan an optimal path through the discrete search space. Algorithms such as *Dijkstra's* and *A** show good performance in static, limited-sized and single agent path planning problems. However, real-world problems often involve uncertainty and dynamics in a large search space. Traditional path planning algorithms are not able to cope with such problems due to memory issues and computation time. In order to overcome these problems, researches use heuristics to guide path planning, such as potential field techniques [3] and decentralized algorithms [84, 92]. Note that the decentralization only implies decoupling of the problem per agent for multi-agent path planning. Although such techniques reduce the issues, performance still suffers with increasing complexity. In general given a single agent problem with an environment discretized using a grid of m squares, the running time and memory scales by $\mathcal{O}(m^2)$. Remark that the most efficient algorithms rely on unified discretization of the environment, mostly by a squared grid. Uncertainty or dynamical environment or discretizations will only increase the required memory and running time.

Note that the *Communication Network*- and *Beacon Chain*-concepts as discussed in 2.5 resemble some aspects of *Dijkstra*-algorithms. In fact the message propagation policy as proposed by [69], is a distributed version of the wavefront-propagation-method of the *Dijkstra's* algorithm.

Randomized approaches such as *RRTs* (Rapidly-Exploring Random Trees) [42, 46, 47, 47] and *Probabilistic Roadmaps* [45] offer a solution to the issues occurring for the *Dijkstra*-type path planning algorithms by randomly sampling points in the (continuous) search space and then grow the current solution toward these points

As it turns out, the stigmergic concept we have been interested in up till now: deploying an infrastructure of beacons, has much in common with the RRT algorithm. In the remainder of this section we will shortly discuss the idea behind the RRTs and describe the algorithm of the most basic version a RRT. We will employ the similarities between RRT and our design while deriving formal results.

2.8.1. Rapidly Exploring Random Trees (RRTs)

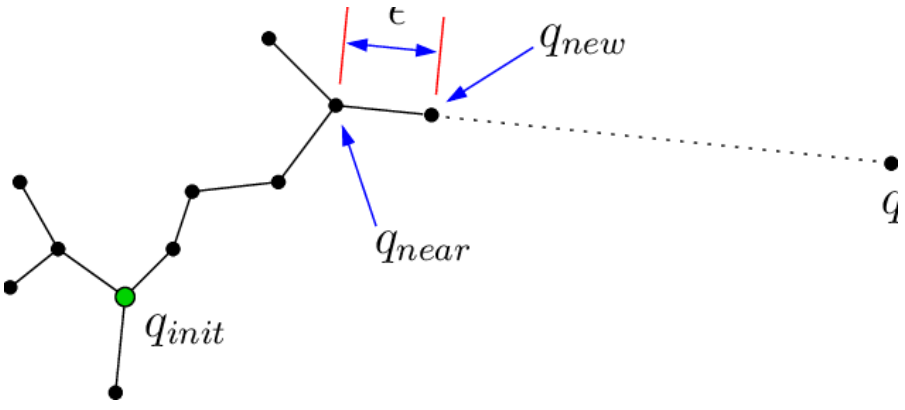


Figure 2.4: The extend operation of the RRT Algorithm [42]

RRTs [42, 46, 47, 47] build on ideas from optimal control theory, non-holomonic planning, and randomized path planning. The basic idea is to incrementally grow a search tree from an initial state by applying control inputs over short time intervals to reach new states. Each vertex in the tree represents a state, and each directed edge represents an input that was applied to reach the new state from a previous

state. When a vertex reaches a desired goal region, a trajectory from the initial state is represented by the tree. The key idea is to bias the exploration toward unexplored portions of the space by sampling points in the state space, and incrementally ”pulling” the search tree toward them.

The basic RRT construction algorithm is given by Algorithms 2 and 3. A simple iteration is performed in which each step attempts to extend the RRT by adding a new vertex that is biased by a randomly-selected configuration. The EXTEND function, illustrated in Figure 2.4, selects the nearest vertex already in the RRT to the given sample configuration, q. The function NEW_CONFIG makes a motion toward q with some fixed incremental distance ϵ, and tests for collision. Three situations can occur: $Reached$, in which q is directly added to the RRT because it already contains a vertex within ϵ of q; $Advanced$, in which a new vertex $q_{nes} \neq q$ is added to the RRT; and $Trapped$, in which the proposed new vertex is rejected because it does not lie in C_{free}.

Algorithm 2: [91] BUILD_RRT(q_{init})

```
1 T.init(q_init);
2 for k = 1 to K do
3 |   q_rand ← RANDOM_STATE()
4 |   EXTEND(T, q_rand)
5 Return T
```

Algorithm 3: [91] EXTEND($\mathcal{T}, x$)

```
1 q_near ← NEAREST_NEIGHBOR(q, T);
2 if NEW_STATE(q, q_near, q_new) then
3 |   T.add_vertex(q_new);
4 |   T.add_edge(q_near, q_new);
5 |   if q_new = q then
6 |   |   Return Reached;
7 |   else
8 |   |   Return Advanced;
9 |   Return Trapped
```

2.9. Preliminaries

We use calligraphic letters for sets ($\mathcal{A}$), regular letters for scalars ($a \in \mathbb{R}$) and bold letters for vectors ($\mathbf{a} \in \mathbb{R}^n$). We consider discrete time dynamics $k \in \mathbb{N}$, and we define an inter-sampling time $\tau \in \mathbb{R}_+$ such that we keep a "total" time measure $t = \tau k$. With vectors we use $||v||$ as the euclidean norm, and $\langle v \rangle \coloneqq \frac{v}{||v||}$.

2.10. Problem Description

Before we formally define the foraging problem and our assumptions, let us first define the swarm and its environment. A swarm consists N agents $\mathcal{A} = \{1, 2, ..., N\}$ navigating in a bounded domain $\mathcal{D} \subset \mathbb{R}^2$, where $\mathcal{D}$ is compact (possibly non-convex). We define $x_a(k) \in \mathcal{D}$ as the position and $\alpha_a(k)$ as the heading direction of agent a at time step k. Take v_0 the constant speed of an agent, then, its velocity is given by $v_a(t) = v_0 \begin{pmatrix} \cos\alpha(k) & \sin\alpha(k) \end{pmatrix}^T$ with $\alpha_a(k) \in [-\pi, \pi)$. We consider the dynamics of the system to be discrete time, such that the positions of the agents evolve as

$$x_a(k+1) = x_a(k) + v_a(k)\tau,$$

The region $\mathcal{D}$ contains two sub-region, $\mathcal{S}$ and $\mathcal{T}$ both of radius $\delta_{S,T}$.

Definition 1 (Foraging Problem)**.** *A foraging problem on an unknown domain $\mathcal{D}$ is the joint problem of finding a target region $\mathcal{T} \subset \mathcal{D}$ when starting from a different region $\mathcal{S} \subset \mathcal{D}$, $\mathcal{S} \cap \mathcal{T} = \emptyset$, and eventually following (semi) optimal trajectories between $\mathcal{S}$ and $\mathcal{D}$.*

The swarm aims to solve the *foraging problem*. Usually the main goal of the foraging problem is said to be: complete trajectories between $\mathcal{S}$ and $\mathcal{D}$ as fast as possible (through the shortest path), back and fourth. Since for our swarm some of the agents will facilitate the infrastructure to solve the foraging problem, we generalize the goal to: maximize the trips between $\mathcal{S}$ and $\mathcal{D}$ performed by all agents of the swarm. We consider the foraging problem to be solved if, eventually, all foraging agents in the swarm are able to follow trajectories that are relatively close to the set of optimal trajectories.

As we have shown in Section 2.5, literature has proposed many approaches to solve the foraging problem. With implementation in real robots as ultimate goal, the performance of a concept should be interpreted relatively to the assumed robotic capabilities. In Table 2.1 we presented an overview of the most interesting solutions to the *foraging problem* structured by assumed capabilities. We will make the following assumptions on the swarm agents' capabilities.

Assumption 1 (Swarm Constrains)**.**

1. *All agents have equal capabilities and dynamics, i.e. the swarm is homogeneous.*
2. *Agents have a small memory, enough to store scalars and vectors in $\mathbb{R}^2$, and enough computational power to perform sums and products of vectors.*
3. *Agents have the ability to send and receive basic signals (up to 6 scalar values), within a maximum range δ_s.*
4. *Agents have some form of collision avoidance mechanism, acting independently of the design dynamics.*
5. *Agents have sensing ability to detect $\mathcal{S}$, $\mathcal{T}$.*
6. *Agents have some measure of angular orientation (*e.g. *a compass).*
7. *Agents are able to remain static.*
8. *Agents are able to move in any direction (holonomic or omin-directional agents)*

Given these, and the assumptions made by others in the field (Table 2.1), we explicit want to emphasize what we not assume:

- Agents do not have access to any form of global information about $\mathcal{D}$.

- The swarm does not require either any form of infrastructure in place.
- We do not assume the ability to measure directionality in the signals, nor line-of-sight interactions. Therefore, we are not restricted to use the IR or visual medium, but can use radio signals as well.
- We do not assume any form of self-localisation capacity.
- Agents do not have unique identifiers.
- The swarm relies on one-hop communication only. That is, the communication happens on a agent-to-agent basis, and agents do not cascade information through the communication network.

The challenge to be solved in this work is then the following.

Problem 1. *Design a swarm of N agents that solves a foraging problem over an unknown domain $\mathcal{D}$, while satisfying the set of Assumptions 1, and does so with guarantees.*

2.11. Summary Concepts

In this Chapter we created the foundation for our work. We presented the main insights of literature and discussed our decisions on the concepts to use. Let us summarize the main takeaways:

- We explained the biological principles of Self-Organisation, Stigmergy and Path Integration, and introduced the foraging ants phenomenon.
- We showed an overview of all different approaches to implement stigmergy in robotics. We concluded that it is most effective to store local information in beacons, which can either be passive storage devices or the robots themselves. Furthermore we decided that we will implement the *Beacon*-approach using a homogeneous swarm in which robots are able to switch between a beacon and a 'worker' state.
- We introduced the foraging problem and discussed the proposed solutions for it. In general we observed that many of these solutions include implicit assumptions regarding the sensor range, memory storage in the agents, computational capacity or reliability of communication. In practice these assumptions will prevent the solutions of being applied to real situations or large scenarios. In addition, for many of these solutions the complexity will increase dramatically when designing very large multi-robot swarms. Moreover, the previous work has largely been ad-hoc: it assumes a single ant pheromone to help set up a gradient to the food source, plus some arbitrary a priori mechanism to return to the nest
- We discussed the most promising multi-agent solutions found in literature: creating a communication network of beacon agents; creating beacon chains by cascading information through trajectories of beacons; treating pheromones as utility estimates for environmental states stored in deployable beacons; and use landmarks to apply bee-inspired path integration algorithms. We presented an overview of the system and individual agent characteristics implied by each class of solutions (see Table 2.1). We conclude that the interpretation of the pheromones as utilities proposed by the *Stigmergic Pheromone Reinforcement* is an interesting strategy which allows for formalisation and application of reinforcement learning techniques. Moreover the *Landmark* approach of storing navigation information using path integration is very interesting as it drops the need of (relative) location measures.
- We explained the *utility*-model a *Stigmergic Pheromone Reinforcement* concept as proposed by [65] in more detail.
- The use of beacons to implement stigmergy and treat pheromones as utilities took us to the next challenge: How to control the beacons to cover the environment such that they create an optimal global infrastructure to store the pheromone fields? We pointed out three options and concluded that the so called *Voronoi Partitioning* coverage control option is most suited for purposes, although several challenge regarding the construction of a global density function remain.
- We discussed the concept of the Rapidly Exploring Random Trees (RRTs) which share similar principles regarding the building process our beacon infrastructure that can be used to derive formal results.
- At last, we present our problem description in Section 2.10.

3

Proposal: Self Guided Swarm

In this Section we present the design of a self-guided swarm that solves the foraging problem as presented in Section 2.10. We will continue the work of Adams et al. [1]. In Subsection 3.1 we will first explain the basic concept as presented in [1], after which we present in Subsection 3.2 several extensions to the basic self-guided swarm. At last, as the level of abstractions of the mathematical models detracts from the intuitive interpretation of the intended systems dynamics, we describe in Section 3.3 the intended behaviour of the self-guiding swarm on the basis of a more intuitive description, using visualizations and analogies.

3.1. Basic Concept

The design of the self-guiding swarm as presented in [1] is inspired on the dynamic task allocation of Social Insects [6, 38, 64, 75]. Although the swarm is homogeneous, agents will fulfill different roles: An agent can behave as ”beacon”(agent in charge of guiding others) or ”forager” (agent in charge of travelling between $\mathcal{S}$ to $\mathcal{T}$). We first describe the different states of an agent, introduce the region of influence, introduce the switching rules between states, and last define the dynamics in every mode.

3.1.1. States

Besides the distinction between agents acting as beacon and forager, we distinguish foraging agents looking for $\mathcal{T}$ and foraging agents looking for $\mathcal{S}$, hereby creating in total three types of agents. We use the state variable $s_a(k) \in \{B, F_1, F_2\}$ to indicate:

$$s_a(k) = B \Rightarrow a \text{ is a beacon,}$$
$$s_a(k) = F_1 \Rightarrow a \text{ is a forager searching for} \mathcal{T},$$
$$s_a(k) = F_2 \Rightarrow a \text{is a forager looking for} \mathcal{S}.$$

To group the agents in time-dependent sub-sets, we use s to refer to the foraging states F_1 and F_2, and $\overline{s}$ to refer to the inverse foraging state: $\overline{F_1} = F_2$ and $\overline{F_2} = F_1$. Then, we define $\mathcal{B}(k) \coloneqq \{a \in \mathcal{A} : s_a(k) = B\}$ as the beacon set, $\mathcal{F}(k) \coloneqq \{a \in \mathcal{A} : s_a(k) \in \{F_1, F_2\}\}$ as the forager set, and $\mathcal{F}^s(k) \coloneqq \{a \in \mathcal{A} : s_a(k) = s\}$ as the set of foragers in state s.

3.1.2. Region of Influence

We assumed that agents have the ability to send and receive signals within a maximum range δ (see Assumption 1). We will use this maximum communication range to define the regions of influence of every agent as $\mathcal{D}_a(k) \coloneqq \{x \in \mathcal{D} : \|x - x_a(k)\|_2 \leq \delta\}$. Let us now define the set of neighbouring beacons to an agent, $a \in \mathcal{A}$ as:

$$\mathcal{B}_a(k) \coloneqq \{b \in \mathcal{B}(k) : x_b(t) \in \mathcal{D}_a(k)\}, \tag{3.1}$$

the set of neighbouring foragers to a beacon $b \in \mathcal{B}(k)$ as

$$\mathcal{F}_b(k) \coloneqq \{f \in \mathcal{F}(k) : x_f(k) \in \mathcal{D}_b(k)\} \tag{3.2}$$

and the set of neighbouring foragers in state s to a beacon $b \in \mathcal{B}(k)$ as

$$\mathcal{F}_b^s(k) \coloneqq \{f \in \mathcal{F}^s(k) : x_f(k) \in \mathcal{D}_b(k)\} \tag{3.3}$$

3.1.3. State Transitions

At $t = 0$ all agents start at $\mathcal{S}$. One of the agents is initialized as beacon, all others are initialized as foragers looking for $\mathcal{T}$:

$$x_a(0) \in \mathcal{S}\ \forall a \in \mathcal{A},\ |\mathcal{B}(0)| = 1,\ |\mathcal{F}^{F_1}(0)| = N - 1.$$

Which agent starts as initial beacon is chosen at random, or by some predetermined order of deployment of the swarm. As time evolves, the agents switch between states following the logic rules:

$$s_a(k+1) = \begin{cases} "B" & \text{if } \mathcal{B}_a(k) = \emptyset, \\ "F_1" & \text{if } s_a(k) = "F_2" \wedge x_a(k) \in \mathcal{S}, \\ "F_2" & \text{if } s_a(k) = "F_1" \wedge x_a(k) \in \mathcal{T}, \\ s_a(k) & \text{else,} \end{cases} \quad \forall a \in \mathcal{A}. \tag{3.4}$$

The switching rule in (3.4) is interpreted in the following way. If a forager moves to a point in the domain where there is no other beacons around, it becomes a beacon. If a forager is looking for the set $\mathcal{T}$ (state F_1) and finds it, it switches to finding the starting set $\mathcal{S}$ (state F_2), and the other way around. For now we will not consider transitions from beacon to forager, but we will explore in Section 3.2 how the swarm can further optimize its resources by allowing beacons to return to forager states under certain conditions.

3.1.4. Dynamics

In Section 3.2 we will explore the option to optimize the beacon infrastructure by moving the beacons, for now we will assume that beacons remain static while in beacon state. So,

$$v_b(k) = \mathbf{0},\ x_b(k) = x_b(k_b)\ \ \forall k \geq k_b, \tag{3.5}$$

where k_b is the time step when agent b switched to beacon state. Beacon agents store weight values $\omega_b^s(k) \in \mathbb{R}_+$ and guiding velocity vectors $v_b^s(k) \in \mathbb{R}^2$, both initialised at zero for all agents in the swarm. At every time-step, beacons broadcast their stored values $\omega_b^s(k), v_b^s(k)$ with a signal of radius δ. At every time-step each forager receives a set of signals from neighbouring beacons, and compute a reward function $\Delta_f^s(k) \in \mathbb{R}_+$:

$$\Delta_f^s(k) = \gamma_f^s(k) + \lambda \max_{b \in \mathcal{B}_f(k)} \omega_b^s(k), \tag{3.6}$$

where $\lambda \in [0, 1]$ is a discount factor, called the diffusion rate, and,

$$\gamma_f^s(k) = \begin{cases} r & \text{if } s_f(k) = "F_1" \wedge x_f(k) \in \mathcal{S}, \\ r & \text{if } s_f(k) = "F_2" \wedge x_f(k) \in \mathcal{T}, \\ 0 & \text{else.} \end{cases} \tag{3.7}$$

where $r \in \mathbb{R}_+$ is called the reward. The reward function in (3.6) should be interpreted as follows: Foragers listen for weight signals from neighbouring beacons and broadcast back the maximum discounted weight plus a constant reward if they are, depending on their state, in the regions $\mathcal{S}$ or $\mathcal{T}$. Observe that (3.6) depends on s, indicating that foragers listen and reinforce only the weights corresponding to their internal state value. After listening for a period τ, the beacons update their weight values using a discount factor $\rho_w \in [0, 1]$, called the weight's evaporation rate, as

$$\omega_b^s(k+1) = (1 - \rho_w)\omega_b^s(k) + \rho_w \frac{\sum_{f \in \mathcal{F}_b^s(k)} \Delta_f^s(k)}{|\mathcal{F}_b^s(k)|}. \tag{3.8}$$

The iteration in (3.8) is only applied if there are indeed neighbouring foragers around a beacon, so $|\mathcal{F}_b^s(k)| \geq 1$. Otherwise, no iteration step is applied at that instant. The update rule of $v_b^s(k)$ is similarly:

$$v_b^s(k+1) = (1 - \rho_v)v_b^s(k) + \rho_v \frac{\sum_{f \in \mathcal{F}_b^s(k)} -v_f(k)}{|\mathcal{F}_b^s(k)|}. \tag{3.9}$$

with $\rho_w \in [0,1]$ a discount factor, called the guiding velocities evaporation rate. The update rule for $v_b^s(k)$ works as follows: at the same time that beacons update their stored weight values based on the foragers around them, they update as well the guiding velocity vectors by adding the corresponding velocities of the foragers around them (with an opposite sign). The logic behind this has to do with the reward function in (3.6). Foragers looking for $\mathcal{T}$ update weights and guiding velocities associated with state F_1, but to indicate that they are in fact moving out of $\mathcal{S}$, we want to update the guiding velocities based on the opposite direction that they are following.

Now we have defined the dynamics of the beacon agents: position and velocities in (3.5) and update rules for $\omega_b^s(k)$, $v_b^s(k)$ in (3.8) and (3.9), we only have to define the dynamics of the foragers. At every step, the foragers listen for guiding velocity and weight signals from beacons around them. With this information, they compute the guiding vector:

$$\hat{v}_f^s(k) := \left\langle \sum_{b \in \mathcal{B}_f(k)} \omega_b^{\bar{s}}(k) v_b^{\bar{s}}(k) \right\rangle. \tag{3.10}$$

Note that we used the inverse foraging state to compute the guiding vector. A foraging agent in state F_1 reinforces the weights and guiding velocities corresponding to state F_1 and uses the fields of state F_2 to determine its movement. At every time-step foragers choose stochastically, for a design exploration rate $\varepsilon \in (0,1)$, if they follow the guiding vector $\hat{v}_f^s(k)$ or they introduce some random noise to their movement. Let α_u be a random variable taking values $(-\pi, \pi)$, following some probability density function $p(\alpha_u)$. Then, the velocity of the foragers follows the stochastic evolution:

$$\begin{aligned} &\Pr\{v_f(k+1) = v_0(\cos(\alpha_a(k) + \alpha_u) \;\; \sin(\alpha_a(k) + \alpha_u))^T\} = \varepsilon, \\ &\Pr\{v_f(k+1) = \hat{v}_f^s(k)\} = 1 - \varepsilon, \end{aligned} \tag{3.11}$$

for all $f \in \mathcal{A}^s(k)$. Additionally, we will add a fixed condition for an agent to turn around when switching between foraging states. That is,

$$v_f(k+1) = -v_f(k) \quad \text{if } s_f(k+1) \neq s_f(k+1). \tag{3.12}$$

With (3.11) and (3.12) the dynamics of the foragers are defined too.

3.2. Extended Concept

For the basic concept we did not considered transitions from beacon to forager, and set agents to remain static once switched to the beacon state. Once the swarm has explored the environment and created paths between regions $\mathcal{S}$ and $\mathcal{T}$, this approach has two drawbacks:

1. Agents in beacon state that do not store any relevant information, i.e. store very small weight values and guiding velocity vectors, do not contribute in solving the foraging problem: they can not act as foragers, nor do they create the infrastructure for the paths to guide the foraging agents.
2. First, remind that the environment is randomly explored by the agents, and as a result the positional configuration of the beacons is created at random. Second, realize that the configuration of the beacons determines the set of paths that can be stored in the infrastructure. Therefore, the optimality of the (to be) created paths is limited by the randomly created configuration of the beacons.

In an attempt to overcome drawback 1 we introduce in Subsection 3.2.1 the *Beacon-Forager Switching*-extension which enables beacon agents to switch back to a foraging state. In Subsection 3.2.2 we introduce the *Moving Beacons*-extension to enable beacon agents to optimize their position and solve drawback 2.

