Bio-inspired multi-robot coordination

Thesis submitted in accordance with the requirements of
the University of Liverpool for the degree of Doctor in Philosophy by

Bastian Broecker

July 2018

Contents

Illustrations

List of Figures

List of Tables

Notations

The following notations and abbreviations are found throughout this thesis:

ABC Artificial Bee Colony.

ACO Ant Colony Optimisation.

ANN Artificial Neural networks.

AODV Ad-hoc On-demand Distance Vector-Routingalgorithmus.

AS Ant System.

BCO Bee Colony Optimization.

BeePCo Bee Pheromone Coverage.

DOF Degrees of Freedom.

DQN Deep-Q-learning-Network.

FNN Feed-forward Neural network.

GRU Gated Recurrent Units.

HybaCo Hybrid Bee and Ant Pheromone Coverage.

I^2C Inter-Integrated Circuit.

IMU Internal Measurement Units.

IR Infra Red.

LCD Liquid Crystal Display.

LSTM Long Short-Term Memory.

MAV Micro Aerial Vehicle.

MaxCo Maximal Potential Coverage.

MCL Monte Carlo Localisation.

MDP Markov Decision Process.

MIMO Multiple Input Multiple Output.

MRS Multi Robot System.

PI Path Integration.

PID Proportional Integral Derivative.

POI Point of Interest.

PS Pheromone Signalling.

QMP Queen Mandibular Pheromone.

QR Queen Robot.

RL Reinforcement Learning.

RNN Recurrent Neural network.

SISO Single Input Single Output.

SPI Serial Peripheral Interface.

STC Spanning Tree Coverage.

StiCo Stigmergic Coverage.

TOA Time of Arrival.

TOF Time of Flight.

UAV Unmanned Aerial Vehicle.

UWB Ultra Wideband.

WiFi Wireless Fidelity.

WSN Wireless Sensor Nodes.

WSRNs Wireless Sensor Robot Networks.

Preface

This thesis is primarily my own work. The sources of other materials are identifed.

Abstract

This works addresses the complexity of coordinating in large multi-robot systems (MRS), by drawing inspiration from simple and effective strategies, observable in social-insect colonies e.g. ants and bees. These insects have evolved over a long period of time and show remarkable behaviours that are highly suitable for addressing the complex tasks that they are facing. In this thesis we introduce two novel social-insect based approaches, providing simple and effective coordination in the field of multi-robot coverage and explore their performances and properties in extensive case studies. To address their limitations in real-world situations, we introduce and evaluate an end-to-end system to allow their deployment on light weight robotic systems.

Acknowledgements

The completing of the work presented in this thesis, represents one of the biggest and longest challenges of my life to date. I would not been able to reach this point without support of the people who have been involved over the last four years.

A large share of this gratitude must go to my supervisory team Prof. Karl Tuyls, Dr. Elizabeth Sklar and Dr. Daniel Hennes. To Karl, I owe beside the thanks for great guidance, insight and research ideas, also the opportunity and trust to peruse my PhD in the first place and for being a great mentor in and out-side the research. To Elizabeth and Daniel I owe thanks for their ability to provide me with new concepts and research angles to progress my research in the right direction. I would also like thank my advisers Dr. Clare Dixon, Prof. Frans Coenen and Prof. Simon Parson to provide a different perspective to my research and confirming the research's trajectory.

Also a big thanks to my friends and team-mates of the smARTLab robocup team: Richard William, Daniel Claes and Joscha Fossel, for having the funniest, but more exhausting programming session I will hopefully ever have and taking the win in Brazil, Portugal and Germany. Also to my other fellow PhD students Eric Schneider, Jeffery Rafael, Gabrielle Dos Santos, Matoula Kotsialou, Elisa Cucco, Richard Klima, Paul Gainer, Maryam Kamali and Dr. Daan Bloembergen for going through the same process and providing the needed distracting in coffee breaks, after hour socialising and the annual Sunday cinema breaks.

Outside the University, I would like to thank my childhood friends: Annika, Julia, Markus, Sascha and Teresa, who are still keeping contact with me, even when the thesis writing made me temporally unavailable.