In the basic concept, foraging agents only reinforce the weight values and guiding velocities corresponding to their internal state. [36] showed for a similar solution to the foraging problem an increase in convergence if foragers in addition reinforce the to-follow pheromone. Although the approach of these authors differs from our approach on some important aspect, which we will explain in Subsection 5.6.2, we want to investigate experimentally if the double updating principle can improve the performance of our swarm too. So, at last, in Subsection 3.2.3 we will introduce the *Double-Updating*-extension which makes forager agents simultaneously reinforce the weight and guiding velocities of both foraging states.

3.2.1. Beacon-Forager Switching

Before we explain the conditions under which we allow an agent to switch from beacon state to forager state, let us define the last foraging state of a beacon agent by: $s_a^-(k)$, formally defined as:

$$s^-(k) \coloneqq s(k^-), \quad \text{with} \quad k^- \coloneqq \max(\{k_{\text{switch}} | k_{\text{switch}} \in \mathbb{N}, k_{\text{switch}} < k, s(k_{\text{switch}}) \neq s(k)\}) \tag{3.13}$$

To allow for switching from beacon state back to forager state, the logic rules for state switching (see (3.4) is updated as:

$$s_a(k+1) = \begin{cases} "B" & \text{if} \quad s_a(k-2) \in \{"F_1", "F_2"\} \wedge \mathcal{B}_a(k) = \emptyset, \\ "F_1" & \text{if} \quad s_a(k) = "F_2" \wedge x_a(k) \in \mathcal{S} \text{ or} \\ & \quad\quad s_a(k) = "B" \wedge s^-(k) = F_1 \wedge \sum_s w_a^s(k) < \eta_w \wedge \mathcal{F}_a = \emptyset \\ "F_2" & \text{if} \quad s_a(k) = "F_1" \wedge x_a(k) \in \mathcal{T} \text{ or} \\ & \quad\quad s_a(k) = "B" \wedge s^-(k) = F_2 \wedge \sum_s w_a^s(k) < \eta_w \wedge \mathcal{F}_a = \emptyset \\ s_a(k) & \text{else} \end{cases} \quad \forall a \in \mathcal{A}. \tag{3.14}$$

with $\eta_w \in \mathbb{R}_+$. Compared to the old switching rules, two conditions for the foraging states are added, these work as follows: A beacon agent can only switch to a foraging state if there are no foraging agents within its region of influence and its summed stored weight values are smaller than some threshold value η_w.

After a beacon agent switches to a foraging state, it will always perform a random step, as by construction there is no other beacon within the agents range. Whether or not the agent will end up in the region of influence of another beacon after the first step, is related to the number of former neighbouring beacons. If the former beacon was fully surrounded by beacons, the agent will always end up in the region of influence of a beacon after one step. Else, there is a chance of ending up in a region not covered by a beacon after the first step. To prevent an agent from switching directly back to the beacon state in such case, we restrict the switching from forager state to beacon state such that a forager agent needs to be for at least 2 time steps in a foraging state before it can switch back to the beacon state.

Hereby increasing the change that a former beacon reaches the region of influence of a beacon, and potentially decreasing the number of state-switches.

At last, we force an agent to 'forget' the weight and guiding velocity values after switching from the beacon state to a forager state, i.e. ω_a^S and v_a^S are set to zero after a state transition. Since the information stored at a beacon is location depended, preserving the former beacon information will only disturb the system if a former beacon agent transforms to a beacon again at another location.

3.2.2. Moving Beacons

In order to optimize the beacon infrastructure we enable the beacons to move. We first derive a control law for the beacon movement, after which we need to update the state transition rules.

Beacon Dynamics

Let us first analyze the role of a beacon in guiding the foraging agents: A beacon stores weights and guiding velocities for both state F_1 and F_2. These guiding velocities provide the direction to follow in the region of influence of a beacon, or in other words: The guiding velocity of state F_1 captures a part of the path from $\mathcal{S}$ to $\mathcal{T}$.

Take $\theta_b \in (0, \pi)$ the smallest angle between the guiding velocities, $v_b^{F_1}$ and $v_b^{F_2}$, stored at some beacon agent. Suppose that we deploy the swarm in an environment without obstacles. The optimal path is a straight path between region $\mathcal{S}$ and $\mathcal{T}$. Or, from a single beacon point of view: if a beacon is located close to the optimal path, the stored guiding velocities will point in opposite direction, i.e. $\theta_b = \pi$.

Now consider the case of an environment with obstacles that blockade the straight path between $\mathcal{S}$ and $\mathcal{T}$. As we assume the agents to be able to move in any direction, the optimal path always consists of straight lines connected by nods located at the surface of an obstacle. If the region of influence of a beacon only captures a straight part of the optimal path, the same statement regarding θ_b as for the no-obstacle environment holds. However, if a beacon is located close to a nod, it is hard to make any statement on the optimal θ_b: Close to the nod, a beacon close to the optimal path can have a smaller θ_b than a beacon further from the optimal path, but closer to the obstacle.

Given a beacons limited knowledge of the environment, it is hard to control the preferred behavior close to nods. We therefore neglect the optimality close to nods, and apply a simple control law for the beacons velocity:

$$v_b(k) = \begin{cases} 0 & \text{if} \quad \mathcal{D}_b(k) \cap \mathcal{S} \neq \emptyset \vee \mathcal{D}_b(k) \cap \mathcal{T} \neq \emptyset \\ v_{b,0}\left(|v_b^{F_1}(k)| + |v_b^{F_2}(k)|\right) & \text{else} \end{cases} \tag{3.15}$$

with $v_{b,0} \in \mathbb{R}_+, v_{b,0} << v_0$ the beacons speed. This simple P-controller works as follows: The summed guiding velocities vector points in the direction in which the beacon should theoretically move to increase θ_b. The size of the summed vector scales by the inverse of θ_b, i.e. for small values of θ_b the beacon will move faster than for large values of θ_b. In addition, the beacon will remain static if it can not move due to collision avoidance. In our experiments we experienced the need for beacons surrounding the $\mathcal{S}$ and $\mathcal{T}$ regions, therefore we force beacons δ close to these regions to remain static. Although we assume that agents can only detect the target regions if they are located within these regions (Assumption 1), measuring if a beacon is δ close to a region can easily be implemented by detecting if a beacon receives rewards by foragers within its region of influence.

Following the stigmergy-principle, the stored weights and guiding velocities depend on the location of the beacons. Moving the beacon does conflict this connection. Therefore we will choose the speed of the beacon much smaller than the speed of the agents, such that the stored information can adjust to the changing location of the beacon, without destroying the fields of navigational information.

State Transitions
Moving the beacons can conflict the condition to remain in the beacon state, because possibly other beacons will enter the region of influence of a beacon, conflicting the beacon state condition: $\nexists b \in \mathcal{B}(k) : x_b(k) \in \mathcal{D}_a(k)$ (see Equation (3.4)). Therefore we will have to adjust the state transition rules. Since we want to realize a minimalistic beacon infrastructure, each region should be covered by maximal one beacon. If beacons are within each others region of influence, we only want to preserve the most valuable beacon for the navigation infrastructure, or in other words, we want to preserve the beacon with the highest stored weight values.

Furthermore, to prevent that beacons far away from the optimal area first have to move towards this area before getting removed, the *Moving Beacons*-extension is always implemented in combination with the *Beacon-Forager Switching*-extension.

Then, the state transition rules of the *Beacon-Forager Switching*-extension (see Equation 3.13) in combination with the preferred beacon-state transition conditions, is given by:

$$s_a(k+1) = \begin{cases} "B" & \text{if} \quad s_a(k-2) \in \{"F_1","F_2"\} \wedge \mathcal{B}_a(k) = \emptyset \text{ or} \\ & \quad\; s_a(k-2) \in \{"F_1","F_2"\} \wedge \forall b \in \mathcal{B}_a(k) : \sum_s w_b^s(k) < \sum_s w_a^s(k) \\ "F_1" & \text{if} \quad s_a(k) = "F_2" \;\wedge\; x_a(k) \in \mathcal{S} \text{ or} \\ & \quad\; s_a(k) = "B" \;\wedge\; s^-(k) = F_1 \;\wedge\; \sum_s w_a^s(k) < \eta_w \;\wedge \mathcal{F}_a = \emptyset \text{ or} \\ & \quad\; s_a(k) = "B" \;\wedge\; s^-(k) = F_1 \;\wedge\; \exists b \in \mathcal{B}_a(k) : \sum_s w_b^s(k) > \sum_s w_a^s(k) \quad \forall a \in \mathcal{A}. \\ "F_2" & \text{if} \quad s_a(k) = "F_1" \;\wedge\; x_a(k) \in \mathcal{T} \text{ or} \\ & \quad\; s_a(k) = "B" \;\wedge\; s^-(k) = F_2 \;\wedge\; \sum_s w_a^s(k) < \eta_w \;\wedge\; \mathcal{F}_a = \emptyset \text{ or} \\ & \quad\; s_a(k) = "B" \;\wedge\; s^-(k) = F_2 \;\wedge\; \exists b \in \mathcal{B}_a(k) : \sum_s w_b^s(k) > \sum_s w_a^s(k) \\ s_a(k) & \text{else} \end{cases} \tag{3.16}$$

with $\eta_w \in \mathbb{R}_+$ the threshold value. The conditions for state $"B"$ are adjusted such that other beacons are allowed within the region of influence of a beacon only if this beacon stores a higher amount of summed weights than the other beacons within its region of influence (). The conditions for states F_1 and F_2 are adjusted such that a former agent in beacon state will change to a foraging state if a beacon is located within its region of influence and this beacon stores a higher amount of summed weights. Again we force an agent to 'forget' the weight and guiding velocity values after switching from the beacon state to a forager state, i.e. after a transition ω_a^s and v_a^s are reset at zero.

3.2.3. Double Updating

The *Double Updating*-extension is inspired on [36] who realize faster convergence for the weights in a similar two-pheromone-system by additionally updating the to-follow pheromone. This theoretical improvement is made possible by the symmetry of the environment in the foraging problem. The authors discretized the environment and simulate the dynamics of the agents over a graph. For a symmetric graph the proability of transitioning from node i to node j is equal to the probability of transitioning from node j to i. The question is if this principle is also applicable for our self-guiding swarm design.

The *Double Updating*-extension, in which foragers reinforce not only the fields corresponding to their internal state, but also the fields they are following, is implemented by updating the beacon weights and guiding velocities not only using the set of foraging agents in corresponding state ($\mathcal{F}_b^s$), but using all foragers in the region of influence ($\mathcal{F}_b$). Equation (3.8) and (3.9) change to:

$$\omega_b^s(k+1) = (1-\rho)\omega_b^s(k) + \rho \frac{\sum_{f \in \mathcal{F}_b(k)} \Delta_f^s(k)}{|\mathcal{F}_b(k)|}. \tag{3.17}$$

$$v_b^s(k+1) = (1-\rho)v_b^s(k) + \rho \frac{\sum_{f \in \mathcal{F}_b(k)} -v_f(k)}{|\mathcal{F}_b(k)|}. \tag{3.18}$$

3.3. Concept Intuition

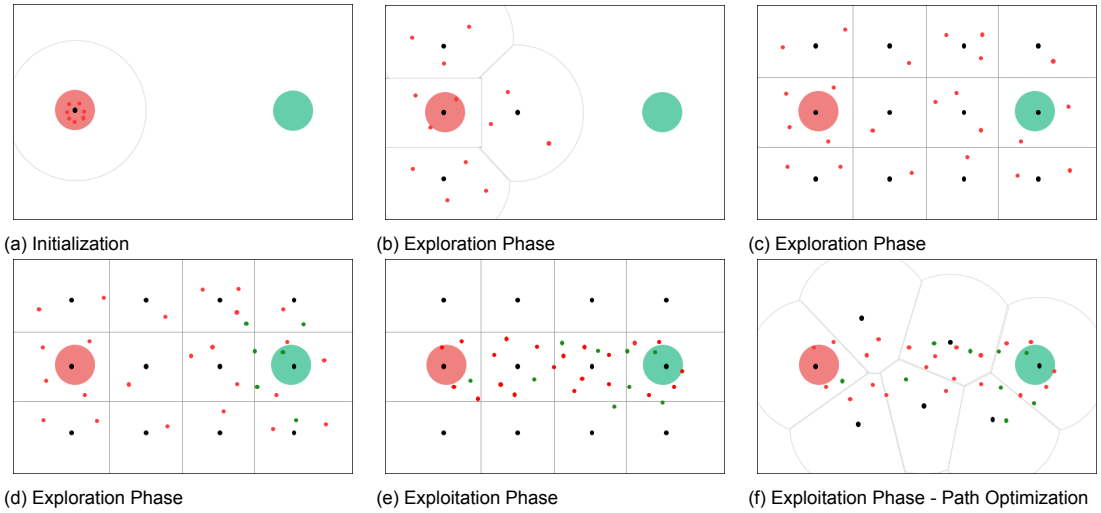
In the previous subsections we presented the mathematical frameworks of the self-guiding swarm and its extensions. In this section we will describe the proposed system in a more intuitive way, by visualizations and analogies between our concept and swarms of Social Insects.

We first describe the conceptual swarm behaviour, next the

3.3.1. Conceptual Swarm Behaviour

Figure 3.1 illustrates the main idea behind the self-guiding swarm. All agents of the homogeneous swarm start at region $\mathcal{S}$ in state F_1, i.e. all agents start looking for the $\mathcal{T}$ region, except for one agent that starts in state B and forms the first beacon of our navigation infrastructure (see Figure 3.1a). The agents randomly explore the environment while creating an infrastructure of beacons and storing local information regarding navigation to $\mathcal{S}$ at the beacons (see Figures 3.1b and 3.1c). We will refer to this phase as the *exploration*-phase.

Once an agent reaches region $\mathcal{T}$, its internal state switches to F_2 and the agent tries to return to region $\mathcal{S}$ using the information stored in the beacons, while storing local information regarding navigation to region $\mathcal{T}$ at the beacons. Agents in state F_1 can now start using the navigation information stored at the beacons to find region $\mathcal{T}$ (see Figure 3.1d). After some time the local information will converge such that the infrastructure stores the optimal path. All foraging agents will now travel between $\mathcal{S}$ and $\mathcal{T}$ using the shortest path (see Figure 3.1e). After the swarm has established a stable path, we say the swarm entered the *exploitation* phase, named after the foraging-ant-phenomenon in which ants en masse start to exploit the food once a path is created. In the exploitation phase only a part of the beacon infrastructure will be used. Ideally the beacon infrastructure is optimized, such that the least amount of agents create the beacon infrastructure storing the shortest path (see Figure 3.1f).



(a) Initialization (b) Exploration Phase (c) Exploration Phase

(d) Exploration Phase (e) Exploitation Phase (f) Exploitation Phase - Path Optimization

Figure 3.1: Illustration of the conceptual dynamics behaviour of the self-guiding swarm, where: the green and red marked region represent $\mathcal{S}$ and $\mathcal{T}$, respectively; the red, green and black dots are agents in state F_1, F_2 and B, respectively; and the cells are the Voronoi partitioning of the beacons bounded by a maximum range of δ.

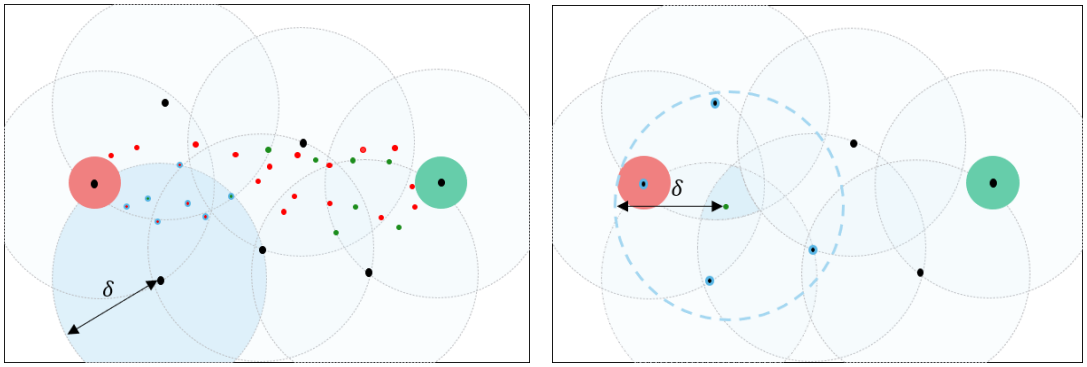
3.3.2. Discretization of the Environment by Beacons

Inspired by ants foraging in nature, our concept is build indirect communication via stigmergy, local storage of information. As discussed in the Section 2.3, continuous marking of the environment, like the ants do, is in practice hard to achieve. Hence, we choose to discretize the environment into small regions and store the local information of a region in a beacon. For the foraging problem, and in general for navigational tasks over discretized environments, it holds that the quality of the solutions improve for finer discretization. However, the efficiency of an autonomous beacon solution would decrease, as the relative amount of agents able to travel between the target regions decreases.

We tried to refine the discretization grid in a smart way, using overlapping regions of influence and different a different discretization for updating and applying local information. Figures 3.2a and 3.3b illustrate the region of influence of a beacon and foraging agent, respectively.

A beacon agent is updated by all the foraging agents within its region of influence (see equations 3.8 and 3.9). Or from the forager agent's perspective: a foraging agents updates all the beacon agents within its region of influence. The update of an beacon by a foraging agent depends on whether or not the foraging agent is located in the rewarding regions, and on the information stored in the neighbouring beacons of the foraging agent. As a result of this mechanism, every region covered by an unique combination of beacons, results in an unique update of the covering beacon agents. Figure 3.3b highlights such an unique area. The crux of our method is that a foraging agent computes the guiding velocity using the averaged guiding velocities stored at its neighbouring beacon agents (see equation (3.10)). In other words, every unique region stores unique local navigation information.

Obviously this does not imply that we increase the refinement of the discretization without increasing the number of beacons. The unique regions are connected: The information stored in an unique area is influenced by updates from other unique regions. Hence the unique regions are not a perfect discretization of the environment. Note however that primarily during the exploitation phase the small discretization is required to optimize the created paths. In this phase the agents will most likely be concentrated on a few unique areas per beacon, such that the information stored per unique area is mainly influenced by updates from this area.



(a) Region of influence of a beacon agent. The blue colored dashed circles represent the region of influence per beacon. The foraging agents located within the region of influence of the lower-left located beacon are marked dark blue.

(b) Region of influence of a foraging agent. The blue colored dashed circle represent the region of influence of the foraging agent. The beacons within the region of influence of the foraging agent are marked dark blue. The unique area the foraging agent is located in is highlighted in light blue.

Figure 3.2: Discretization of the environments by the regions of influence of agents

3.3.3. Local Navigation Information: Weights & Guiding Velocities

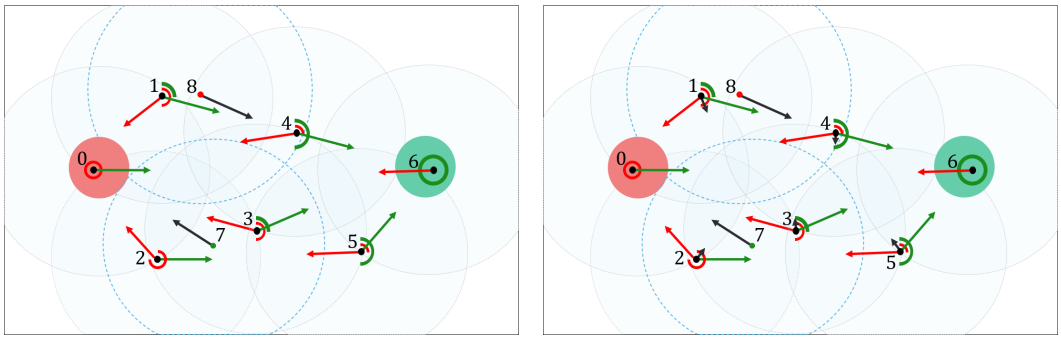
Next, let us discuss the information stored at the beacons. The continuous dropping of pheromones by ants creates a continuous pheromone field. The ants employ the gradient of this field to navigate. Given a field of beacons storing weights (imitating pheromones), one could simply enable the agents to detect the location of the beacons and make them to move in the direction of the beacon storing the highest weight value. Or, one could enable the beacons to broadcast some signal decaying in intensity over traveled distance, in this way imitating a continuous pheromone field, and enabling foraging agents to approximate the gradient of this field and move accordingly. All these kind of methods would require advanced capabilities of the agents: detection of direction of a signal; measuring the intensity of a signal; and broadcasting distance-intensity-decreasing signals.

For the agents in our self-guided swarm, we assume none of these advanced capabilities. We only assume the ability of an agent to measure its global orientation (a compass). Using only the agents knowledge of its global orientation we want to store in each region the global direction an agent should move, so called guiding velocities. To achieve this, we use the symmetry of our system: Agents update

the local information regarding navigation to $\mathcal{S}$ while travelling to $\mathcal{T}$, and vice-versa. If we reasonably assume that the direction towards $\mathcal{S}$ is opposite to the direction to $\mathcal{T}$, we can use the opposite direction of heading of an agent in state F_1 to update the guiding velocities corresponding to state F_1. Remark that this method relies on the fact that agents start to explore the environment from region $\mathcal{S}$ or $\mathcal{T}$. In the random exploration phase the agents will move outwards the initialization region, in this way initializing guiding velocities pointing towards the initialization region.

Figure 3.3a illustrates the proposed dynamics. Foraging agent 8 (in state F_1) is heading in the direction computed by summing the green guiding velocities stored at its neighbouring beacons (agents 1 and 4) weighted by their corresponding weight value. While looking for region $\mathcal{T}$, agent 8 updates the guiding velocities corresponding to state F_1 of its neighbouring beacons by its opposite heading direction, indicating the direction to region $\mathcal{S}$.

One question we hope to answer in this thesis, is how the guiding velocity behave with respect to the weights field. Will the guiding velocities behave as the gradients of the weights field? Or is the guiding velocity field better interpreted as a map of paths, updated by the weights stored on the paths. In other words, will the weights dictate the dynamics of the guiding velocities, do the weight dynamics follow the guiding velocities, or do they adapt simultaneously?



(a) Basic Concept

(b) Basic Concept + *Moving Beacons*-extension

Figure 3.3: illustration of the weight and guiding velocities field, and the dynamics of the self-guiding swarm. Red coloured symbols correspond to state F_1 and green coloured symbols correspond to state F_2. Regions $\mathcal{S}$ and $\mathcal{T}$ are marked red and green, respectively. Red, green and black dots correspond to agents in state F_1, F_2 and B, respectively. The arrows represent the (normalized) guiding velocities stored at a beacon. The black arrow point in the heading direction of an agent. The full, three-quarter, half and quarter circles indicate the weights stored at a beacon. The dashed circles represent the region of influence per agent.