Finally and most importantly to my family. Where my father, Gerd sparked my interest in technology early on and still supports and implements every unconventional idea I have, with me. To my mother Ursula for being the best and supporting me even if I doubt myself. To my brother Julian, for being a good friend and for being there, when I need distractions in form of sport or riddles. And to the whole family for being my support system.

Chapter 1

Introduction

1.1 Bio Inspired Multi-robot Coordination

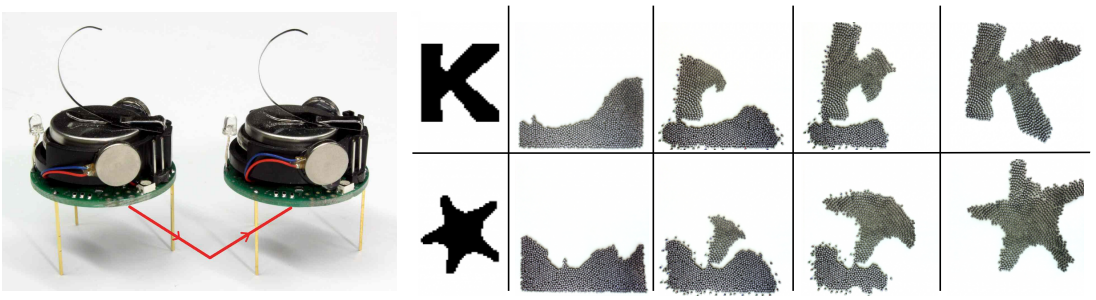
In recent years, there has been a rapidly growing interest in using teams of mobile robots for automatically surveilling environments of different types and complexity. This interest is mainly motivated by the broad spectrum of potential civilian, industrial and military applications of multi-robot surveillance systems. Examples of such applications are the protection of safety-critical technical infrastructure, the safeguarding of country borders, and the monitoring of high-risk regions and danger zones which cannot be entered by humans in the case of a nuclear incident, a bio-hazard or a military conflict. Triggered by this interest, today automated surveillance is a well-established topic in multi-robot research, which is considered to be of particular practical relevance [42, 64, 67]. An important topic in the multi-robot field is the coordination of large groups robots/swarms. This could entail futuristic scenarios like the interconnection of simple micro robots to form any desired structure or formation to solve a certain task. Figure 1.1 illustrates a robot-platform based on this idea. Also use-cases for space-debris cleaning [65] or the exploration of planets and moons are possible applications in the future. Advantages of employing greater groups are the parallelisation of tasks and

FIGURE 1.1: Reconfigurable ATRON modules

the systems robustness, since it suffers less from a single failure, compared to systems relying on one or few individuals. Furthermore, they are able to overcome the individual limitations, e.g. payload, computational and sensing limitations, by combining the capabilities and strengths of the whole group. But with those advantages comes

also the problem and complexity of controlling all units. This can either be done in a centralised or decentralised fashion. In the former approach, a centralised computing unit, able to observe the states of all robots, plans and communicates the best action to each individual. These actions are than received and executed by each agent to reach a global goal. In order to provide a fully observable state, these systems apply sensors like top-down or motion capture cameras, which can observe the states of all robots at once. Alternatively, this can also be accomplished by observing the state individually and communicating it to the central control unit. The major issue with these kind of approaches is that the global state is sometimes not observable by a single unit, because there is no sensor or device, for this specific environment capable of capturing this kind information, or a stable communication link between all units is not possible, because of the sheer size of the swarm. Additionally, a critical problem of this approach is that all units rely on the input of the centralised controller, which is therefore the systems single point of failure. Meaning, the system is not able to perform if the communication or the main controller malfunction. This makes decentralised systems more interesting, instead of having one or a couple of sophisticated agents responsible for the planning, these kind of systems build on the principle that the planning is done individually by each agent. In order to reduce complexity, the decision process of most systems relies on local observations and communication of an agent, instead of employing a global state of all agents combined. In general, decentralised approaches aim, in terms of hardware and algorithms, for a simple and robust implementation, since complicated sensors and high computational calculations require expensive robot platforms. Many approaches in this field taking inspiration form natural swarms, as they tend to follow simple behaviour rules on an individual level, to accomplish complex cooperative tasks. One popular example was proposed by Rubenstein et. al. 2014 [86], where a complex group formation was achieved, on a low cost robot swarm, by employing a simple rule decision process. The robots in this approach apply basic edge following and infra-red line of signed communication to estimate positions and shape the swarm in the desired formation (see Figure 1.2).