3.3.4. Extensions

In order to optimize the beacon infrastructure we propose the *Beacon-Forager Switching*- and *Moving Beacons*-extension. Figure 3.3b provides an illustration of the *Moving Beacons*-extension in which the beacons move in the direction of the summed guiding velocity weights. Note that this illustration shows the normalized guiding velocities, in reality the beacon speed will be much smaller than the speed of the foraging agents.

3.3.5. Abstraction Methods

To analyze formal properties of a stigmergic foraging systems, it is most common to describe the dynamics using graphs. Pheromones are translated as weights assigned to the nodes or edges, and the relative weight of an edge or the difference in weights between nodes provides the transition probabilities of moving from one location to another.

In our case, abstracting the system as a graph using a fixed discretization of the environment is complex, since the beacon configuration is randomly created. Other options, such as choosing the beacons, or the intersections of the regions of influence, as nodes, do not seem to simplify things either, since this framework of nodes can not directly be connected to the movements of the foraging agents.

The most suitable abstraction method seems to be a graph in which each uniquely covered region is defined as node. Observe that each unique region is at least connected to its neighbouring unique regions. The transition probabilities between connected nodes, can be derived using the unique guiding velocity stored in a region, a node. However, some complexities arise. First of all, to which neighbouring region a guiding velocity will guide an agent within one step, depends on where an agents location within this unique region. Additionally, the guiding velocity can even guide the agent to non-neighbouring region, skipping over the neighbouring region. Moreover, the weights and guiding velocities related to a node are affected by agents at (even non-) neighbouring nodes, because neighbouring areas share the same beacons. In the next Chapter we will further emphasize on the difficulties of deriving formal guarantees or conditions on the behavior of the self-guiding swarm.

4

Results and Guarantees

In this Chapter we will try to derive formal guarantees on the behaviour of the proposed (basic) self-guided swarm given the problem description, the extensions to the basic self-guided swarm design are disregarded.

In Section 4.1 we will assume that the probability distribution of the random noise added to an agents movement ($p(\alpha_u)$, see Equation (3.11)) is uniformly distributed (for $\alpha_u \in (-\pi, \pi)$), such that the agents, while exploring or moving random in exploitation phase, in fact perform a random walk. Moreover we will assume the domain $\mathcal{D}$ to be a grid polyon. These assumptions allows us to apply the results of Wagner et al. [91] (see Section 2.7.1) and conclude on the expected cover time, the variance of the cover time and the commute time for travelling back and forth between two regions.

In Section 4.2 we show another approach in which we only assume the probability distribution of the random noise added to an agents movement ($p(\alpha_u)$, see Equation (3.11)) to be non-zero (for $\alpha_u \in (-\pi, \pi)$). We will show that under these assumptions for some non-zero probability agents will reach every region of the domain.

4.1. Random Walk Exploration

For our analysis of the cover and commute time we will use the formal results of [91] who show that the expected cover time of an unknown planar domain is proportional to the electrical resistance of the domain. The approach presented in Wagner et al. [91] relies on the fact that at every step an agent chooses a random next location from a circle around its current location. In this Section we will show that under certain assumption the formal results for the PC-algorithm can be directly applied to the exploration dynamics of the self-guiding swarm. Let us first state the assumptions for the following theoretical results.

Assumption 2.

1. *Each agent coves a compact disk of radius $r \in \mathbb{R}_+$ centered at the agents location. The radius is chose such that: $r > v_0\tau$.*

2. *Domain $\mathcal{D}$ is equal to a μ-grid polygon of size n, i.e. a polygon made of a connected set of n squares of size μ on the grid. The size of the squares is chosen such that: $d < r$. Two squares on the grid are considered connected if they have a common edge.*

3. *The probability distribution of the random noise added to an agents movement is assumed to be uniform distributed, i.e. $p(\alpha_u) = \frac{1}{2\pi}$ for $\alpha_u \in (-\pi, \pi)$ in Equation* (3.11)*.*

4. *An agent only performs obstacle-robot avoidance, agent-agent collisions are neglected, i.e. agents are assumed to be particles*

5. *If during a step collision avoidance is applied, an agent will always end up at least* $\frac{1}{8}v_0\tau$ *distance away from its previous location. In other words we assume: for* $a \in \mathcal{F}(k)$ *it holds that* $\|x_a(k) - x_a(k+1)\|_2 > \frac{1}{8}v_0\tau$, $\forall k$. *Given the assumptions on* $\mathcal{D}$, *this can, for example, be realised by continuing the path after collision parallel to the domains boundary*

Remark 1. *Visiting all the* n*-grid squares is sufficient to guarantee a full coverage of* $\mathcal{D}$.

Note that Remark 1 is a result of our choice of an agents cover radius r and the grid square size d for which holds that $d < r$. If an agent is located anywhere within a grid square, the whole square is covered (actually, some of the neighbor squares are also partially covered, but this does not make any harm to our upper cover time bound result).

First of all, we can first derive some general results on the minimum number of steps required to cover $\mathcal{D}$.

Corollary 2. *If* R *is an* d*-grid polygon of size* n, *then at last* $\left\lceil \frac{6d^2n}{(4\pi+3\sqrt{3})r^2} \right\rceil$ *steps of an agent with covering radius* r *are necessary to cover it*

Proof. We can use Lemma 1 and take $A = nd^2$ and $a = r^2\pi$. □

Let us now look at the similarities of the PC-process and the exploration dynamics of the self-guided swarm under Assumptions 2. For our analysis we set the maximum step size of the PC-process (h_0) equal to the distance traveled per time step by foraging agents in the self-guided swarm: $h_0 = v_0\tau$. Furthermore, take $\mathcal{R}_r(x)$ as a disk of radius r around x: $\mathcal{R}_r(x) \coloneqq \{x' : \|x' - x\|_2 \leq r\}$

First, one can verify that if an agent is located at $x_k \in \mathcal{D}$ such that $\mathcal{R}_r(x_k) \subset \mathcal{D}$, the dynamics as implied by the PC-Process (See Algorithm 1) are equal to the dynamics of a foraging agent of the self-guided swarm picking a random direction ($\epsilon = 1$, Equation (3.11)). In other words, for situations in which for both the PC-process and the self-guided swarm no obstacle avoidance is performed (i.e. $\mathcal{R}_r(x_k) \subset \mathcal{D}$) the exploration dynamics of the self-guided swarm under Assumptions 2 and the PC-Process, both result in fact in agents performing a random walk.

Next, let us look at the situations in which one of the processes does perform collision avoidance. For the PC-Process, if an agent is closer than $2v_0\tau$ to the edge of $\mathcal{D}$, its step-size will decrease to preserve the basic principle of the PC-Process: an agent chooses its next location uniformly from a circle around its current location. Taking a range of $2v_0\tau$ avoids the change of the robot going to $\partial\mathcal{D}$. For the self-guided swarm we applied a fixed step in combination with a continuous collision avoidance algorithm that prevents an agent from reaching $\partial\mathcal{D}$ by changing the heading direction (using some policy satisfying Assumption 2). In the remainder of this section, we will refer to the process of performing a sequence of random steps in the self-guided swarm under Assumptions 2 as the 'fixed step'-approach.

It can be shown that the 'fixed step'-approach always covers as least as much surface of the region $\mathcal{D}$ at two successive time steps as the PC-process. Denote the step size of the PC-process by $h_{PC}(x) \in \mathbb{R}_+$, which is given by,

$$h_{PC}(x_k) = \min\{v_0\tau, \max_{\mathcal{R}_{2h'}(x_k)\subset\mathcal{D}}\{h'\}\} \tag{4.1}$$

The maximum fixed step size of the 'fixed step'-approach, without interruption of the collision avoidance algorithm is denoted by $h_{fix}(x) \in \mathbb{R}_+$ and is equal to,

$$\begin{aligned} h_{fix}(x_k) &= \min\{v_0\tau, \max_{\mathcal{R}_{h'}(x_k)\subset\mathcal{D}}\{h'\}\} \\ &= \min\{v_0\tau, 2h_{PC}(x_k)\} \end{aligned} \tag{4.2}$$

In addition, define $R_{PC}(x_k)$ and $R_{fix}(x_k)$ as the set of points that are with non-zero probability reachable from point x_k using the PC-process and fixed step-approach, respectively. By definition $R_{PC}(x_k)$ is always a circle of radius $h_{PC}(x_k)$. If no collision avoidance is performed for the 'fixed step'-approach, i.e. $h_{fix}(x_k) = v_0\tau$, then $R_{fix}(x_k)$ is also a circle of radius $v_0\tau$. At last, denote by x_{k+1}^{PC} and x_{k+1}^{fix} a sample from $R_{PC}(x_k)$ and $R_{fix}(x_k)$, respectively.

In general it holds that the larger the distance between the centers of two overlapping disks the larger the area covered by these two disks (see Proof Lemma 1 of [91]). So, to prove that two successive steps performed by the 'fixed step'-approach result in a larger covered area than two successive steps performed by the PC-process, we need to guarantee that

$$\|x_{k+1}^{PC} - x_k^{PC}\|_2 \leq \|x_{k+1}^{fix} - x_k^{fix}\|_2 \tag{4.3}$$

Or in other words, since the 'fixed step'-algorithm always travels an absolute distance of $v_0\tau$, we need to guarantee that the distance traveled after collision does not end up h_{PC} close to x_k. The distance after collision is equal to $v_0\tau - h_{fix}$. The maximum distance from the edge of $\mathcal{D}$ at which x_{k+1}^{fix} can end up h_{PC} close to x_k, is if an agent is $\frac{1}{2}v_0\tau$ close to the edge of $\mathcal{D}$, such that $h_{PC} = \frac{1}{4}v_0\tau$. The closer we get to the edge of $\mathcal{D}$, the smaller h_{PC} and the larger $v_0\tau - h_{fix}$ will become. Since, for the 'fixed step' approach we assumed an agent to end up at time $k+1$ at least a distance $\frac{1}{4}v_0\tau$ away x_k, the constrained given by Equation (4.3) is by assumption always guaranteed.

We showed that under Assumption 2 it is guaranteed that the area covered by the disks $\mathcal{R}_{fix}(x_k) \cup \mathcal{R}_{fix}(x_k)$ is equal or lager than the area covered by $\mathcal{R}_{PC}(x_k) \cup \mathcal{R}_{PC}(x_k)$. In other words, an agent stepping according to the fixed step approach will cover as least as much as stepping according to the PC-process. Therefore we conclude that the formal results as derived by Wagner et al. [91] and discussed in Section 2.7.1, can be applied for the fixed size approach, i.e. our self-guided swarm under Assumptions 2.

4.2. General Autoregressive Exploration

In the previous section we made some impactful assumptions to derive formal results for the exploration behaviour of the self-guided swarm: We only considered environments that can be represented as grid-polygons and restricted $p(\alpha_u)$ to have an uniform distribution. In this section we will try to find more general formal results regarding the exploration of the domain under less strict assumptions. For this, we take inspiration from the results in RRT exploration [42, 46, 47, 47] (see Section 2.8.1). This offline path planning tool for continuous (non-convex) domains uses a similar fixed step length exploration approach as the self-guided swarm. In fact the tree constructed in the RRT algorithm shows many similarities with the beacon field created by our self-guided-swarm.

Let us first state the assumptions for the following theoretical results.

Assumption 3.

1. *The probability distribution of the random noise added to an agents movement is assumed to be non-zero, i.e. for $p(\alpha_u) > 0$ for $\alpha_u \in (-\pi, \pi)$.*
2. *The inter-sampling time is always chosen such that: $\tau < \frac{\delta}{v_0}$. Moreover, τ can be chosen small enough in comparison to the diameter of the domain $\mathcal{D}$.*
3. *The regions $\mathcal{S}$ and $\mathcal{T}$ are compact discs of radius, at least, τv_0.*

Remark 2. *Any forager a is always in the region of influence of a beacon. In other words, $\exists b \in \mathcal{B}(k)$: $x_a(k) \in \mathcal{D}_b(k)$ $\forall k$, $\forall a \in \mathcal{F}(k)$.*

Observe Remark 2 holds by construction. From the transition rule in (3.4) it follows that, whenever $x_a(t) \notin \mathcal{D}_b(t)$ for any beacon $b \in \mathcal{B}(t)$, it becomes a beacon, therefore covering a new sub-region of the space.

Proposition 1. *Let $\mathcal{D}$ be convex. Let some $a \in \mathcal{F}(k_0)$ have $x_a(k_0) = x_0$. Then, for any convex region $\mathcal{D}_i \subset \mathcal{D}$ of non-zero volume, there exists $\kappa_i \in \mathbb{R}_+$ and finite time $k_i \in \mathbb{N}$ such that*

$$\Pr[x_a(k_0 + k_i) \in \mathcal{D}_i \mid x_0] \geq \varepsilon^{k_i}\kappa_i.$$

Proof. To prove the statement, we will show that the ball of reachable points includes the entire domain $\mathcal{D}$ for any agent and large enough times.

We can consider from Remark 2 that a forager is always in the region of influence of at least one beacon, and therefore will remain in foraging state. At every time-step foragers with some probability ϵ chooses a random heading direction: $\alpha_a(k+1) = \alpha_a(k) + \alpha_u$, with α_u taking values $(-\pi, \pi)$, following some non-zero density function $p(\alpha_u)$. It holds that $\alpha_a(k+1) \in (-\pi, \pi)$. Take $p_a(x, k|x_0)$ as the probability density of forager a being at point x at time k with $x_a(k_0) = x_0$, and $\mathcal{X}(k)$ as the set of points x forager a can be located at time k with $p_a(x, k|x_0) > 0$ probability. One can show that $\mathcal{X}(k_0 + 1)$ forms a circle of radius $v_0\tau$ in $\mathbb{R}^2$ around x_0:

$$\mathcal{X}_a(k_0 + 1) = \{x \in \mathcal{D} : \|x - x_0\|_2 = v_0\tau\},$$

and for $k = k_0 + 2$ the set $\mathcal{X}(k_0 + 2)$ forms a disk of radius $2v_0\tau$ around x_0,

$$\begin{aligned}\mathcal{X}_a(k_0 + 2) &= \{x \in \mathcal{D} : \|x - x_1\|_2 = v_0\tau,\ x_1 \in \mathcal{X}_a(k_0 + 1)\} \\ &= \{x \in \mathcal{D} : \|x - x_0\|_2 = 2v_0\tau\} \qquad (4.4)\end{aligned}$$

.

One can verify that $\mathcal{X}(k_0 + 2)$ forms a disk of radius $2v_0\tau$ around x_0 as follows: Take a point x_2 in $\mathcal{X}_a(k_0 + 2)$, then one can always find a point $x_1 \in \mathcal{X}(k_0 + 1)$ such that (x_0, x_1, x_2) forms a triangle where $\|x_0 - x_1\|_2 = v_0\tau$ and $\|x_1 - x_2\|_2 = v_0\tau$. By construction, for any point x in $\mathcal{X}(k_0 + 2)$ it holds that $p_a(x, k_0 + 2|x_0) > 0$, so for a subset $\mathcal{D}_2 \subseteq \mathcal{X}(k_0 + 2)$,

$$\Pr[x_a(k_0 + 2) \in \mathcal{D}_2 \mid x_0] \geq \varepsilon^2 \int_{\mathcal{D}_2} p_a(x, k_0 + 2|x_0)dx \geq \varepsilon^2\kappa_2,$$

where $\kappa_2 \in \mathbb{R}_+$ is a function of the set $\mathcal{D}_2$ and the probability density $p_a(x, k_0 + 2|x_0)$. Remark that the first inequality is due to the fact that, by choosing a non-random velocity, the agent could still end up at $\mathcal{D}_2$. One can now see how for $k \geq 2$ the sets $\mathcal{X}_a(k_0 + k)$ are balls centred at x_0 and radius $kv_0\tau$. Let $\mathcal{D}_i$ be any subset of $\mathcal{D}$ with non-zero volume and take $k_i = \min\{k : \mathcal{D}_i \subset \mathcal{X}_a(k)\}$. Then, $\Pr[x_a(k_0 + k_i) \in \mathcal{D}_i \mid x_0] \geq \varepsilon^i\kappa_i$ for some $\kappa_i > 0$. □

Taking inspiration from [42], we can draw similar conclusions regarding the exploration for the case that $\mathcal{D}$ is non-convex.

Lemma 3. *Let $\mathcal{D}$ be a non-convex connected domain. Let some $a \in \mathcal{F}(k_0)$ have $x_a(k_0) = x_0$. Then, for any convex region $\mathcal{D}_i \subset \mathcal{D}$ of non-zero volume, there exists $\tau > 0$ and $\kappa_i > 0$ such that we can find a finite horizon $k_i \in \mathbb{N}$,*

$$\Pr[x_a(k_0 + k_i) \in \mathcal{D}_i \mid x_0] \geq \varepsilon^{k_i}\kappa_i.$$

Proof. If $\mathcal{D}$ is connected, then for any two points $x_0, x_{k_i} \in \mathcal{D}$, we can construct a sequence of balls $\{\mathcal{X}_0, \mathcal{X}_1, ..., \mathcal{X}_{k_i}\}$ of radius $R \geq v_0\tau$ centred at $x_0, x_1, ..., x_{k_i}$ such that the intersections $\mathcal{X}_k \cap \mathcal{X}_{k+1} \neq \emptyset$ and are open sets, and $x_k \in \mathcal{X}_{k-1}$. Then, we can pick τ to be small enough such that every ball $\mathcal{X}_k \subset \mathcal{D}$ does not intersect with the boundary of $\mathcal{D}$, and we can apply now Proposition 1 recursively at every ball. If $\|x_k - x_{k-1}\|_2 < 2v_0\tau$, then from Proposition 1 we know $p(x_k, k+2|x_{k-1}) > 0$ since, for a given x_{k-1}, any point x_k has a non-zero probability density in at most 2 steps. Then, it holds that $p(x_{k_i}, k_{i-1} + 2|x_0) > 0$ for $k_{i-1} \in [k_i, 2(k_i - 1)]$, and for a target region $\mathcal{D}_i$: $\Pr[x_a(k_0 + k_i) \in \mathcal{D}_i \mid x_0] \geq \varepsilon^{k_i} \int_{\mathcal{D}_i} p_a(x, k_i|x_0)dx \geq \varepsilon^{k_i}\kappa_i$ for some $\kappa_i > 0$. □

It follows from Lemma 3 that for large enough k_i the probability of an agent at any starting point $x_0 \in \mathcal{D}$ having visited any region $\mathcal{D}_k$ is non-zero, and therefore every forager agent visits every region infinitely often.

For a given initial combination of foraging agents, we have now guarantees that the entire domain will be explored and covered by beacons as $k \to \infty$. We leave for future work the formal guarantees regarding the expected weight field values $\omega_b^s(k)$ and guiding velocities $v_b^s(k)$.

5

Experimental Analysis and Results

In this Chapter we will perform an experimental analysis of the proposed self-guided swarm. First, we introduce our experimental framework: In Section 5.1 we explain the experimental setups and in Section 5.2 we introduce the performance measures used to analyse the foraging performance and quality of the performance. Next, we discuss our experimental results for the following aspect:

- **Parametric Performance**
  In Section 5.3 we analyse the impact of the hyper-parameters, the evaporation rates and exploration rate, on the foraging performance. In addition we analyze the cross-relation between these hyper-parameters, and their relation to the swarm size.

- **Influence of Swarm Size**
  In Section 5.4 we extensively analyze the impact of different swarm size on the swarms foraging performance.

- **Robustness**
  In Section 5.5 we investigate the robustness of the system against measurement and communication noise, and temporal agent failures, and we investigate the effect of limitations on the agents listening capacities.

- **Model Extensions**
  In Section 5.6 we analyze the performance of the proposed *Beacon-Forager Switching*-, *Moving Beacons*- and *Double Updating*-extensions.

We conclude this chapter with a summary of the obtained results in Section 5.7.

5.1. Implementation

The self guided swarm is first implemented by simulating the robots as particles. This requires relative few computational power and allows us the analyze the performance of the swarm for broad ranges of parameters and environmental settings. In the remainder of this chapter we refer to this simulator as the Particle Simulator. The self guided swarm is also implemented in a realistic robot simulator called Webots, to conclude on the actual performance in real-life settings. In this section we explain the algorithms to implement the dynamics of the self guided swarm as covered in Chapter 3, the setup of both simulators, the parameters, and the different environmental settings.

5.1.1. Algorithms

The dynamics of the basic self guided swarm as covered in Section 3.1 is implemented according to Algorithms 4 and 5. Per extension the following adaptions to the standard Algorithms are made:

1. For the *Beacon-Forager Switching*-extension (See Subsection 3.2.1) line 8 of both Algorithms changes to: ”Check transitions in (3.14)”.

2. For the *Moving Beacons*-extension (see Subsection 3.2.2) line 5 of Algorithm 4 changes to: "Move according to $v_b(k+1)$ given by (3.15)" and line 8 of both Algorithms changes to: "Check transitions in (3.16)".

3. For the *Double Updating*-extension (see Subsection 3.2.3) line 4 of Algorithm 4 changes to: "Compute $\omega_b^s(k+1), v_b^s(k+1)$ according to (3.17) and (3.18)".

Algorithm 4: Behaviour of Beacons

```
1 while s_b(k) = 0 do
2     Broadcast ω_b^s(k), v_b^s(k);
3     Listen for signals during τ seconds;
4     Compute ω_b^s(k+1), v_b^s(k+1) according to (3.8) and (3.9);
5     Move according to v_b(k+1) as given by (3.5);
6     if Obstacle then
7         Do not move
8     Check transitions in (3.4);
```

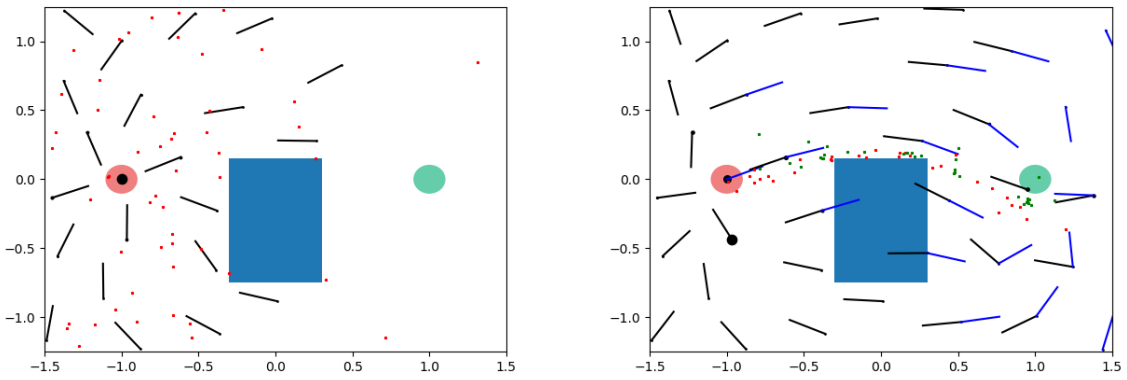
Algorithm 5: Behaviour of Foragers

```
1 while s_f(k) ≠ 0 do
2     Listen for signals during τ seconds;
3     Broadcast v_f(k), Δ_f^s(k);
4     Compute v_f(k+1) from (3.11) and (3.12);
5     Move according to v_f(k+1);
6     if Obstacle then
7         Move to avoid obstacle
8     Check transitions in (3.4);
```

5.1.2. Particle Simulator

In the Particle Simulator, Algorithms 4 and 5 are implemented, modeling the robots as particles in continuous space and discrete time. The time steps are equal to τ. Since particles do not occupy any space, robot-robot collisions do not occur and can be ignored. Robot-obstacle avoidance is implemented using the mirror law: If a particle hits an obstacle, calculate the angle of incidence to the obstacle surface. The particle bounces with the inverse angle of incidence from the surface and continues its path. As in reality the robots are released in batches to prevent overcrowding in the start region, the robots in the particle simulations are also released in batches of 2. Note that we consider our robots to be capable of driving in any direction (holonomic or omin-directional robots), so apart from the environmental constraints there are no constraints on possible movements of the particles. The Particle Simulator is written in Python, the code can be found at *GITHUB - particle_simulator_self_guided_swarm*.

An example of the results of a particle simulation is given by Figure 5.1. The red and green highlighted regions indicate the nest and food location. Black dots are agents acting as beacon. Green and yellow dots are foraging agents looking for the target (in state F_1) and source region (in state F_2). The size of the black dots represents the relative summed weights stored at a beacon. The black and blue lines indicate the guiding velocity vectors corresponding to the nest and food seeking pheromone stored at a beacon. Figure 5.1a shows a swarm in the exploration phase, i.e. the agents are searching for the target region while building the weight and guiding velocity fields corresponding to the nest-seeking pheromone. Figure 5.1b shows the swarm in exploration phase, i.e. the swarm established a path between the source and target region.



(a) Exploration phase

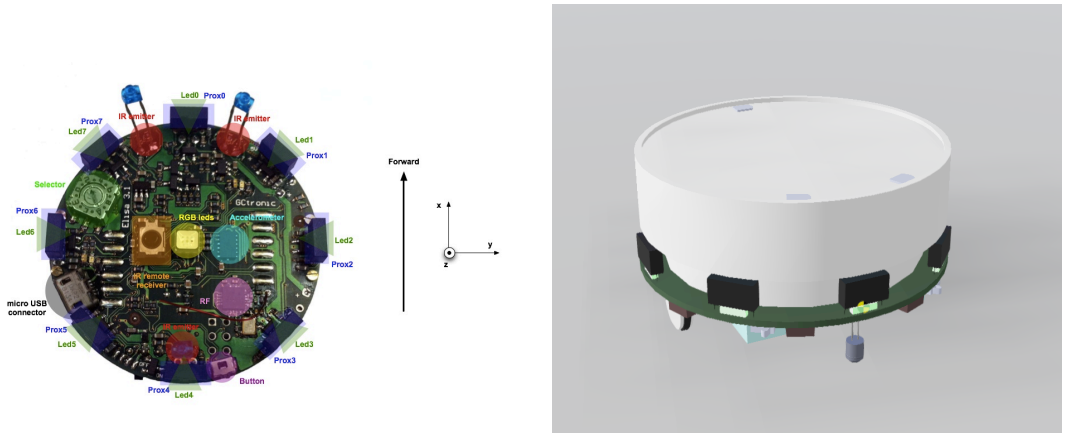
(b) Exploitation phase

Figure 5.1: Snapshots of a particle simulation in which a self-guiding swarm consisting of 101 agents is deployed in the environment with asymmetric obstacle.

5.1.3. Webots Simulator

We use the Webots [59] simulator to implement the self guiding swarm in a realistic setting. The Webots Simulator is able to simulate realistic robots, including the Elisa-3 robot (GCTronic) (See Figure 5.2b), in physical environments. In the future we want to use the Elisa-3 robot to implement our work in real life. The Elisa-3 robot satisfies the restrictions of Assumption 1, expect that the ELisa-3 robot does not possess a global orientation measure (yet). The Webots simulations are performed using time steps of 64 microseconds, resulting in very realistic robotics behavior.

In the Webots simulations we limit the robots to perform only four types of movement: move left or right, or move forward or backward. By limiting the movements of the robots, we preserve the holonomic nature of the Elisa-3, but simplify the (future) controller of the robot. The Elisa-3 robots is equipped with 8 IR proximity sensors (see Figure 5.2a) which are used for collision avoidance. To present our collision avoidance algorithm, define the event of detection of an obstacle by sensor # as *Trigger_Prox_#*. The very simplistic collision avoidance algorithm is given by Algorithm 6. The algorithm comes down to: move away from the direction a object is detected.



(a) Hardware components of an Elisa-3 Robot. The IR proximity sensors are highlighted by blue boxes and tagged as 'Prox#'.

(b) Webots version of Elisa-3 Robot.

Figure 5.2: Simulation of Elisa-3 in Webots

Algorithm 6: Collision Avoidance Elisa-3 Robot

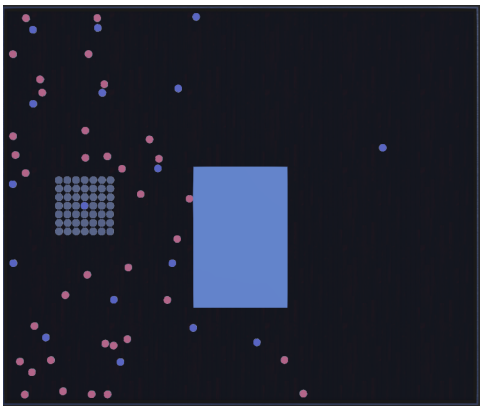
```
if Trigger_Prox_1 then
    Turn 10° left
else if Trigger_Prox_7 then
    Turn 10° right
else if Trigger_Prox_0 then
    Turn 180°
else if Trigger_Prox_5 then
    Turn 10° left
else if Trigger_Prox_3 then
    Turn 10° right
else if Trigger_Prox_4 then
    Turn 180°
```

The agents are able to listen almost-continuously. The incoming signals are stored in a buffer, which is emptied every τ seconds. In Section 5.5 we analyse the performance of the system for a limited size of this buffer. The robots are released in batches of 2 per second to prevent overcrowding. The robots

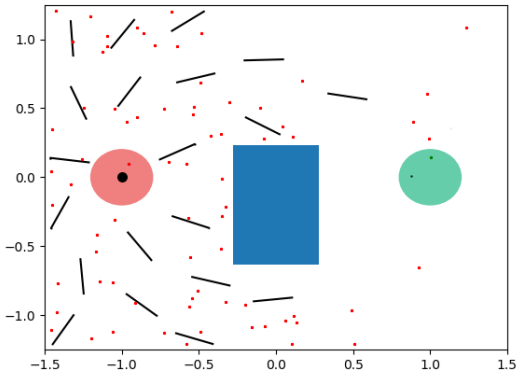
are initially aligned in a square at the nest location and released in bathes of 2 per second to prevent overcrowding at the area around the nest.

The robot controllers are written in C. For the simulations presented in this thesis we used a Supervisor to control the communication between robots and extract measures of the system. The Supervisor is written in Python. The code for the Webots Simulator of the self-guided swarm can be found at *GITHUB - webots_simulator_self_guided_swarm*. The README-file provides information on implementation and usage of the project.

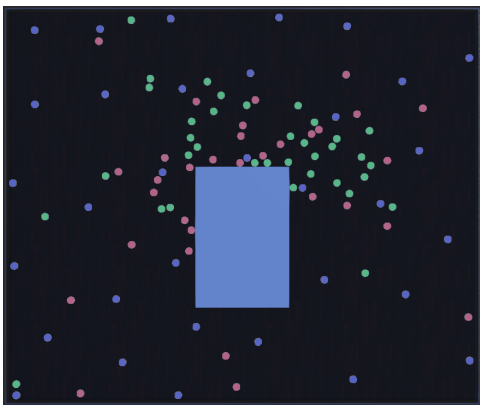
Figures 5.3a and 5.3c show snapshots of a Webots simulation. The blue colored robots are in the beacon state (B), the green colored robots are foragers in 'target region seeking' (F_1) state and the red colored robots are foragers in 'starting region seeking' state (F_2). Grey colored robots are not deployed yet. To analyze the behavior of the swarm we created similar abstraction plots as used to present the results of the the Particle Simulator shown in Figures 5.3b and 5.3d. The colors of the points correspond to the same robot states as used in the Webots simulation; the blue and black lines illustrate the stored guiding velocities ($v_b^s(k)$) corresponding to state F_1 and F_2, respectively, and the size of dots illustrate the relative amount of summed weights ($\omega_b^s(k)$) stored at a beacon agent.



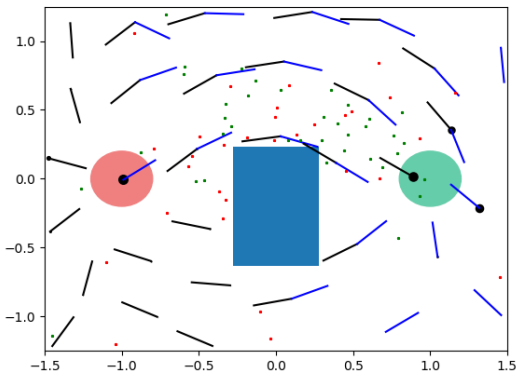
(a) Snapshot exploration phase



(b) Abstraction exploration phase



(c) Snapshot exploitation phase



(d) Abstraction exploitation phase

Figure 5.3: Snapshot and the corresponding abstractions of a Webots simulations in which a self-guiding swarm consisting of 101 robots is deployed in the environment with asymmetric obstacle.

5.1.4. Parameters & Worlds

The default parameters used for the simulations are presented in Table 5.1. We refer to the evaporation rates (ρ_v, ρ_w) and the exploration rate (ϵ) as hyper-parameters. In Section 5.3 we analyse the optimal value of the hyper-parameters for an environment with symmetric obstacle and different swarm sizes. The hyper-parameter values as presented in Table 5.1 are the optimal values as concluded in Section 5.3.

In addition to these parameters, we need to specify the probability density function $p(\alpha_u)$ used by foraging agents to add random noise to their movement if they are exploring or choose to move randomly (see Equation (3.11)). In the experiments we use an uniform distributed $p(\alpha_u)$, i.e. the agents perform a random walk. In addition, we also want to mention the option to bias the forager to its former heading direction, by choosing a normal density function centred at zero and with standard deviation σ_p, i.e. $\alpha_u \sim \mathcal{N}(0, \sigma_p^2)$. The default value of σ_p is given in Table 5.1. The normal distribution is only used for our demonstration results, as referred to in Section 5.7.

Since the optimal characteristics of an robotics swarm is directly related to the type of environment, we use several environments to investigate the performance of the self guided swarm. Table 5.2 provides an overview of the characteristics of the environments used. The location and shape of the square obstacles are given by the coordinates of the vertices. The Webots-world (.wbt) files of the environments can be found in the Github repository *GITHUB - webots_simulator_self_guided_swarm*.

Obviously, the swarm should be able to overcome obstacles. The environments with (non-)symmetric obstacles try to replicate the famous 'double bridge experiment as introduced used by [17, 30] . For the environment with a symmetric placed obstacle it is interesting to analyse if the system converges to one path. For the environment with non-symmetric obstacle it is interesting to show convergence to the optimal (shortest) path.

We want to use as less beacons for our guiding infrastructure as possible. So, it is interesting to analyse if the system is able to converge to one (optimal) path. Since it is inherent harder to converge if two paths are of almost equal length, than when one range of paths is more optimal than the other due the geometry of the environment, as in the non-symmetric case, we mostly use the environment with symmetric obstacle in our analysis.

We also implement a clean environment without any obstacles. Such an environment seems easy to solve, since the swarm only has to create a straight path. However, it can be challenging for the swarm to deal with the 'freedom' this environment provides to create paths. Since the robots are not pushed by obstacles to choose between paths, in theory, there are many close to optimal paths possible.

At last, for demonstration purpose we implement a large environment with multiple obstacles to show the overall capabilities of the system.

ρ_w	ρ_v	λ	r	η_w	ε	$\tau(s)$	$\delta(m)$	$\delta_{s,t}(m)$	$v_0(m/s)$	$v_{b,0}(m/s)$
0.01	0.01	0.8	1	$1e-5$	0.01	1	0.4	0.2	0.25	0.02

Table 5.1: Default parameters for the simulations

Environment Reference	Size	Nest Location	Food Location	Obstacle Coordinates
No Obstacle	$[3, 2.5]$	$(-1, 0)$	$(1, 0)$	
Symmetric obstacle	$[3, 2.5]$	$(-1, 0)$	$(1, 0)$	$(-0.3, -0.45)$, $(-0.3, 0.45)$, $(0.3, 0.45)$, $(0.3, -0.45)$
Asymmetric obstacle	$[3, 2.5]$	$(-1, 0)$	$(1, 0)$	$(-0.3, -0.75)$, $(-0.3, 0.15)$, $(0.3, 0.15)$, $(0.3, -0.75)$
Demonstration	$[10, 8]$	$(-1, 0)$	$(4.5, 5)$	$(-1, 2), (-1, 4), (1, 4), (1, 2)$ & $(3, 0), (3, 2), (5, 2), (5, 0)$

Table 5.2: Characteristics of all the environments used in the simulations. Coordinates and sizes are given in meters.

5.2. Performance Metrics

The goal of the foraging problem is to maximize the number of trips between the target regions. The general measure for the foraging performance is the number of trips with respect to the number of robots used and the running time (per robot) (See e.g. [23]), which we define as the navigation delay. The navigation delay does not provide much inside in the quality of a solution. We try to solve the foraging problem by creating (optimal) trajectories by emergent self-organized behavior. Therefore, the accumulation of agents around trajectories should provide a measure of the quality of a solution. We use entropy to measure the clustering of robots. Moreover, the navigation delay does not provide any insight in the optimal systems behavior. We are in particular interested in the systems ability to optimize the paths. Therefore we also introduce a measure for the shortest time it took agents to complete a number of successive trips.

5.2.1. Average Navigation Delay

To measure the foraging performance, we measure the average navigation delay of the swarm. Navigation delay is defined as the average time it takes an agent to travel between the target regions. To define a trip we use the state transitions ((3.4) or (3.14)), which implies that if an agent in state F_1 reaches region $\mathcal{T}$ it switches to state F_2, and vice versa. Lets define a completed trip at time step k as,

$$\text{trip}_a(k) = \begin{cases} 1 & \text{if } s_a^-(k), s_a(k) \in \{F_1, F_2\} \wedge s_a^-(k) \neq s_a(k) \\ 0 & \text{else} \end{cases} \quad \forall a \in \mathcal{A}. \tag{5.1}$$

Note that this definition of a trip also includes potential trips of agents in beacon state switching back to the forager state in the *Beacon-Forager Switching*-extension, since an agent in beacon state will switch back to its previous foraging state (see (3.14)). Define all trips of a swarm over a finite horizon $t \in [k_0, T]$ as,

$$\text{trips} = \sum_{a \in \mathcal{A}} \sum_{k \in [t_0, T]} \text{trip}_a(k) \tag{5.2}$$

Then, the average navigation delay is given by,

$$d\,(k_0, T) = \frac{|\mathcal{A}|(T - k_0)}{\text{trips}} \tag{5.3}$$

For our analysis we are often only interested in the performance of a swarm in the exploitation phase, i.e. after a path has been created. In this context we say we measure the navigation delay after convergence. The convergence time is roughly defined as the point in time at which at least one agent has travelled a full cycle: Travelled from the nest to the food and back. We always specify the convergence time used for a measure.

In the basic self-guiding swarm method we assume a static infrastructure of beacons. In this case it is interesting to analyze the navigation delay of the foraging agents only. The average navigation delay of the foraging agents is obtained by only measuring the navigation delay for the agents in forager state, i.e. replacing $\mathcal{A}$ with $\mathcal{A}^s$ in Equation (5.2) and (5.3).

To put the foraging performance of the systems in perspective, we derive upper and lower bounds for the navigation delays. As lower bound we take the absolute minimum traveling time, i.e. the time it takes to travel the optimal path at maximum speed. Note that this is an extremely conservative bound as it represents the case in which only a single robot knows and perfectly follows the optimal path. This lower bound does not take potential robot-robot or robot-environment collisions into account, nor the limitations on the robot movements (turn on spot, move in straight line). To obtain the upper bound we run simulations in which all agents start and stay in foraging state moving at all time randomly (move according to Equation 3.11 with ϵ = 1).

5.2.2. Minimum Navigation Delay

To determine the optimally of the created paths, we measure the minimum times it takes an agent to perform n-consecutive trips. Employing that the chance of successive random optimal trips decays to zero when measuring more than one consecutive trips. The set containing the number of steps it took agent a to perform n-consecutive trips is given by,

$$\mathcal{D}_a(n, k_0, T) = \left\{ l \;\middle|\; \sum_{k \in [k_{\text{start}}, k_{\text{start}}+l]} \text{trip}(k) = n,\ k_{start} \in (k_0, T-1),\ l \in (0, T - k_{\text{start}}) \right\} \quad a \in \mathcal{A} \tag{5.4}$$

For our analysis we take as measure of optimality the average of the 10-fastest n-consecutive trips within the finite time interval $[k_0, T]$, which we define as $d_{min}(n, k_0, T) \in \mathbf{R}_+$.

5.2.3. Entropy

We use the entropy to measure the accumulation of agents around trajectories. Entropy has often been used to quantify forms of clustering in robots to investigate if stable self-organization arises (see i.a. [23]). We use the hierarchic social entropy as defined by [3] applying single linkage clustering. In single linkage clustering a robot is assigned to a cluster if the relative distance between one of the robots in the cluster and the robot considered is smaller than h. Define by $\mathcal{C}(t, h)$ the set of clusters at time t with minimum single linkage distance h and by $\mathcal{A}_{c_i}$ the subset of agents in cluster c_i, with $c_i \in \mathcal{C}(t, h)$. The entropy of a set of robots $\mathcal{A}$ is defined as

$$H(\mathcal{A}, h) = -\sum_{c_i \in \mathcal{C}(t,h)} \frac{|\mathcal{A}_{c_i}|}{|\mathcal{A}|} \log_2 \left(\frac{|\mathcal{A}_{c_i}}{|\mathcal{A}|} \right). \tag{5.5}$$

Hierarchic social entropy is then defined as integrating $H(\mathcal{A}, h)$ over all values of h:

$$S(\mathcal{A}, h_{max}) = \int_0^{h_{max}} H(\mathcal{A}, h) dh \tag{5.6}$$

with $h_{max} = \infty$. The absolute lower bound of the entropy measure can be calculated analytically. The robots can not be closer together than the robots diameter ($\delta_r \in \mathbb{R}_+$), so the hierarchic social entropy can never be smaller then,

$$S(\mathcal{A}, \delta_r) = -\log_2 \left(\frac{1}{|\mathcal{A}|} \right) \delta_r \tag{5.7}$$

The theoretical absolute maximum entropy is obtained by distributing the agents uniformly over the environment, i.e. maximizing the average distance between the robots. The maximum entropy can be approximated by taking the maximum entropy of a large number of sets of randomly distributed agents.

For our analysis we are only interested in the difference in entropy of simulations. Therefore, we exclude the minimum entropy from our measure and normalize our measure by the approximated maximum entropy. With $S_{max}(\mathcal{A})$ being the approximated maximum entropy of N agents, the corrected entropy measure is defined as:

$$\bar{S}(\mathcal{A}) = \frac{S(\mathcal{A}, \delta_r)}{S_{max}(\mathcal{A})} \tag{5.8}$$

Since we are only interested in the accumulation of foraging agents around trajectories, in our analysis we measure the entropy over the set of agents $\mathcal{A}^s(k)$.

At last, note that despite the fact that agents in the particle simulator can be closer than δ_r, for the particle simulator too we only integrate over distances greater than δ_r. By taking the exact same corrected entropy measure for both the Particle and Webots Simulator, we can compare the absolute measures, and draw conclusions on the difference in clustering at some point in time.

5.3. Parametric Performance Analysis

In this section the impact of the hyper-parameters, the exploration rate (ε), the weight and directional vector evaporation rates (ρ_w and ρ_v), on the foraging performance of the self guiding swarm is investigated.

5.3.1. Setup

As we expect the three hyper-parameters to be correlated, we first investigate the swarm performance for combinations of three hyper-parameters, while fixing the swarm size to 101. Next, as the swarm size is of particular interest in our later analysis, we analyse the performance for each hyper-parameter for a range of swarm sizes. During the analysis we fix the unconsidered hyper-parameter(s) at the optimal value as derived in the cross-hyper-parameter-analysis (see Table 5.1). The following ranges of hyper-parameters and swarm size are evaluated:

$$\epsilon \in \{0; 0.001; 0.01; 0.125; 0.25\}, \qquad \rho_v \in \{0, 0.001, 0.01, 0.1, 0.2\},$$
$$\rho_w \in \{0; 0.01; 0.05; 0.1; 0.2\}, \qquad N \in \{41; 81; 121; 161; 201\}$$

Because of the relatively heavy computational load of the realistic Webots simulations, a choice between the Webots and Particle Simulator is a choice between the executable number of simulations, and the level of abstraction of the simulations. Since we want to investigate the performance for a broad range of hyper-parameter and swarm size values, the analysis is performed using the Particle Simulator. Using the Particle Simulator a swarm is employed in the symmetric obstacle environment (see Table 5.2) for 300 seconds. All (non-hyper) parameters are chosen equal to the default values as given by Table 5.1.

For the parametric analysis it suffices to derive conclusions based on the average foraging performance of the self guiding swarm in the exploitation phase. We therefore only consider the average foraging performance after convergence measure. In general we observe that the runs converged after 150 seconds. So, the navigation delay is measured over the interval $t \in [150s, 300s]$. Every data point shown is the median of 50 independent simulation runs.

5.3.2. Expectations

Let us analyze the relation between the evaporation rates and the system behavior. For ant systems the evaporation rate can intuitively be interpreted as the rate at which a system forgets about the past and learns about the present. In the ant system, updating of the systems knowledge is done by continuous marking of the traveled paths. The system forgets about the past by evaporation of the markings. The higher the rate, the faster the markings evaporate, but also, the more marking is dropped. In our case updating implies rewarding for positive actions and propagating these rewards through the system. The higher the evaporating rate, the higher the impact of single positive rewards, but also, the faster these updates will be forgotten. Note that for an evaporation rate of zero, the impact of actions is zero, so the system will behave randomly.