Many researchers, draw inspiration from observations of social insects such as ants and bees. These insects have evolved over a long period of time and display remarkable behaviours that are highly suitable for addressing the complex tasks that they are facing. Swarm optimization algorithms, like ant colony optimization (ACO) (Dorigo et al., 2006b [29]), rely on pheromone trails to mediate (indirect) communication between agents. This is based on the principle, that ants leave pheromone trails behind when they are looking for food and other ants are attracted by those trails. If an ant finds a food source, it will transport food between the location and the nest until all the food source is completely depleted. Since the pheromones decay over time, shorter routes will converge towards a stronger pheromone trail than longer ones, ants are therefore attracted by efficient solutions. This is illustrated in Figure 1.3a. Considering the ants have no specific recipient and they convey information indirectly via the placement of

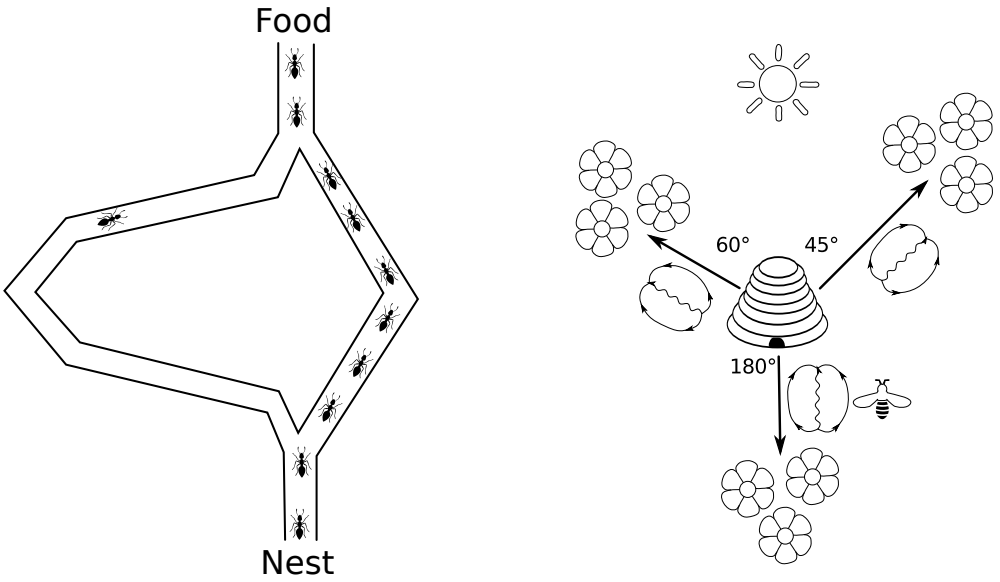


(A) (B)

FIGURE 1.2: Swarm formations with 1000 kilobots. A: Illustrates the kilo-robot platform. It employs three rigid legs in combination with vibration motors for movement and uses infra-red emitter and receive for direct communication. B: Illustrates the progression of the swarm formation. The robots start in a random group formation and converge slowly towards the desired shape.

pheromone, this type of communication is defined as indirect communication. Beside the pheromone attraction, have ants a minor probability of choosing random paths, this allows them to discover new solutions and they are able to reroute blocked paths, making their strategy overall extremely robust.