So, a high or low evaporation rate is the difference between high impact of single present actions and fast forget about the history of actions, and low single impact, but long memory. In the exploration phase we want the system to learn fast. We want that an agent that receives a reward has a large impact on the system, such that this action attracts other agents quickly to the area of reward. Once the system has converged, we do not want the trails to be destroyed by single non positive (random) actions, we want to slowly improve the paths created.

It is harder to analyze the impact of the guiding velocities' evaporation rate. If one thinks about the guiding velocities as the gradients of the weight field, following the evaporation rate chosen for the weights seems as the most logical thing to do. However, as explained in Section 3.3, The system dynamics do not necessarily lead to this coupling between the weight and guiding velocity fields. Therefore the choice for the rates is not that straightforward and the expected performance for different rates is hard to predict.

Remark that, although we will only consider static environments for now, higher evaporation rates

have a positive impact on the ability of a system to adapt to environmental changes. If for example the location of obstacles changes, possibly the possible and optimal paths change, therefore the preferred actions change, such that we want to forget the past and learn from the present. This implies that higher evaporation rates should result in better adaptability performance.

The impact of the exploration rate on the system behavior is easier to analyse. The higher the exploration rate, the more often agents will randomly pick their movement action. The exploration rate has a low impact on the swarms behavior in the exploration phase as the agents will move already randomly if no pheromone field is available. The exploration rate has mainly impact on the ability to optimize the existing paths once the food is found. Or formally said, a non zero exploration rates enables the system to overcome local minima. The exploration rate does has a direct impact on the performance while exploiting: a system will inherently perform worse for higher exploration rate as agents will more often not follow the created paths, and because of that not contribute to the fullest to the exploiting task. Again whether a environment is static or not is of importance. Since agents will deviate more often from the beaten paths for a higher exploration rate, the swarm will be able to overcome changes to the environment more easily.

5.3.3. Results

Figures 5.4a, 5.4b and 5.5a show the performance for different combinations of the hyper-parameters (ϵ, ρ_w and ρ_v). Figures 5.5a, 5.5b and 5.5c show the relation between the hyper-parameters and the swarm size. Given the simulation results we observe the following:

1. Figure 5.4c shows a correlation of zero between the evaporation rates. The evaporation rate of the guiding velocity seems to dictate the performance. Only for ρ_v equal to 0.1 a (negative) effect on the performance for increasing ρ_w can be observed. Figures 5.4a and 5.5a confirm that ρ_w has relatively small impact on the performance of the system. Figures 5.4b and 5.5b confirm the high impact of a ρ_v on the systems performance.

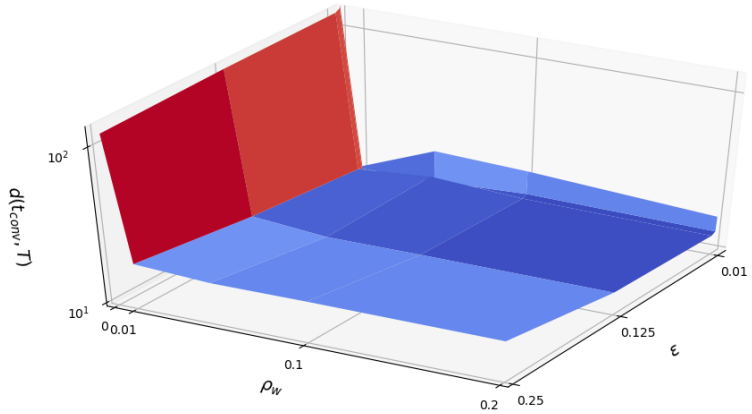
   This result seems to contradict with the interpretation of the guiding velocities as the gradients of the weight fields. As it seems, the system requires the establishment of stable paths, which are optimized over time pushed by the weight field. Besides, it is surprising that the ρ_w has such a low impact on the system.

2. Figures 5.4a and 5.5c show a similar pattern for the impact of the exploration rate on the performance: An increasing exploration rate results in a decrease in performance. However, the performance for an exploration rate of zero differs. Figure 5.5c does not show any negative effect on the performance, where Figure 5.4a does show a large negative effect. The results do not show the expected ability for non-zero exploration rate to overcome local minima and enable the swarm to converge to the most optimal path, the global minimum. We doubt whether the system is able to optimize the paths after exploration.

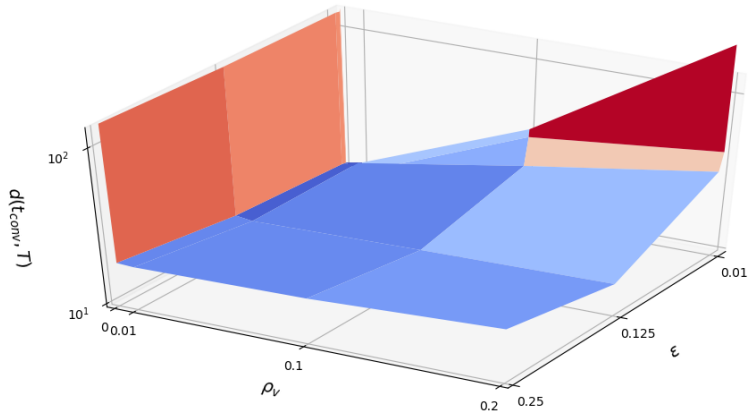
3. Figure 5.4b confirm our observation for the need of slowly adapting guiding velocities. If ρ_v is increased, the only way to improve the performance is to increase the randomness, i.e. not use the information stored in the field.

4. Figures 5.5a, 5.5b and 5.5c all show no relation between the swarm size and the choice hyper-parameters. Moreover we can already conclude that an increasing swarm size has a positive impact on the systems performance (in case of particle simulations). We will further analyze this relationship in Section 5.4.

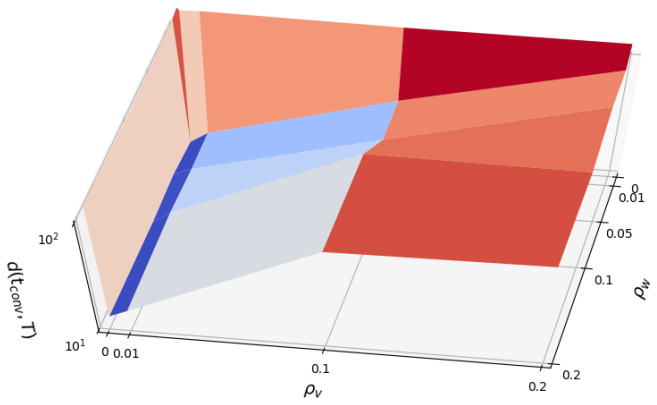
5. The best performance can be achieved for ρ_w, ρ_v and ε equal to 0.01, 0.001 and 0.01 respectively.



(a) exploration rate (ϵ) vs. evaporation rate of the weights (ρ_w)

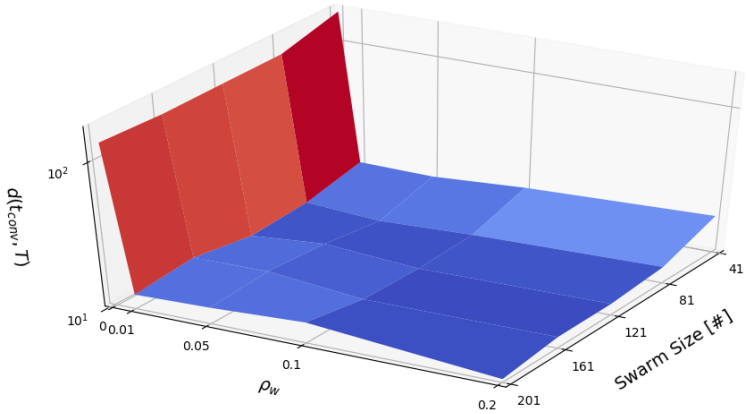


(b) exploration rate (ϵ) vs. evaporation rate of the directional vectors (ρ_v)

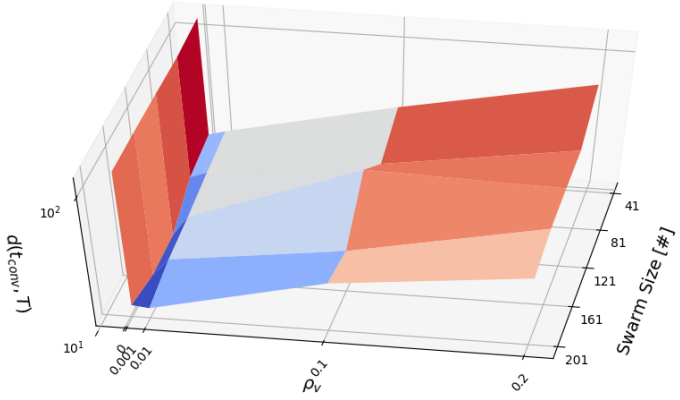


(c) evaporation rate of weights (ρ_w) vs. evaporation rate of the direction vectors (ρ_v)

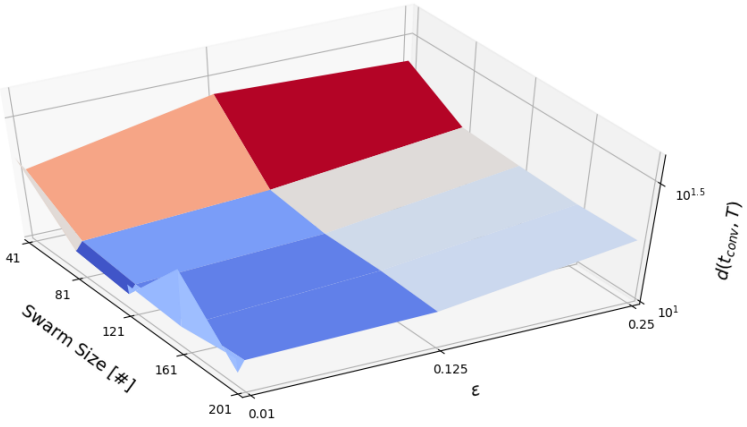
Figure 5.4: Average foraging performance, measured by the average navigation delay, of the self-guiding swarm of size 101 applying different exploration and evaporation rates and deployed in the environment with symmetric obstacle.



(a) evaporation rate weights (ρ_w) vs. swarm size



(b) Evaporation rate directional vectors (ρ_v)



(c) Exploration rate (ϵ)

Figure 5.5: Average foraging performance, measured by the average navigation delay, of the self-guiding swarm of different sizes applying different exploration rates and deployed in the environment with symmetric obstacle.

5.4. Swarm Size Analyses

In this section the impact of the swarm size (N) on the foraging performance of a self-guiding swarm is investigated, while fixing the hyper-parameter values to the values derived in Section 5.3.

5.4.1. Setup

Since we are investigating the impact of the swarm size, which is inherently related to the interaction of the agents bodies, we will run in our simulations in both the particle as the Webots simulator. For the simulations the defaults parameters are used (as given by Table 5.1). We evaluate the systems performance for the environment with and without symmetric obstacle (see Table 5.2) for 400 seconds. The performance is measured both by the navigation delay and the entropy (over time). The navigation delay is measured for all agents and for the foraging agents only, from $t = 200$ to $t = 400$. Each data point includes results from 12 independent test runs. We investigate the performance for the following swarm sizes: $\{41, 81, 121, 161, 201\}$.

5.4.2. Expectation

Increasing the number of agents has potentially positive and negative effects on the foraging performance of a swarm. The more agents, the more the system is enriched by updates from different locations per time step, which should imply faster learning, regarding both creation and optimization of the path. As more random moving agents simply cover more of the environment we expect a system to explore the environment on average faster for increasing swarm size. On average, because our system is driven by randomness, so there will always be a swarm consisting of a small number of agents that will create a path faster. Regarding this random aspect, we also expect that increasing the swarm size will decrease the spread in performance for a number of independent runs. The more agents, the more the combination of all agents will move like the expected behavior of the swarm. Also, if a (whether or not optimal) path is established, more agents would imply relatively more agents that can act as foragers, because the number of beacons only depends on the size and shape of the environment. This should result in a lower average navigation delay measured over all agents

In practise increasing the number of agents has one major disadvantage: The robots occupy space and need to perform obstacle avoidance manoeuvres to not collide with each-other. The robots are around 4cm in diameter, and start collision avoidance manoeuvres when they are close ($\approx 2cm$) to an obstacle or another robot. The more agents the more agents need to perform collision avoidance. While performing collision avoidance, agents are not of value to the system. Too many agents attracted to a path can create 'traffic jams', which could decrease the performance of the swarm overall. We will refer to an impact on the foraging performance due to too high agent densities by the *overcrowding* effect. We provided an example of expected negative impact due to overcrowding. But one could also think of a positive effect: Overcrowding could enforce agents to speed up the exploration of an environment as they are pushed away from the area explored by others.

The question is: Up till which point will the positive effects of a larger swarm size will be undone by the negative overcrowding effects? Note that the behavior of the system in the Particle simulations, in which we simulate the agents as particles, is not impacted by the overcrowding effect. Therefore, the difference in performance between the Particle and Webots simulations will provide useful insights into the overcrowding effect.

5.4.3. Results

Figure 5.6 shows the average navigation delay and entropy measure for both Particle and Webots simulations for the environment with symmetric obstacle. Figure 5.7 shows the same type of plots for the environment without obstacle. Given these results we can conclude the following:

1. In terms of the average navigation delay, the self-guiding swarm performs significant better than the fully randomly moving swarm for every situation considered. The average navigation delay of the foraging agents of a swarm is significant larger than the most optimal navigation delay. Note however, that this lower bound is extremely conservative (as explained in Subsection 5.2.1).

2. For all swarm sizes considered, it holds that the average navigation delay is larger for the envi-

ronment with obstacle (Figure 5.6a and 5.7a), than for the environment without obstacle (Figure Figures 5.6b and 5.7b). Which should be obvious as the optimal path without obstacle is shorter. but confirms correct behavior of the self guiding swarm.

More surprisingly, the spread of the average navigation delay is much larger for the environment without obstacle than the environment with obstacle. Given the large number of outliers and the large sizes of the third and fourth quartiles, our hypothesis is that the paths break after some time during the exploration phase. Analyzing abstraction plots of under-performing particle simulations for this environment showed that the particles are not able to create one single path, but preserve a broad range of paths. As an result, the weight and guiding velocity fields stay relatively 'weak' and random moving agents have much impact, resulting in death ending paths or loops.

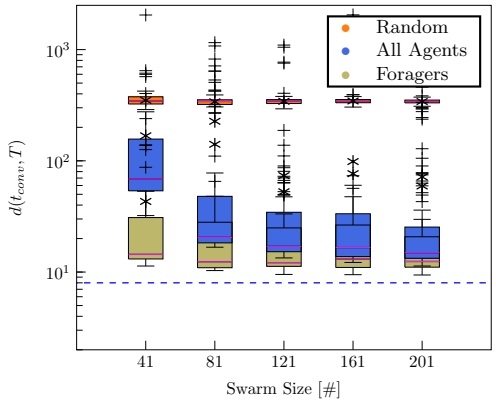
3. The spread of the average navigation delays is much larger for the Particle simulations (Figures 5.6a and 5.7a) than for the Webots simulations (Figures 5.6b and 5.7b). Given the amount of outliers and the size of the third and fourth quartiles, it seems that for the particle simulator the change of extremely bad results is much higher than for the Webots simulator. Probably the limited room to move for the robots in the Webots simulator prevents them from choosing to radical movements.

4. For a size of $N = 49$ or smaller, too many agents are needed as beacons, hence the performance (specially when considered the full swarm) is significantly worse than for bigger swarms. We can see for both the numerical (Figures 5.6a and 5.7a) and Webot (Figures 5.6b and 5.7b) simulations that, for growing swarm sizes, the performance increases and the variance in the results reduces. For the Webots simulations this tendency reverses at a certain point (around $N = 110$). Since this phenomenon is not observed for the Particle simulations, this must be due to the *overcrowding* effect. Experiments are run on a small arena, so the swarm reaches a point where there can be too many robots in the same space.

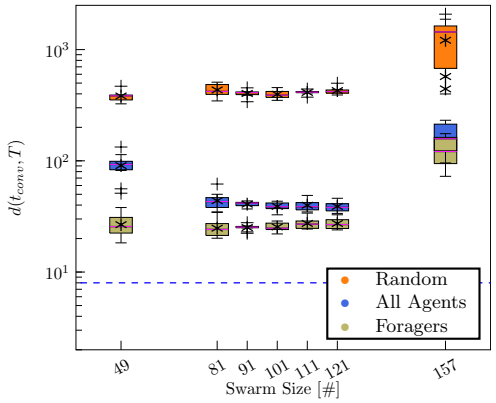
5. The average entropy of the Particle simulations (Figures 5.6c and 5.7c) is for all situations smaller than the entropy of the Webots simulations (Figures 5.6d and 5.7d). This observation supports our previous explanation of the decreasing performance due to the *overcrowding* effect. The entropy measure confirms that the particles are much closer clustered around the trajectories, Remark that we also exclude the entropy measure for linkage distances smaller than δ_r, such that one can derive conclusions based on the absolute difference in entropy measures for the Particle and Webots simulator.

6. At $t = 0$ all agents are starting at the nest, from there the entropy start to increase, which relates to the swarm covering the environment in search for the target region. After the exploration phase, the entropy begins to settle to lower values as the robots accumulate over the trajectories. In general the entropy is higher for the symmetrical obstacle environment (Figures 5.6c and 5.6d) than no obstacle environment (Figures 5.7c and 5.7d) due to first the split of agents among the two possible paths, and second the fact that the minimum length path is longer than without obstacles.

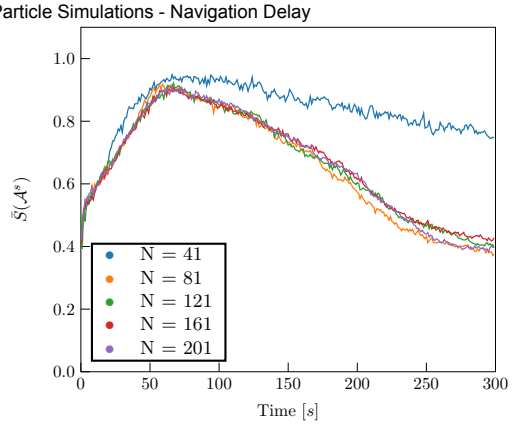
7. The entropy time series of the swarm size per situation and simulator show similar shapes. In the exploitation phase a swarm of size 41 does not have enough agents in foraging state to cover the whole trajectory. Hence during the exploitation phase, swarms of this size show much lower clustering than the other sizes considered.



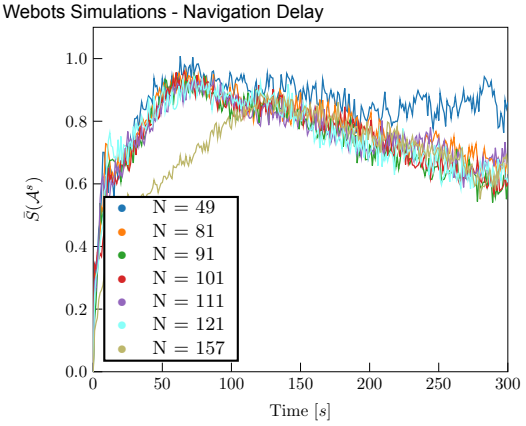
(a) Particle Simulations - Navigation Delay



(b) Webots Simulations - Navigation Delay

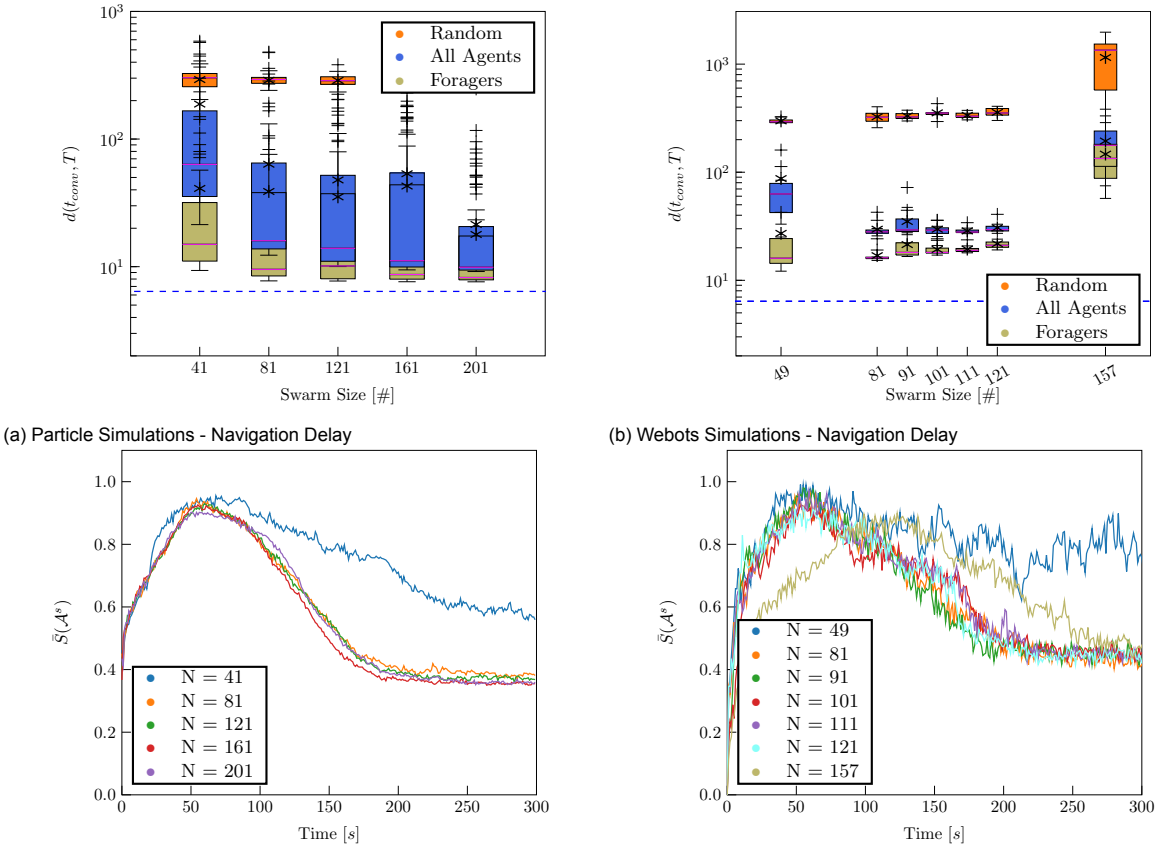


(c) Particle Simulations - Entropy



(d) Webots Simulations - Entropy

Figure 5.6: Average foraging performance, measured by the average navigation delay and entropy, of self-guiding swarm of different sizes deployed in the environment with symmetric obstacle.



(a) Particle Simulations - Navigation Delay

(b) Webots Simulations - Navigation Delay

(c) Particle Simulations - Entropy

(d) Webots Simulations - Entropy

Figure 5.7: Average foraging performance, measured by the average navigation delay and entropy, of self-guiding swarm of different sizes deployed in the environment without obstacle.

5.5. Robustness Analysis

The results shown in the previous sections rely on perfect communication and measure capabilities of robots. We assumed perfect continuous listening capacity; perfect storage of the received messages of the beacons per time step; perfect heading direction measurement of robots; and perfect communication of the weights, heading directions and guiding velocities between foragers and beacons. All these assumptions are in reality impossible to achieve. In this section we investigate the robustness of the self-guiding swarm against disturbances. The potential sources of noise, summed above, are captured in three potential factors of disturbance of the system: perturbed communication, limited listening capacity of the beacons, and temporal agent failure.

5.5.1. Setup

To simulate for the measurement and communication noise related to the communication of the weights and guiding velocities we introduce the perturbed weights and directional vectors,

$$\begin{aligned} \tilde{\omega}(k) &= \omega(k)|\mu_k^{\omega}| \\ \tilde{v}(k) &= (\cos\tilde{\alpha}(t) \quad \sin\tilde{\alpha}(t))^T \qquad \mu_k^{\omega} \sim \mathcal{N}(1,\sigma_{\omega}^2) \\ \tilde{\alpha}(t) &= \alpha(t) + \mu_k^{v} \qquad \mu_k^{v} \sim \mathcal{N}(0,\sigma_{v}^2) \\ \text{with } v(k) &= (\cos\alpha(t) \quad \sin\alpha(t))^T \end{aligned} \tag{5.9}$$

Since for the weights, the relative difference in weights captures the information, the weights are multiplied by a random variable. Perturbations on the guiding velocities, i.e. measurements and communication noise, can best be represented as addition of a random variable, as the absolute differences in angle captures the information. For the analysis of the robustness against measurement noise, we assume the robots to broadcast the perturbed weights and guiding velocity, or in other words, for the robustness analysis $\omega_b^s(t)$ and $v_b^s(t)$ in line 2 of Algorithm 4 are replaced by $\tilde{\omega}_b^s(t)$ and $\tilde{v}_b^s(t)$, and $v_f(t)$ and $\Delta_f^s(t)$ in line 2 of Algorithm 5 are replaced by $\tilde{v}_f(t)$ and $\tilde{\Delta}_f^s(t)$. We first analyse the effect of perturbed communication on the weight and guiding velocities separately, after which we analyse the situation in which we apply noise on both. We run simulations for a broad range of possible standard deviations: $\sigma_{\omega}, \sigma_{v} \in \{0.01, 0.05, 0.1, 1, 2.5\}$.

To simulate for a limited listening capacity of the beacons, we define the maximum number of messages a beacon can store per τ, as the maximum buffer capacity $n_{buf} \in \mathbb{R}_+$. The maximum buffer capacity is simulated by randomly filtering n_{buf} messages per time step for each beacon. In reality, while continuous listening, one would stop listening if a beacon has received n_{buf} messages. Simulations are performed for a range of maximum buffer values, $n_{buf} \in \{1, 2, 5, 10, 20, 50\}$

Regarding the implementation of temporal agent failure we define some time span, τ_{off}, during we disable n_{off} agents. Every τ_{off} seconds, the to disable agents are randomly sampled from the set of all, already deployed, agents. A disabled agent does not move, nor does it receives or sends messages. After τ_{off} seconds the disabled agents become again part of the swarm, and the steps repeat. As we only want to simulate temporal failure of agents, the memory of a disabled agent is not reset, so beacons do remember their stored weight and guiding velocity values, and foraging agents remember their former state. Simulations are performed for different amounts of agents to disable every $\tau_{off} = 20$ seconds, we analyze: $n_{off} \in \{1, 5, 20, 50\}$

For our simulations, a swarm of 101 robots is employed in the environment with symmetric obstacle (see Table 5.2) for 400 seconds. The simulations in which only noise on the weight or guiding velocities communication is applied, are only executed in the particle simulator. All the other simulations are performed using both the particle and the Webots simulator. The default parameter set as provided by Table 5.1 is used. The performance is measured using the average navigation delay and entropy. The navigation delay is measured only for the foraging agents from $t = 200$ to $t = 400$. Each data point created by the Particle simulator includes results from 50 independent runs and each data point created by the Webots simulator includes results from 10 independent runs.

5.5.2. Expectations

We expect the system to be resistant to small disturbances of guiding velocity updates, because of the randomness already included in the movement dynamics of the robots. One could argue that swarms of larger size should be more resistant against communication noise then smaller swarms, as the more random local update actions per time step, the more the randomness should tend to the expected value overall, i.e. the added communication randomness is averaged by the increased number of communication actions.

Considering a limited beacon buffer, lets first refresh that the region covered by a beacon, consists of unique-sub-regions all covered by a unique combination of beacons. Therefore an update of a foraging agents located in this unique region is unique as well. Every unique update captures some knowledge of the environment. Especially in the exploration phase, it should hold that the more unique updates per time step, the faster the system converges to the shortest path, the better the foraging performance. As a consequence of a limitation on the beacon buffer, the allowed number of updates per time step is restricted, so one would expect the performance to decrease. Note however that in reality one beacon will dominate a region, i.e. will have the largest weight of all its neighbouring beacons, hence it is very unlikely that each update from an unique region results in an unique update.

However, it is questionable if this effect will be visible for swarms of the sizes we consider. Since the number of unique regions is relatively large compared to the amount of foraging agents. To put this in perspective, assume that in the exploration phase the whole area is covered with beacons. This will take around 35 robots, which leaves 76 foraging agents. If these agents are uniformly distributed over the environment every beacon will on average have two agents within its region of influence, disregarding the change that these agents are in the same unique areas. So one can doubt if an beacon listening cap larger than two has much impact on the performance of the system in the exploration phase.

Theoretically the maximum number of unique areas within the region of a beacon is equal to 6 (can be shown as result of sphere packing theory in 2d with overlap). So, if the foraging agents are always perfectly distributed over the unique areas, a beacon only requires a buffer of size 6. In reality, this is of course not the case. Moreover, in the exploration phase the robots are clustered around the path. A limitation on the buffer size should therefore be of direct impact on the capability of the system to optimize the path during this phase, especially in the particle simulations, in which there is no limitation on the distance between the robots. For the Webots simulations we expect no difference in performance for any n_{buf} larger than 10, as per time step only around five moving robots can occupy the region of influence of a beacon. For this same reason, the change that several robots within the same region of influence of a beacon, cover unique regions, should be large. So for the Webots simulations, one would expect that n_{buf} smaller than 10 is of impact on the swarms ability to optimize the paths.

The systems resistance to beacon failure follows by construction: if a beacon temporarily fails, and if a beacon is required in this region, the failing beacon will be replaced by a foraging agent. As a beacon only updates if it receives updates from foragers (See (3.8) and (3.9)), disabling all foraging agents would only pause the swarm. Temporarily failure of single foraging agents will only slow down the creation of the weight and guiding velocity fields.

5.5.3. Results

Figure 5.8 shows the average navigation delay for different values n_{buf}, while keeping σ_ω and σ_v equal to zero. Given these results we can conclude the following:

1. The Particle Simulator results (Figure 5.8a) show a significant larger spread for n_{buf} equal to one. Surprisingly there is no difference in performance for n_{max} chosen larger than one. Apparently, receiving more than two (unique) updates per time step does have no performance increasing effect on the weight and guiding velocity fields generated by the beacons.

2. The Webots results (Figure 5.8b) shows an increase in average navigation delay for n_{buf} equal to one, similar as we saw for the particle simulator. For n_{buf} larger than one, no significant difference in foraging performance is observed. As it turns out, the number of unique updates for the beacons is not a limited factor for the overall performance of the system.

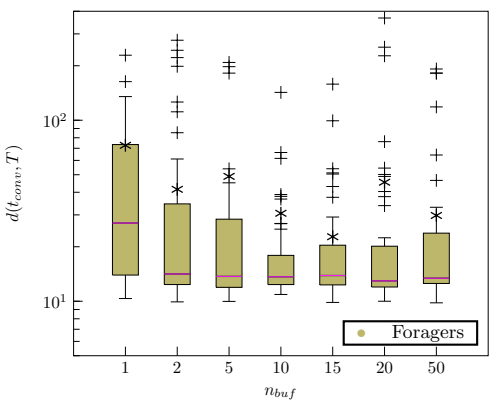
Figure 5.10 shows the average navigation delay and entropy of the Particle and Webots simulations for different values of σ_w^2 and σ_v^2. Given these results we observe and conclude the following:

1. Figures 5.10c and 5.10e show the foraging performance for the particle simulations for noise applied to both the weights as guiding velocity communication. The self-guiding swarm appears to be resistant to high amounts of communication noise. Only for σ_w^2 and σ_v^2 larger than 1, a significant increase in navigation delay is observed. The entropy plot clearly shows less clustering of the agents for variance equal to 1 and 2.5.

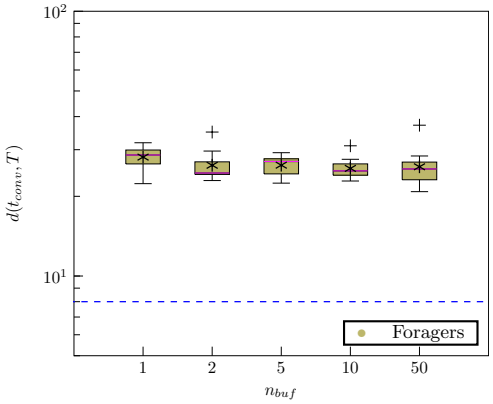
2. Figures 5.10d and 5.10f show the foraging performance of the Webots simulations. The results show a similar pattern as the Particle results: The system seems resistant for small values of the variance: for a variance equal to or larger than 1 a decrease in clustering is observed; and for a variance of 2.5 a clear increase in navigation delay is visible.

3. Figures 5.10a and 5.10b provide insights in which part of the communication, weights or guiding velocities, affects the performance most, if noise is added. The noise on the weight communication slightly increases the average navigation delay for σ_w^2 equal to and larger than 1. Perturbed guiding velocity communication has more impact on the performance: σ_v^2 equal to 1 already has a significant impact on the foraging performance. For very large amounts of noise, σ_v^2 equal to 2.5, the swarm's performance is almost worse than the fully random controlled swarm (see Figure 5.6a). These observations imply that the guiding velocity field is less robust to disturbed updates than the weight field. Probably, frequent inconsistent communication of the direction of heading from forager to beacon results in unstable or non created paths.

   It seems that our reasoning: the system already includes random steps, so it should be capable to deal with randomness, noise, in the communicated heading directions, does not hold. Indeed, this reasoning marginalizes the connection between the heading direction update and the location at which this update is performed. Although an agent can move randomly, the direction it stores does represents the direction of entrance into the region of influence of a beacon. If the stored heading direction does not represent the direction of heading, paths seems to get destroyed easier. Looking at the abstraction plots of under-performing simulations, we observed that especially disturbances close to the food in the beginning of the exploration phase have major impact on the system performance. In this phase the path to the food is unfinished and still brittle. A few updates pointing towards the wrong direction repel foragers from the food. As the beacons close to the food store relatively large amount of weights in this phase, the paths are not easily restored.

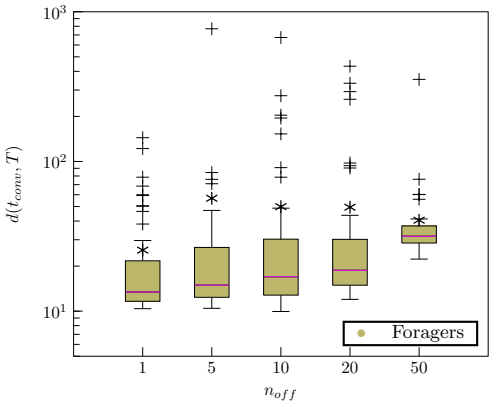


(a) Particle Simulations - Navigation Delay

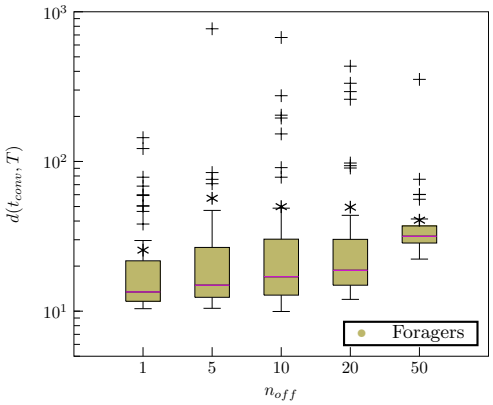


(b) Webots Simulations - Navigation Delay

Figure 5.8: Average foraging performance, measured by the average navigation delay, of the self-guiding swarm with maximum listening buffers for the beacons, deployed in the environment with symmetric obstacle.

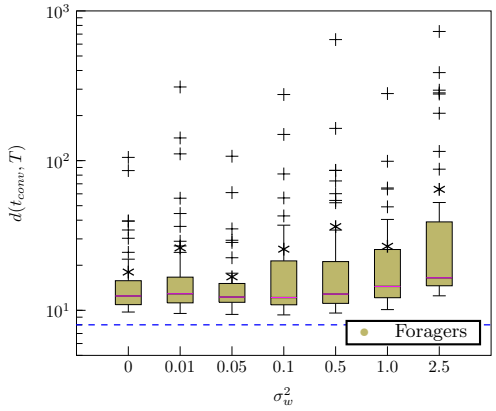


(a) Particle Simulations - Average Navigation Delay

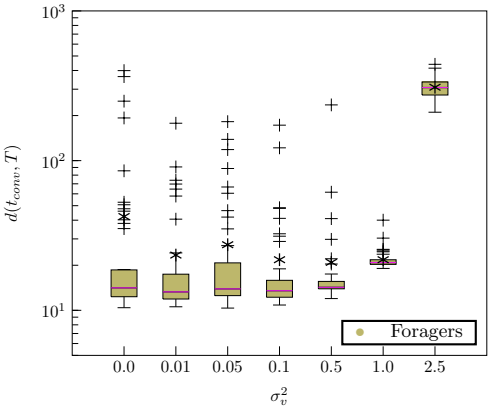


(b) Webots Simulations - Average Navigation Delay

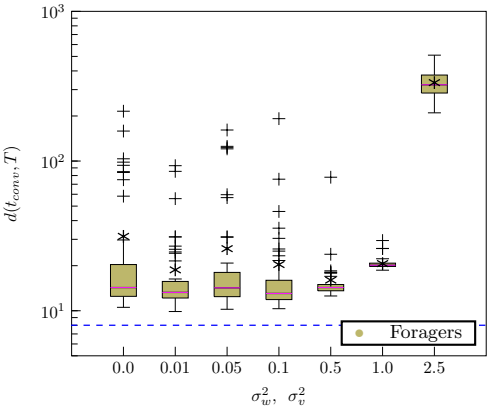
Figure 5.9: Average foraging performance, measured by the average navigation delay, of the self-guiding swarm with temporal agent failure, deployed in the environment with symmetric obstacle.



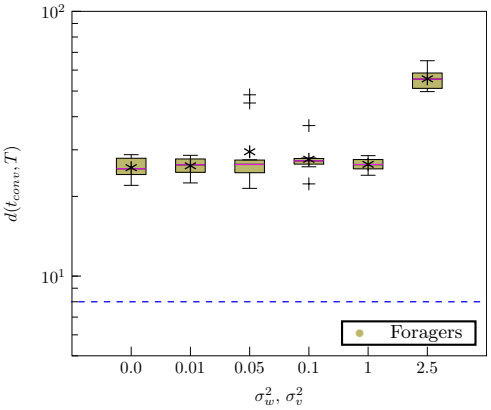
(a) Perturbed communication of weights - Particle Simulations - Navigation Delay



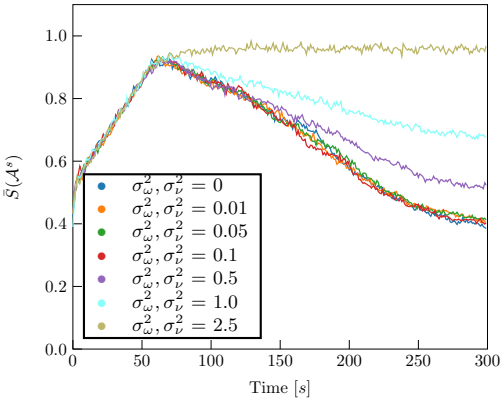
(b) Perturbed communication of guiding velocities - Particle Simulations - Navigation Delay



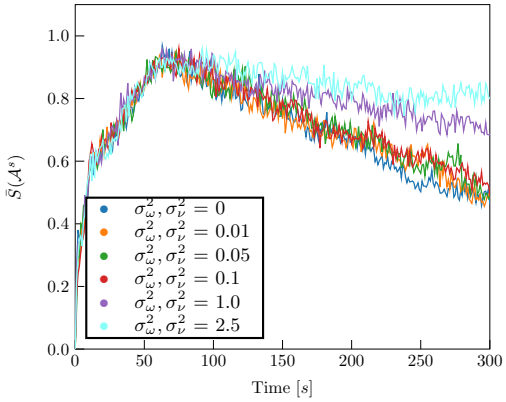
(c) Perturbed communication of both weights and guiding velocities - Particle Simulations - Navigation Delay



(d) Perturbed communication of both weights and guiding velocities - Webots Simulations - Navigation Delay



(e) Perturbed communication of both weights and guiding velocities - Particle Simulations - Entropy.



(f) Perturbed communication of both weight as guiding velocity communication - Webots Simulation - Entropy

Figure 5.10: Average foraging performance, measured by the average navigation delay and entropy, of the self-guiding swarm with perturbed communication, deployed in the environment with symmetric obstacle.

5.6. Model Extension

In section 3.2 we presented extensions for the self-guiding swarm: the *Beacon-Forager Switching* and *Moving Beacons*-extensions to optimize the beacon infrastructure; and *Double Updating*-extension to speed up the convergence of the weight and guiding velocity fields. In this section we investigate if these extensions result in a measurable increase of foraging performance.

5.6.1. Setup

The dynamics of the extensions are described in Section 3.2; in Section 3.3 the intended impact on the swarms behavior is explained; and in Section 5.1.1 it is explained how the extended dynamics are included in the robots behavior. In our analysis we consider the following combinations of extensions (with in the brackets the abbreviation used as reference in the plots):

1. The self-guiding swarm without extension, which serves as reference for the performance to beat **(Normal)**.
2. Self-guiding swarm plus the *Beacon-Forager Switching*-extension **(Switch)**.
3. Self-guiding swarm plus the *Beacon-Forager Switching*- and *Moving Beacons* **(Move)**-extensions.
4. Self-guiding swarm plus the *Beacon-Forager Switching*- and *Double Updating*-extensions **(Dbl.)**.
5. Self-guiding swarm plus the *Beacon-Forager Switching*-, *Moving Beacons*- and *Double Updating*-extensions **(Mov. & Dbl.)**.

Note that the *Moving Beacons*-extension is always implemented in combination with the *Beacon-Forager Switching*-extension, as explained in section 3.2. Because early in our simulations we found that the *Beacon-Forager Switching* extension has a positive impact on the foraging performance, we only analyzed the *Double Updating*-extension in combination with the *Beacon-Forager Switching*-extension.

For our simulations, a swarm of 101 and 49 robots is employed in the environment with symmetric obstacle (see Table 5.2) for 800 seconds. All simulations are both executed in the Particle and the Webots simulator. The default parameter set as provided by Table 5.1 is used. The overall performance is measured by the average navigation delay and entropy. The average navigation delay is measured for all agents for $t \in [200\text{s}, 400\text{s}]$. The minimal navigation delay for consecutive trips is used to analyse the optimality of the converged path. We analyse the 20 smallest ($m = 20$) navigation delays for 2 and 3 consecutive trips ($n \in [2, 3]$) to ensure we only measure trips resulting from following the created paths and not just trips driven by randomness. In addition, we present abstraction plots of the simulation results to illustrate and explain the impact of the extensions on the behavior of the swarm. Compared to our previous analysis, the difference performance for this analysis turn out to be very small, we therefore increase the number of runs per data point: each data point of the particle simulator includes results from 100 independent runs and each data point of the Webots simulator includes results from 15 independent runs.

5.6.2. Expectations

In the exploration phase the swarm creates a beacon infrastructure approximately covering the whole environment. During the exploitation phase only a small part of this infrastructure is used, only the beacons forming the path(s) used to travel between the target regions. The *Beacon-Foraging Switching*-extension should allow unused beacons to participate again as foragers, thereby becoming again of value for the swarm. One can expect that the number of foraging agent increase as result of *Beacon-Foraging Switching*, hereby increasing the number of trips and decreasing the average navigation delay of all agents. However, the additional number of foragers could also decrease performance due to the *overcrowding* effect. In Section 5.4 we observed the overcrowding effect for a swarm of size 157. Therefore, we expect the *Beacon-Foraging Switching* to increase foraging performance for the swarm of 49 robots, and consolidate foraging performance for the swarm of 101 robots.

The locations of the beacons limit the possible paths. So, the performance of the self-guiding swarm is limited by the locations of the beacons. In the standard case, in which agents randomly explore the environment, and switch to beacon state if no beacon detected δ-distance close, we only control the

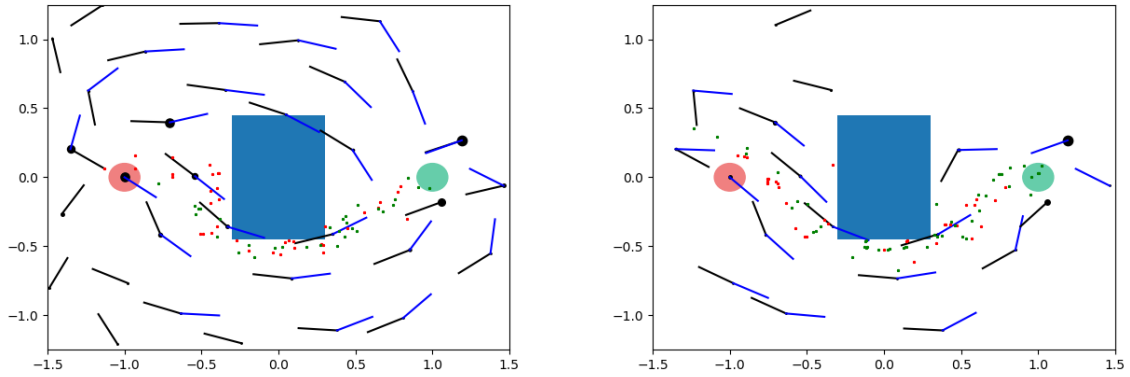
minimal distance between beacons. We do not guarantee any optimality of the beacon location. By the *Moving Beacon*-extension we try to control the beacons to optimal positions. It is doubtful if, especially for the Webots simulations, the effect of the moving beacons is visible in the average foraging performance, as also other factors limit the overall (random) performance. However, we do expect a decrease of the minimal navigation delay for 2 and 3 consecutive trips, as this gives a measure of the most optimal path(s) created per run.