Such insect-inspired multi-agent research has also opened the possibility of applying some of these techniques to robotic systems, i.e., swarm robotics (Dorigo and Roosevelt, 2004 [33], Sahin, 2005 [88]). Swarm robotic systems show potential for different kind of applications, e.g. search and rescue, surveillance and patrolling. Other application areas include exploration and identification of hazardous environments (e.g., nuclear plants and fire detection), wireless sensor and robot networks, space exploration, etc. Though easy to simulate, artificial pheromones are hard to bring into real life robotic applications as pheromones need to be deployed and sensed by robots while they decay over time. Recently, non-pheromone-based algorithms were developed as well (Lemmens, 2011 [69], Dorigo et. al., 2006 [50]). Such algorithms are inspired by the foraging and nest-site selection behaviour of (mainly) bees. Generally speaking, bees explore the environment in search for high-quality food sources and once returned to the hive, they do a wiggle dance to communicate and describe the location of the source. By using this dance, bees recruit other colony members for a specific food source (Figure 1.3b). The more bees adopt a certain transportation path, the more bees will eventually perform the same dance. Since few dances will not attract enough bees, the best transportation path will eventually prevail. These kind of approaches are attractive in the field of multi-robot coordination since they build on a simple rule-based movement behaviour in combination with direct or indirect communication principles.



(A) Ants follow pheromone trails between nest and food source. Since shorter routes can be traversed quicker, their pheromone density is higher compared to longer routes. This attracts consequently more ants and the swarm converges towards an optimal solution.

(B) Bees employ a wiggle-dance to convey information about discovered food sources. The angle relative to the sun and the duration of the wiggle inform cooperating bees about the direction and distance to a specific food source.

FIGURE 1.3: Behaviour of social insects.

1.2 Research Questions and Contributions

Bio-inspired coordination strategies are interesting for multi-robot coordination, since they are mostly based on simple rule-base behaviours, which makes them robust and scalable. During this thesis we are focusing on ant and bee-inspired coordination, which emergences mostly from their communication principles, e.g. pheromone trails or pheromone propagation. Ant communication is refereed to as indirect communication, since they are conveying information by placing pheromones into the environment. The messages have therefore no direct receiver and are only influencing other agent based on their pheromone dosage and placement location. Bee apply both direct and indirect messaging, e.g. they pass information via pheromone propagation through the hive, where the pheromone is passed actively from one bee to another. During this thesis we investigate and propose techniques to apply these principles to the field of multi-robot coverage and examine their differences, strengths and weaknesses in simulation.

Modern robot hardware and sensors are still limiting the application of bio-inspired coordination/coverage strategies in a real-world setting, since their communication strategies e.g. placement of pheromone trails, are still difficult to implement. Additionally are these strategies designed to emerge their potential through the size of the robot team. Meaning, an individual is limited in its capabilities and is mostly not able to accomplish the task on its own, only through simple cooperation within the group, emerges the

potential of the algorithm. In order to provide the size required and to compensate single failures, most applications aim to reduce the cost and consequential the computing power of each robot. These limitation complicate simple, but computational expensive robotic tasks, like e.g. collision avoidance and sensing/localising of other robots. The former functionality is particularity necessary for flying vehicles since crashes between these robots can damage or break these robots. As part of this thesis, we tackle the problems of providing these capabilities with low-cost sensors to Micro Aerial Vehicle (MAV) platforms in real-world applications. In order to handle the low-computational power, we investigate and provide a machine learning based system, since the execution of a learned algorithm can be much quicker and reactive, than modern deterministic and random sampling based approaches. Machine learning, particular deep-learning has shown in the past years its capability of learning complex movement strategies and its accuracy in prediction tasks, but it is mostly used in clean and simulated environments. We investigate in our research how we can utilise these properties to provide a coordination framework, which allows the application of bio-inspired coordination principles on ultra-light weight robots platforms, specifically pocket-drones.

The research questions, that the work in this thesis aims to answer, are the following:

Research Question RQ1: To what extent can we devise appropriate algorithms for stigmergy-based/bio-inspired multi-robot exploration/coverage, in simulation and ground-based robots?
Research Question RQ2: What are the advantages of direct and indirect communication and to what extent can they be combined?
Research Question RQ3: Given algorithms derived from RQ1 and RQ2, to what extent does the environment shape and size affect their performance?
Research Question RQ4: How can modern sensors and technology be applied to provide a bio-inspired coordination on pocket drones?
Research Question RQ5: To what extent can deep learning be beneficial for the performance of the hardware discovered in RQ4?

In context of the research questions the following is a summary of the key contributions of the work presented in this thesis:

1. Chapter 4 presents a novel bee pheromone inspired coverage approach called BeePCo (Bee Pheromone Coverage). It is designed for achieving maximal sensor coverage, applicable in the context of mobile wireless sensor nodes, aiming to answer **RQ1**.