As described in section 3.3 our concept could be abstracted as a graph by defining the unique areas as nodes, and the weights and guiding velocities per area can be translated to the transition probabilities to the neighbouring nodes. The *Double Updating*-extension as inspired on [36] requires this graph to be symmetric. Although the nodes and edges are independent of the weights, the transition probabilities of the resulting graph (per pheromone type) are not necessarily symmetrical. This is partly due to the fact that weight (and guiding velocity) updates are not only impacting one node: the update of a forager within a specific unique area also impacts all other unique areas within the reach of the beacons covering this specific unique area, thereby impacting transition probabilities of other nodes. It is therefore very doubtful if double-updating will work in our case. The outcome depends on how symmetric our graphs in practice are.

5.6.3. Results

Figures 5.11 and 5.12 show some abstractions of a particle simulation including the *Beacon-Forager Switching*-extension and the *Moving Beacon*-extension, respectively. Figure 5.11a shows that all the foraging agents are attracted to a stable, as far as we can determine visually, close to optimal path facilitated by the agents in beacon state. Clearly, after the creation of the path, only a small number of beacons are needed to create the path. The *Beacon-Forager Switching*-extension enables beacons which are not of use anymore, to switch back and participate in the exploitation of the target region again, as shown in Figure 5.11b. Note that the location of the beacons in Figure 5.11b has not changed.

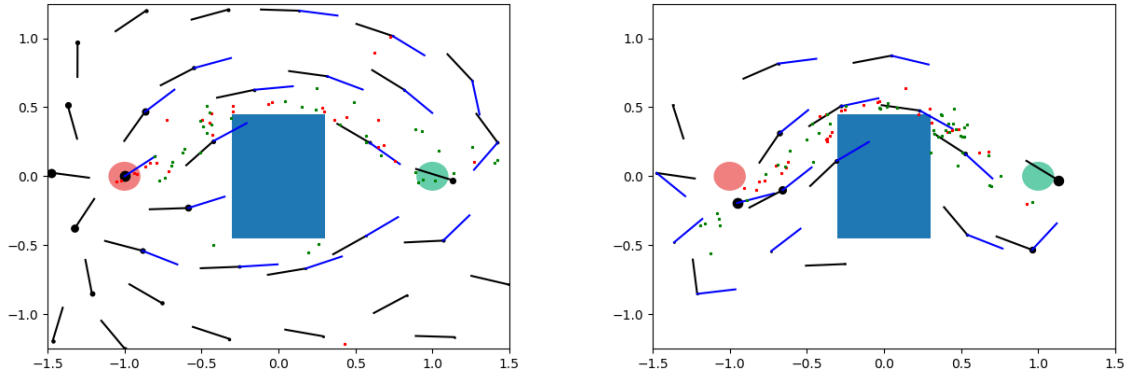
The optimality of the path is restricted by the random initialization of the beacons in the exploration phase. Figure 5.12b shows the path created if we apply the *Moving Beacons*-extension. Remember that beacons move towards the shortest angel between the two guiding velocities stored, trying to align both vectors and thereby create a straight (without obstacles) most optimal path. Comparing Figures 5.12a and 5.12b we observe, besides the removal of beacons, that the beacons moved as close as possible towards the obstacle resulting in a shorter path. Hence we conclude that both the *Beacon-Forager Switching*- and *Moving Beacons*-extension do work as expected in the situations considered. The question is: What is the average foraging performance gain of these extensions compared to the normal case, which does include the sustainability and consistence of the added complexity?



(a) Particle Simulations - $t = 100$s

(b) Particle Simulations - $t = 200$s

Figure 5.11: Abstraction plots of a particle simulation of the self-guiding swarm plus *Beacon-Forager Switching*-extension.



(a) Particle Simulations - $t = 110$s

(b) Particle Simulations - $t = 350$s

Figure 5.12: Abstraction plots of a particle simulation of the self-guiding swarm plus *Moving-Beacon*-extension.

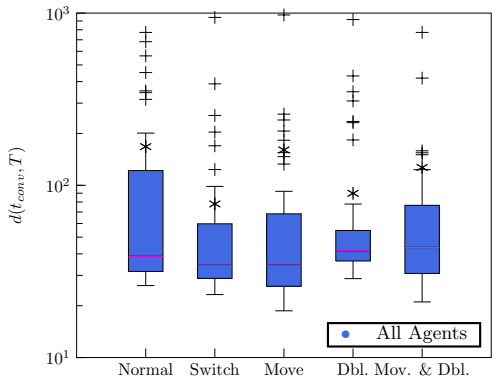
Figure 5.13 shows the average foraging performance results of the extended methods for both the Particle and the Webots simulations. For our analysis we take the normal-case as reference. From these we can conclude the following:

1. For the Particle simulations the *Beacon-Forager Switching*-extension shows a significant decrease of average navigation delay for all agents compared to the normal-case (Figures 5.13a and 5.13e). Figures 5.13c and 5.13g show no difference in the clustering of the agents, the entropy, over time. So it seems that the increased average foraging performance is simply due to an increased number of foraging agents.

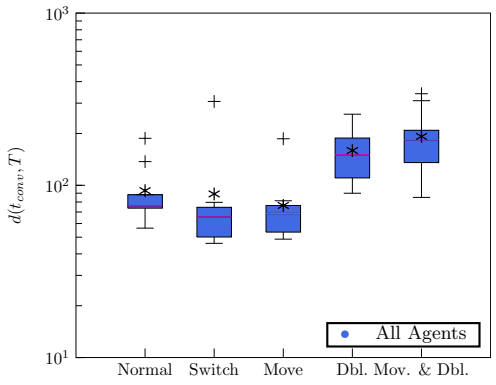
   The Webots simulations for a swarm of size 49 show a similar decrease in average navigation delay (see Figure 5.13b), hence we conclude that that the *Beacon Forager Switching*-extension also enables more robots to act as foragers in a realistic setting. In reality, as we have seen before, at some point the performance gain realized by increasing the number of foragers is nullified, or even turned into a performance loss, by the overcrowding effect. This effect explains why for a swarm of size 101 no difference in foraging performance is observed (see Figure 5.13f).

2. The Particle simulations show a significant higher spread in average navigation delay for the *Moving Beacons*-extension than the *Beacon-Forager Switching*-extension (Figure 5.13a and 5.13e). It seems that moving the beacons using our P-controller (see (3.15)), can both positively and negatively impact the guiding infrastructure.

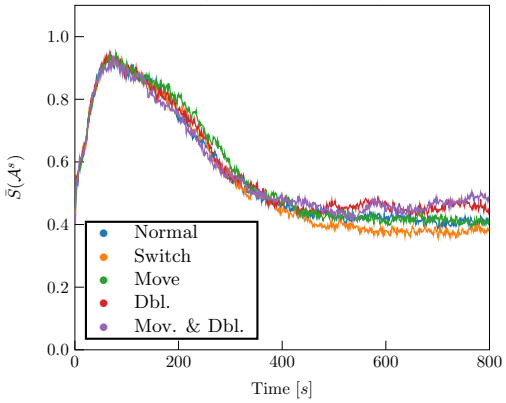
3. Both the Particle and the Webots simulations show that the *Double Updating*-extension does not result in better foraging performance (Figure 5.13a, 5.13e, 5.13b and 5.13f). The entropy plots (Figures 5.13c, 5.13d, 5.13g and 5.14 show that up till some point during the transition from the exploration to exploitation phase (between 200 and 400 seconds) the clustering behaviour of the double updating cases follow a similar trajectory as the other cases. After this point the entropy starts to increase again, which indicates that paths somehow break after creation.



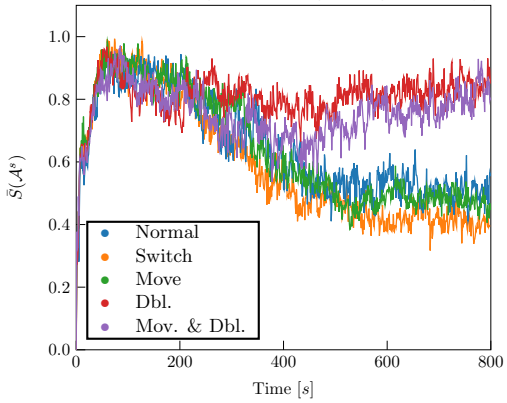
(a) Particle Simulations - 49 Agents - Average Navigation Delay



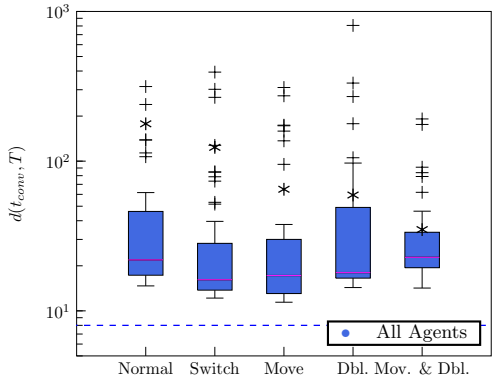
(b) Webots Simulations - 49 Agents - Average Navigation Delay



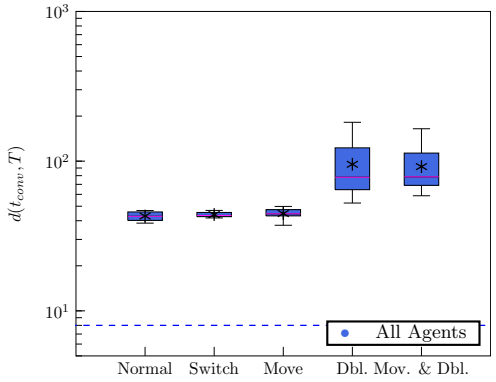
(c) Particle Simulations - 49 Agents - Entropy



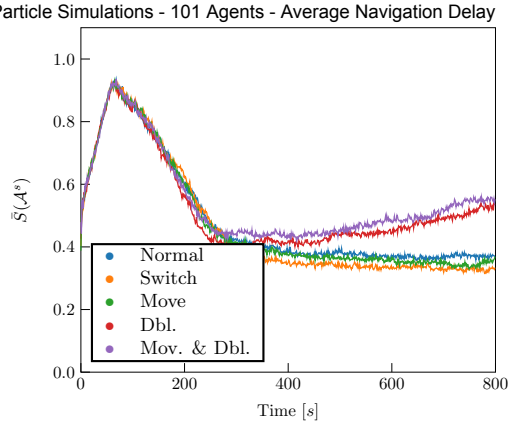
(d) Webots Simulations - 49 Agents - Entropy



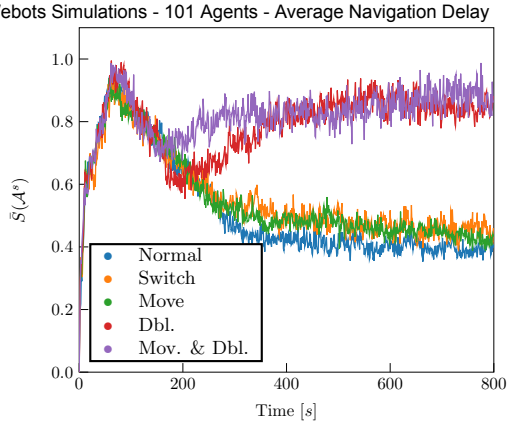
(e) Particle Simulations - 101 Agents - Average Navigation Delay



(f) Webots Simulations - 101 Agents - Average Navigation Delay



(g) Particle Simulations - 101 Agents - Entropy

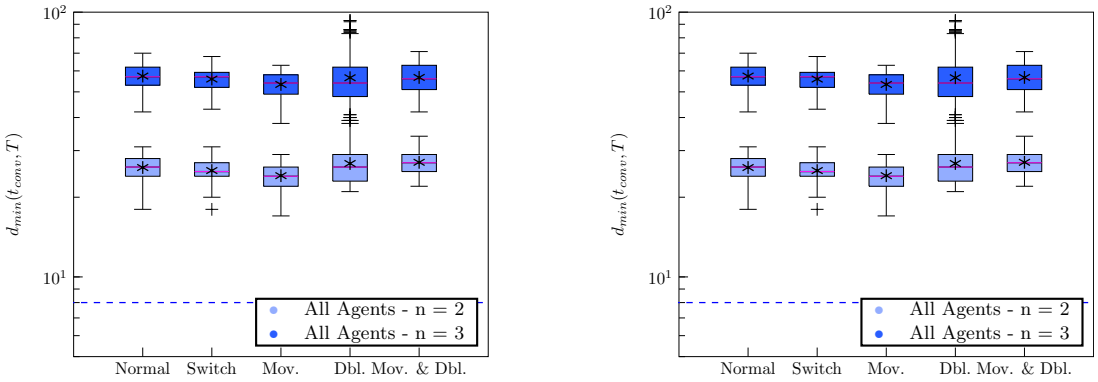


(h) Webots Simulations - 101 Agents - Entropy

Figure 5.13: Average foraging performance, measured by the average navigation delay and entropy, of the self-guiding swarm plus extensions of swarm sizes 101 and 49, deployed in the environment with symmetric obstacle.

We use the navigation delays of 2 and 3 successive trips to measure the optimality of the path (at some point in time). Figure 5.14 shows the optimal foraging performance results for the Webots simulations. From these results we conclude the following:

1. Figures 5.14a and 5.14b both show slightly smaller optimal navigation delays for the *Beacon-Forager Switching*- and *Moving Beacons*-extension. This observation confirms our hypothesis that the *overcrowding* effect vanishes the potential performance gain for the swarm of size 101.
2. Figure 5.14a and 5.14b both show smaller optimal navigation delays for the *Moving Beacon*-extension compared to both the normal case and the *Beacon-Forager Switching*-extension. Although the *Moving Beacon*-extension does not result in better average foraging performance, it does seem that at some point in time the moving beacons create a more optimal path.



(a) Webots Simulations - 49 Agents

(b) Webots Simulatiosn - 101 Agents

Figure 5.14: Optimal foraging performance, measured by the minimal navigation delay for 2 and 3 successive trips, of the self-guiding swarm plus extension.

5.7. Summary

The main insights of this section can be summarized by the following key points:

1. The proposed self-guiding swarm performs reasonable well in both small and medium scale environments with and without (static) obstacles. Moreover, as the results of the Particle simulations can be interpreted as the expected performance for large scale environments, for which the agents occupy a very large area, i.e. communicate over large ranges (δ), the results indicate good foraging performance for large scale environments. Hence, we may assume that the concept is scalable.

2. Although, in theory, increasing the amount of agents of a swarm will increase its foraging performance, in reality the *overcrowding*-effect will limit the foraging performance of the swarm for increasing size. In our experiments, this tipping point is between the .... and agents per m^2 (with single Elisa-3 robot capturing ... m^2).

3. The agents only require a small buffer ($3 \leq n_{buf} \leq 5$) to store and receive messages every τ seconds.

4. The self-guiding swarm is resistant to realistic communication disturbances and individual temporal agent failures.

5. The *Beacon-Forager Switching*-extension does increase the average foraging performance of the self-guiding swarm. Note that for large swarms the *overcrowding*-effect can nullify the performance gains.

6. The *Moving Beacons*-extension does not increases the average foraging performance, but does increase the optimal foraging performance.

At last, we want to remark that, as also mentioned at the particular experiments, we created some illustrating GIFs for some of the experiments. These GIFs can be found on our Youtube channel: *YOUTUBE - Stigmergy Based Swarm Robotics*. In addition to the experiments discussed here, we also included animations of a large-scale Webots simulation as presented at the *ICRA - Real World Swarms*-Workshop at this channel. For this demonstration we released 100 Robots in a large environment with obstacles (See 'obstacle'-environment in Table 5.2. Note that we used the default parameters as given by Table 5.1, but in contrast to the experiments presented here, we used the normal distribution for the noise on the heading direction for the random movement.

6

Conclusion

6.1. Summary of Results

After the explanation of the foundations in Chapter 2, the presentation of our design in Chapter 3, the derivation of formal results in Chapter 4, and the analysis of the experimental results in Chapter 5, we can draw several conclusions. The summary of the results of this work can be presented as follows:

- The different approaches of implementing stigmergy in swarm robotics are classified and reviewed. It is concluded that the use of beacons that locally store, process and broadcast information is most promising. To be more specific, the use of a homogeneous swarm with autonomous beacons, i.e. a robotic swarm in which all agents have equal capabilities and agents can switch to a beacon mode, will result in most efficient performance.
- The foraging problem is introduced and the most promising swarm robotic solutions are discussed. In general we observed that many of the swarm robotic solutions to the foraging problem include implicit assumptions that will prevent them of being applied to real situations or large scenarios. It is concluded that the so called *Stigmergic Pheromone Reinforcement*-concept in which pheromones are treated as utility estimates for environmental states stored in beacons, and decision-making functions somewhat resemble, but are not the same as, utility updates and state transition functions in reinforcement learning, is an interesting strategy which allows for formalisation, thereby optimization and formal analyses, and application of reinforcement learning techniques. Moreover the *Landmark*-concepts of storing navigation information using path integration is interesting as it drops the need of (relative) location measures.
- We presented a foraging swarm design where robots are able to guide each-other to forage without the need of position measurements, infrastructure or global knowledge. Moreover, the system relies on one-hop communication only with limited range, and does not require direction or distance information on signals, nor line-of sight connectivity. The system does require agents to know their orientation and have some maximal communication range.

  Our design is based on letting the agents in the swarm act as foragers or as guiding agents (beacons) and pheromone-inspired communication. Beacons store, process and broadcast local navigation information, which consists of two types of weights and guiding velocities, one per target region. In a manner similar to reinforcement learning we cast the weights as local state utility values, and apply formal utility update equations based loosely on value iteration and temporal difference learning. From the perspective of the foraging agents, the beacons are a graph of states with utility values. However, we do not let the foragers walk on this graph. We exploit the symmetry of the foraging problem and store the inverse heading directions of agents as the local guiding velocity corresponding to a local weight, together forming the local navigation information. Hereby marginalizing the localization information to global directions.
- In addition to the basic concept, we proposed the *Beacon-Forager Switching*- and *Moving Beacons*-extensions which create a dynamic beacon infrastructure and aim to increase the efficiency,

the relative amount of required agents in beacon state, of the swarm and optimize the created paths, restricted by the locations of the beacons. Moreover. we introduced the *Double Updating*-extension, to enforce faster path creation and convergence.

- In Chapter 4, we tried to derive formal results on the behavior of the self-guiding swarm. In Section 4.1, we made some restrictive assumptions regarding the domain, a grid polygon, and the random movements of agents, random walks, such that we could apply the formal results of the *Probabilistic Coverage*-Process (as presented in Subsection 2.7.1) to the self-guiding swarm and derive bounds on the expected *cover time* and its variance, and the *commute time* between two sub-regions. In Section 4.2, we tried to derive more general formal results regarding the exploration of the domain. For this we took inspiration from the results in RRT exploration as introduced in Subsection 2.8.1. We derived the guarantee that the entire domain will be explored and covered by beacons as time goes to infinity.

- We introduced the average and optimal navigation delay as measures of the foraging performance. In addition we introduced the hierarchical entropy as quality measure of the solution to the foraging problem, providing insights in the accumulation of agents around trajectories.

- In general, we conclude that the proposed self-guiding swarm implemented on Elisa-3 robots performs reasonable well in both small and medium scale environments with and without (static) obstacles. Moreover, as the results of the Particle simulations can be interpreted as the expected performance for large scale environments, for which the agents occupy a very large area, i.e. communicate over large ranges (δ), the results indicate good foraging performance for large scale environments. Hence, we may assume that the concept is scalable.

  In addition, we investigated the performance for different swarm sizes, the robustness of the design against disturbances and temporal agent failures, and the required agent listening buffers. We observed that although in theory increasing the amount of agents of a swarm will increase its foraging performance, in reality the *overcrowding*-effect will limit the foraging performance of the swarm for increasing size. Also, we verified that the self-guiding swarm is resistant to realistic communication noise and individual temporal agent failures. Failure of foraging agents will only (temporal) decrease, or slow down the increase of, the foraging performance. If a beacon breaks or is removed, it is replaced by a foraging agent, therefore only temporal decreasing the foraging performance. At last, we showed that the agents only require a small buffer ($3 \leq n_{buf} \leq 5$) to store and receive messages every τ seconds.

- The *Beacon-Forager Switching*-extension does increase the average foraging performance of the self-guiding swarm. Note that for large swarms the *overcrowding*-effect can nullify the performance gains.

- It is observed that the optimality of the trajectories is affected by the resulting distribution of the beacons. Although the *Moving Beacons*-extension does not increase the average foraging performance, the extension does increase the optimal foraging performance.

6.2. Applications

In our work we aimed to develop an efficient and practical implementable swarm robotic solution for the foraging problem. Let us first conclude on the advantages of applying the self-guided swarm to tackle the foraging problem in comparison to other proposed solutions:

- Consider the general navigation task for large amounts of robots. In general, it holds that a swarm robotic solution is preferred over a classical robotic solution, because of its inherent scalability with the number of robots, robustness with respect to noisy conditions, and fault tolerance in case of individual failure. Moreover, complex navigation tasks relying on single agents will require a representation of the environment a priori, leaving a robot with the non-trivial task of self-localization or require agents to construct the map itself while moving in the environment, which is already difficult in a static environment and becomes increasingly complex when multiple robots are considered, or for dynamic environments.

- Why would you, if a covering beacon network is available, prefer an approach relying on indirect reinforcement of pheromone-fields over path planning algorithms for networks graphs, such as the *Dijkstra* and *A** algorithm?

  Besides that, as we shortly discussed in Section 2.8, the *A** and *Dijkstra* algorithms are not able to cope with uncertainties and still suffer large amounts of memory for large search spaces, especially for a non standardized environment discretization. In practice, the *A** and *Dijkstra* algorithms rely on line-of-sight communication or some other (visual) mechanism to confirm beacon connectivity. Either way, the connection mechanism will restrict the maximum distance between beacons and increase the number of required beacons for environments with many obstacles. The self-guided swarm does not include any of these requirements, and as such can be implemented for larger and more environmental situations.

- Consider the swarm robotic solutions as discussed in Section 2.5. An overview of the required capabilities of the self-guided swarm and the reviewed solutions is provided by Table 2.1. Observe that all the reviewed work requires robots to either have some form of position measurement (global or relative), some form of infrastructure or centralised knowledge entity, or both. Therefore our design would especially be suited for navigation in environments where any access to global or relative positioning is not possible, and where the infrastructure is limited. Additionally, some solutions cascade tables of agent data through the network. hereby limiting the scalability of these methods. As confirmed by the Particle Simulator results, the self-guided swarm is scalable up to large amount of agents.

We believe the self-guided swarm could almost immediately be applied to problems similar to the foraging problem in the fields of swarm robotics and multi-agent systems:

- In theory, the self-guided swarm should be able to deal with shifting target-regions. Therefore we believe the self-guided swarm should be able to solve the Hunter-Prey problem [60], in which multiple agents initialized in some region try to catch some shifting target. Future research should be done on any necessary adaptions of the dynamics, and hyper-parameters to deal with shifting target regions.

- In theory, the inclusion of more internal states could allow for more sophisticated behavior than just there-and-back-again foraging trails. For example, with additional states and pheromones, agents could be able to achieve tours between multiple way-points, including ones with intersecting paths. Realisation of more sophisticated behaviour including more pheromones types is shown by [65].

At last we present some more creative ideas for applications:

- Since the self-guided swarm does not require any positioning infrastructure in place, only needs a single point of reference for its global orientation, and is be able to cover large environments, our approach could be suited for space exploration or deep water mining.

- Our approach could be implemented as a (offline) path optimization tool version being a more-less parallelized version of the RRT-algorithm. Our approach presents an alternative to the methods designed for the RRT-algorithm to decide on the optimal created path. In addition, the *Moving Beacons*-extension could potentially decouple the optimality from the sampling process.

6.3. Future Work

We will summarize the in our opinion most interesting and promising aspects that could be subject of further and deeper work, ranked by concreteness:

- Our main goal was to develop a system directly applicable for swarm robotic systems. Our extensive analysis of the Elisa-3 robot in the realistic *Webots* simulator did not raise any doubts on the changes of a successful implementation of in a real similar situations using the Elisa-3 robot. As we mentioned in Chapter 5, we hope to test our concept soon in real-life using Elisa-3 robots.

- The self-guided swarm should be able to deal with dynamic environments, i.e. moving obstacles or moving target regions. Non-zero exploration and evaporation rates force agents to keep exploring the environments, even after convergence of the weight and guiding velocity fields, and refreshes the local navigation information over time. Potentially, some small adaptions to the self-guides swarms design are needed, such as: evaporate the store weights and guiding velocities even if no update is received, and make an agent in beacon state switch to forager state if it collides with some moving obstacle. Moreover, the values for the hyper-parameters should be investigated once more for the changed setting. Likely higher evaporation rates will result in higher foraging performance.

- We observed how the optimality of the trajectories is affected by the positional configuration of the beacons. Moreover, we showed in Section 5.6 the potential effect of allowing the beacons to reconfigure. The proposed simplistic P-controller (see Section 3.2.2) already results in an decrease of the optimal navigation delay. We believe that relatively easy more effect can be obtained by applying more advanced control rules. For example, we observed that closer to the target regions, one wants to preserve a broader network of beacons. The ratio of the stored weights could indicate the relative distance to a target region, and thereby used to control the allowed angels between the guiding velocities. Moreover, we force beacon agent to move very slowly compared to the foraging agents, allowing the local navigation information to be updated to the new location. It would be interesting to investigate if applying some adjustment of the stored information related to the movement, could allow for faster beacon speeds, and thereby result in faster re-configuration.

- In particular, there is much to be gained in the exploration process. For our current concept we apply very simplistic random exploration of the agents, assuming the agents to perform a random walk while exploration. Even without using any information of the system, but using alternatives for the random walk, for example the Lévy flight [93] or the improved random walk methods proposed by [67] would already result in faster exploration. Another option is to define a third pheromone type, some repulsive pheromone, creating a weight and guiding velocity field pushing exploring agents away from already explored regions.

- We have experimentally verified that, over a diverse variety of set-ups, the weights ω_b^s converge on average to a fixed point forming a gradient outwards the target regions, therefore guiding the swarm to and from the goal regions. Moreover we guaranteed that for a given initial combination of foraging agents, the entire domain will be explored and covered by beacons. We leave for future work the formal guarantees regarding the expected weight field values ω_b^s and guiding velocities ν_b^s. Although our system possess an extra dynamic coupling with respect to the transition probabilities between regions, we are assuming that we can apply the approach of [63] who employed field-techniques to re-formulate a stochastic multi-agent problem into a deterministic autonomous system.

- The agents in our system may be seen as mobile automata responding to and updating external states in the environment. For our system, each agent checks periodically its actions to perform. In particular for the updating actions of the foraging agents, i.e. the dropping of pheromones, an event-triggered scheme could potentially be implemented, using for example the sensed weights and guiding velocities of the beacons within the agents region of influence. In this way, especially as the paths are converged, the efficiency in the usage of system resources could be increased.

Bibliography

[1] Steven Adams, Daniel Jarne Ornia, and Manuel Mazo Jr. A self-guided approach for navigation in a minimalistic foraging robotic swarm. *arXiv preprint arXiv:2105.10331*, 2021.

[2] Sjriek Alers, Karl Tuyls, Bijan Ranjbar-Sahraei, Daniel Claes, and Gerhard Weiss. Insect-inspired robot coordination: foraging and coverage. In *Artificial Life Conference Proceedings 14*, pages 761–768. MIT Press, 2014.

[3] Tucker Balch. Hierarchic social entropy: An information theoretic measure of robot group diversity. *Autonomous robots*, 8(3):209–238, 2000.

[4] Friedrich G Barth et al. *Insects and flowers. The biology of a partnership.* George Allen & Unwin, 1985.

[5] Mike Blow. 'stigmergy': Biologically-inspired robotic art. *Mechatronics and Animatronics in the Creative and Entertainment Industries and Arts*, page 45, 2005.

[6] Eric Bonabeau, Guy Theraulaz, and Jean-Louis Deneubourg. Quantitative study of the fixed threshold model for the regulation of division of labour in insect societies. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 263(1376):1565–1569, 1996.

[7] Eric Bonabeau, Marco Dorigo, Directeur de Recherches Du Fnrs Marco, Guy Theraulaz, Guy Théraulaz, et al. *Swarm intelligence: from natural to artificial systems*. Number 1. Oxford university press, 1999.

[8] Arne Bosien, Volker Turau, and Franco Zambonelli. Approaches to fast sequential inventory and path following in rfid-enriched environments. *International journal of radio frequency identification technology and applications*, 4(1):28–48, 2012.

[9] Gilles Caprari, Patrick Balmer, Ralph Piguet, and Roland Siegwart. The autonomous micro robot" alice": a platform for scientific and commercial applications. In *MHA'98. Proceedings of the 1998 International Symposium on Micromechatronics and Human Science.-Creation of New Industry-(Cat. No. 98TH8388)*, pages 231–235. IEEE, 1998.

[10] Ashok K Chandra, Prabhakar Raghavan, Walter L Ruzzo, Roman Smolensky, and Prasoon Tiwari. The electrical resistance of a graph captures its commute and cover times. *Computational Complexity*, 6(4):312–340, 1996.

[11] K Cheng, TS Collett, A Pickhard, and R Wehner. The use of visual landmarks by honeybees: Bees weight landmarks according to their distance from the goal. *Journal of Comparative Physiology A*, 161(3):469–475, 1987.

[12] Dong-Hwan Choe, David B Villafuerte, and Neil D Tsutsui. Trail pheromone of the argentine ant, linepithema humile (mayr)(hymenoptera: Formicidae). *PLoS One*, 7(9):e45016, 2012.

[13] Matthew Collett, Thomas S Collett, Sonja Bisch, and Rüdiger Wehner. Local and global vectors in desert ant navigation. *Nature*, 394(6690):269–272, 1998.

[14] Matthew Collett, Duane Harland, and Thomas S Collett. The use of landmarks and panoramic context in the performance of local vectors by navigating honeybees. *Journal of Experimental Biology*, 205(6):807–814, 2002.

[15] Jorge Cortes, Sonia Martinez, Timur Karatas, and Francesco Bullo. Coverage control for mobile sensing networks. *IEEE Transactions on robotics and Automation*, 20(2):243–255, 2004.

[16] Jorge Cortes, Sonia Martinez, and Francesco Bullo. Spatially-distributed coverage optimization and control with limited-range interactions. *ESAIM: Control, Optimisation and Calculus of Variations*, 11(4):691–719, 2005.

[17] J-L Deneubourg, Serge Aron, Simon Goss, and Jacques M Pasteels. The self-organizing exploratory pattern of the argentine ant. *Journal of insect behavior*, 3(2):159–168, 1990.

[18] Edsger W Dijkstra et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[19] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39, 2006.

[20] Frederick Ducatelle, Alexander Förster, Gianni A Di Caro, and Luca M Gambardella. Supporting navigation in multi-robot systems through delay tolerant network communication. *IFAC Proceedings Volumes*, 42(22):25–30, 2009.

[21] Frederick Ducatelle, Gianni A Di Caro, Alexander Förster, and Luca Gambardella. Mobile stigmergic markers for navigation in a heterogeneous robotic swarm. In *International Conference on Swarm Intelligence*, pages 456–463. Springer, 2010.

[22] Frederick Ducatelle, Gianni A Di Caro, Carlo Pinciroli, and Luca M Gambardella. Self-organized cooperation between robotic swarms. *Swarm Intelligence*, 5(2):73, 2011.

[23] Frederick Ducatelle, Gianni A Di Caro, Carlo Pinciroli, Francesco Mondada, and Luca Gambardella. Communication assisted navigation in robotic swarms: self-organization and cooperation. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4981–4988. IEEE, 2011.

[24] David Filliat and Jean-Arcady Meyer. Map-based navigation in mobile robots:: I. a review of localization strategies. *Cognitive Systems Research*, 4(4):243–282, 2003.

[25] Ryusuke Fujisawa, Shigeto Dobata, Daisuke Kubota, Hikaru Imamura, and Fumitoshi Matsuno. Dependency by concentration of pheromone trail for multiple robots. In *International Conference on Ant Colony Optimization and Swarm Intelligence*, pages 283–290. Springer, 2008.

[26] Ryusuke Fujisawa, Hikaru Imamura, Takashi Hashimoto, and Fumitoshi Matsuno. Communication using pheromone field for multiple robots. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1391–1396. IEEE, 2008.

[27] Ryusuke Fujisawa, Shigeto Dobata, Ken Sugawara, and Fumitoshi Matsuno. Designing pheromone communication in swarm robotics: Group foraging behavior mediated by chemical substance. *Swarm Intelligence*, 8(3):227–246, 2014.

[28] MA Porta Garcia, Oscar Montiel, Oscar Castillo, Roberto Sepulveda, and Patricia Melin. Path planning for autonomous mobile robot navigation with ant colony optimization and fuzzy cost function evaluation. *Applied Soft Computing*, 9(3):1102–1110, 2009.

[29] Simon Garnier, Faben Tache, Maud Combe, Anne Grimal, and Guy Theraulaz. Alice in pheromone land: An experimental setup for the study of ant-like robots. In *2007 IEEE swarm intelligence symposium*, pages 37–44. IEEE, 2007.

[30] Simon Goss, Serge Aron, Jean-Louis Deneubourg, and Jacques Marie Pasteels. Self-organized shortcuts in the argentine ant. *Naturwissenschaften*, 76(12):579–581, 1989.

[31] Plerre-P Grassé. Les insects dans leur univers. 1946.

[32] Plerre-P Grassé. La reconstruction du nid et les coordinations interindividuelles chezbellicositermes natalensis etcubitermes sp. la théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs. *Insectes sociaux*, 6(1):41–80, 1959.

[33] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[34] Adam T Hayes, Alcherio Martinoli, and Rodney M Goodman. Distributed odor source localization. *IEEE Sensors Journal*, 2(3):260–271, 2002.

[35] Andrew Howard, Maja J Matarić, and Gaurav S Sukhatme. Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In *Distributed Autonomous Robotic Systems 5*, pages 299–308. Springer, 2002.

[36] Brian Hrolenok, Sean Luke, Keith Sullivan, and Christopher Vo. Collaborative foraging using beacons. In *AAMAS*, volume 10, pages 1197–1204, 2010.

[37] Shin Ishii, Wako Yoshida, and Junichiro Yoshimoto. Control of exploitation–exploration meta-parameter in reinforcement learning. *Neural networks*, 15(4-6):665–687, 2002.

[38] Robert L Jeanne. The evolution of the organization of work in social insects. *Monitore Zoologico Italiano-Italian Journal of Zoology*, 20(2):119–133, 1986.

[39] Robert Johansson and Alessandro Saffiotti. Navigating by stigmergy: A realization on an rfid floor for minimalistic robots. In *2009 IEEE International Conference on Robotics and Automation*, pages 245–252. IEEE, 2009.

[40] Richard Kershner. The number of circles covering a set. *American Journal of mathematics*, 61 (3):665–671, 1939.

[41] Ali Abdul Khaliq, Maurizio Di Rocco, and Alessandro Saffiotti. Stigmergic algorithms for multiple minimalistic robots on an rfid floor. *Swarm Intelligence*, 8(3):199–225, 2014.

[42] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 2, pages 995–1001. IEEE, 2000.

[43] Daisuke Kurabayashi and Hajime Asama. Knowledge sharing and cooperation of autonomous robots by intelligent data carrier system. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 1, pages 464–469. IEEE, 2000.

[44] Daisuke Kurabayashi et al. Realization of an artificial pheromone system in random data carriers using rfid tags for autonomous navigation. In *2009 IEEE International Conference on Robotics and Automation*, pages 2288–2293. IEEE, 2009.

[45] Lydia E Kavraki Jean-Claude Latombe. Probabilistic roadmaps for robot path planning. *Pratical motion planning in robotics: current aproaches and future challenges*, pages 33–53, 1998.

[46] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. *The international journal of robotics research*, 20(5):378–400, 2001.

[47] Steven M LaValle et al. Rapidly-exploring random trees: A new tool for path planning. 1998.

[48] Sung G Lee and Magnus Egerstedt. Controlled coverage using time-varying density functions. *IFAC Proceedings Volumes*, 46(27):220–226, 2013.

[49] Sung G Lee, Yancy Diaz-Mercado, and Magnus Egerstedt. Multirobot control using time-varying density functions. *IEEE Transactions on Robotics*, 31(2):489–493, 2015.

[50] Nyree Lemmens and Karl Tuyls. Stigmergic landmarks lead the way. In *The 20th Belgian-Dutch Conference on Artificial Intelligence (BNAIC)*, pages 129–136, 2008.

[51] Nyree Lemmens and Karl Tuyls. Stigmergic landmark foraging. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 497–504, 2009.

[52] Nyree Lemmens and Karl Tuyls. Stigmergic landmark optimization. *Advances in Complex Systems*, 15(08):1150025, 2012.

[53] Nyree Lemmens, Steven de Jong, Karl Tuyls, and Ann Nowé. Bee system with inhibition pheromones. In *European conference on complex systems*. Citeseer, 2007.

[54] Anna Font Llenas, Mohamed S Talamali, Xu Xu, James AR Marshall, and Andreagiovanni Reina. Quality-sensitive foraging by a robot swarm through virtual pheromone trails. In *International conference on swarm intelligence*, pages 135–149. Springer, 2018.

[55] Chaomin Luo, Furao Shen, Hongwei Mo, and Zhenzhong Chu. An improved ant-driven approach to navigation and map building. In *International Conference on Swarm Intelligence*, pages 301–309. Springer, 2017.

[56] Peter Matthews et al. Covering problems for brownian motion on spheres. *Annals of Probability*, 16(1):189–199, 1988.

[57] Ralf Mayet, Jonathan Roberz, Thomas Schmickl, and Karl Crailsheim. Antbots: A feasible visual emulation of pheromone trails for swarm robots. In *International conference on swarm intelligence*, pages 84–94. Springer, 2010.

[58] Jean-Arcady Meyer and David Filliat. Map-based navigation in mobile robots:: Ii. a review of map-learning and path-planning strategies. *Cognitive Systems Research*, 4(4):283–317, 2003.

[59] Olivier Michel. Cyberbotics ltd. webots™: professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, 1(1):5, 2004.

[60] MB Naghibi-S and M-R Akbarzadeh-T. Stigmergy for hunter prey problem. In *Proceedings World Automation Congress, 2004.*, volume 16, pages 169–174. IEEE, 2004.

[61] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6292–6299. IEEE, 2018.

[62] Shervin Nouyan, Alexandre Campo, and Marco Dorigo. Path formation in a robot swarm. *Swarm Intelligence*, 2(1):1–23, 2008.

[63] Daniel Jarne Ornia, Pedro J Zufiria, and Manuel Mazo Jr. Mean field behaviour of collaborative multi-agent foragers. *arXiv preprint arXiv:2103.07714*, 2021.

[64] George F Oster and Edward O Wilson. *Caste and ecology in the social insects*. Princeton University Press, 1978.

[65] Liviu Panait and Sean Luke. A pheromone-based utility model for collaborative foraging. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004.*, pages 36–43. IEEE, 2004.

[66] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems*, 11(3):387–434, 2005.

[67] Bao Pang, Yong Song, Chengjin Zhang, Hongling Wang, and Runtao Yang. A swarm robotic exploration strategy based on an improved random walk method. *Journal of Robotics*, 2019, 2019.

[68] David Payton, Mike Daily, Regina Estowski, Mike Howard, and Craig Lee. Pheromone robotics. *Autonomous Robots*, 11(3):319–324, 2001.

[69] David Payton, Regina Estkowski, and Mike Howard. Pheromone robotics and the logic of virtual pheromones. In *International Workshop on Swarm Robotics*, pages 45–57. Springer, 2004.

[70] Anies Hannawati Purnamadjaja and R Andrew Russell. Guiding robots' behaviors using pheromone communication. *Autonomous Robots*, 23(2):113–130, 2007.

[71] Bijan Ranjbar-Sahraei, Gerhard Weiss, and Ali Nakisaee. A multi-robot coverage approach based on stigmergic communication. In *German Conference on Multiagent System Technologies*, pages 126–138. Springer, 2012.

[72] Bijan Ranjbar-Sahraei, Sjriek Alers, Karl Tuyls, and Gerhard Weiss. Stico in action. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 1403–1404, 2013.

[73] Bijan Ranjbar-Sahraei, K Tuyls, I Caliskanelli, B Broeker, D Claes, S Alers, and G Weiss. Bio-inspired multi-robot systems. In *Biomimetic Technologies*, pages 273–299. Elsevier, 2015.

[74] Andreagiovanni Reina, Alex J Cope, Eleftherios Nikolaidis, James AR Marshall, and Chelsea Sabo. Ark: Augmented reality for kilobots. *IEEE Robotics and Automation letters*, 2(3):1755–1761, 2017.

[75] Gene E Robinson. Regulation of division of labor in insect societies. *Annual review of entomology*, 37(1):637–665, 1992.

[76] Katherine Russell, Michael Schader, Kevin Andrea, and Sean Luke. Swarm robot foraging with wireless sensor motes. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 287–295. Citeseer, 2015.

[77] R Andrew Russell. Mobile robot guidance using a short-lived heat trail. *Robotica*, 11(5):427–431, 1993.

[78] R Andrew Russell. Laying and sensing odor markings as a strategy for assisting mobile robot navigation tasks. *IEEE Robotics & Automation Magazine*, 2(3):3–9, 1995.

[79] R Andrew Russell. Heat trails as short-lived navigational markers for mobile robots. In *Proceedings of International Conference on Robotics and Automation*, volume 4, pages 3534–3539. IEEE, 1997.

[80] R Andrew Russell. Ant trails-an example for robots to follow? In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, volume 4, pages 2698–2703. IEEE, 1999.

[81] Toshiki Sakakibara, Daisuke Kurabayashi, et al. Artificial pheromone system using rfid for navigation of autonomous robots. *Journal of Bionic Engineering*, 4(4):245–253, 2007.

[82] Mac Schwager, Daniela Rus, and Jean-Jacques Slotine. Decentralized, adaptive coverage control for networked robots. *The International Journal of Robotics Research*, 28(3):357–375, 2009.

[83] Frank Schweitzer, Kenneth Lao, and Fereydoon Family. Active random walkers simulate trunk trail formation by ants. *BioSystems*, 41(3):153–166, 1997.

[84] David Silver. Cooperative pathfinding. *Aiide*, 1:117–122, 2005.

[85] Ken Sugawara, Toshiya Kazama, and Toshinori Watanabe. Foraging behavior of interacting robots with virtual pheromone. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 3074–3079. IEEE, 2004.

[86] Jonas Svennebring and Sven Koenig. Building terrain-covering ant robots: A feasibility study. *Autonomous Robots*, 16(3):313–332, 2004.

[87] Mohamed S Talamali, Thomas Bose, Matthew Haire, Xu Xu, James AR Marshall, and Andreagiovanni Reina. Sophisticated collective foraging with minimalist agents: a swarm robotics test. *Swarm Intelligence*, 14(1):25–56, 2020.

[88] Ying Tan and Zhong-yang Zheng. Research advance in swarm robotics. *Defence Technology*, 9 (1):18–39, 2013.

[89] Sebastian B Thrun. Efficient exploration in reinforcement learning. 1992.

[90] Wolfhard von Thienen, Dirk Metzler, Dong-Hwan Choe, and Volker Witte. Pheromone communication in ants: a detailed analysis of concentration-dependent decisions in three species. *Behavioral ecology and sociobiology*, 68(10):1611–1627, 2014.

[91] Israel A Wagner, Michael Lindenbaum, and Alfred M Bruckstein. Robotic exploration, brownian motion and electrical resistance. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 116–130. Springer, 1998.

[92] Ko-Hsin Cindy Wang, Adi Botea, et al. Scalable multi-agent pathfinding on grid maps with tractability and completeness guarantees. In *ECAI*, pages 977–978, 2010.

[93] Vasily Zaburdaev, S Denisov, and J Klafter. Lévy walks. *Reviews of Modern Physics*, 87(2):483, 2015.

[94] Vittorio A Ziparo, Alexander Kleiner, Bernhard Nebel, and Daniele Nardi. Rfid-based exploration for large robot teams. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 4606–4613. IEEE, 2007.