2. Based on the contribution of Chapter 4, we developed a new coverage approach called HybaCo (Hybrid Bee and Ant Coverage). It combines direct and indirect communication to provide an improved continues coverage behaviour. The approach is described in Chapter 5 and aims to answer **RQ1** and **RQ2**.

3. Aiming to answer **RQ3**, Chapter 6 is conduction a case study to determine to what extent the environment shape and size affects the performance of coverage approaches, more specifically BeePCo and HybaCo.

4. The final contribution consolidates multiple machine learning techniques and modern communication technology to provide a local sensing framework for multi-robot coordination of aerial pocket drones. These technologies include Ultra-wideband (UWB) communication and deep neutral networks. The developed system is shown in Chapter 7 and aims to answer **RQ4** and **RQ5**.

1.3 Thesis Outline

The remainder of this thesis is organised as follows. Chapter 2 introduces preliminaries including background in the behaviour of social insects, neural networks, reinforcement learning, monte-carlo localisation, ultra wideband distance estimation and quadrotor control. Chapter 3 presents related work in the field bio-inspired multi-robot coverage with an focus on the techniques this work is relying on.

Chapter 4 describes the proposed bee-pheromone based multi-robot coverage approach BeePCo (Bee Pheromone Coverage) and examines its strengths and weaknesses in experimental evaluations. BeePCo applies the concept of pheromone propagation in bee-hives, to distribute information and achieve maximal sensor coverage.

Chapter 5 introduces HybaCo (Hybrid Bee and Ant Coverage) a bee- and ant-based pheromone coverage approach and investigates its strengths and weaknesses in experimental evaluations. HybaCo employs ant-inspired pheromone trails and bee-inspired information propagation to continuously surveil an environment.

Chapter 6 describes an extensive case-study to determine to what extent the environment shape affects the coverage performance of proposed algorithms BeePCo and HybaCo.

Chapter 7 proposes a novel end-to-end system allowing bio-inspired multi-robot coordination on pocket drones. The developed system combines radio-based distance calculation with a recurrent neural network to estimate relative position between drones and employs a simple deep-q network (DQN) to provide low-level navigation and collision avoidance.

Finally Chapter 8 concludes the thesis with an overall analysis of the research as well as a discussion of the future work.

Chapter 2

Preliminaries

This chapter introduces the concepts, background and techniques used throughout this thesis. Section 2.1 starts by illustrating the behaviours of social insects, e.g. bees and ants and termites, many coordination algorithms are basing on. The knowledge of these concept is necessary to understand the related work in Chapter 3 and the approaches introduced in Chapters 4 and 5.

In Chapter 7 we propose a technique for multi-robots coordination for pocket drones based on local distance sensing. This relies on many computer science techniques and principles described in the following sections: neural networks (Section 2.2) in combination with reinforcement learning (Section 2.3), particle filter (Section 2.4), radio wave distance estimation (Section 2.5) and finally the control of quadrotors (Section 2.6).

2.1 Bio Inspiration

This section provides the required background knowledge of three different biological inspirations for the techniques proposed in Chapters 4 and 5. Subsection 2.1.1 starts by introducing stigmergic behaviour of ants, it continues with the foraging behaviours of honeybees in Subsection 2.1.2 and ends with the pheromone signalling mechanism for queen bee selection within honeybee colonies in Subsection 2.1.3.

2.1.1 Stigmergic Behaviour of Ants

The term "stigmergy" was introduced by French biologist Pierre-Paul Grass in 1959, he defined it as: "Stimulation of workers by the performance they have achieved." [101]. It describes a indirect coordination mechanism where agents communicate through the environment. The agents leave traces in the environment by taking an action this simulates the performance of the next action. This can either effect the same agent or different agents. Meaning, subsequent actions influence/build on each other. In the context of ant, stigmergy works as follows: ants deposit a pheromone trail on the path they take during travel. Using this trail, they are able to navigate toward their nest or food and communicate with their peers. More specifically, ants employ an indirect recruitment

strategy by accumulating pheromone trails. When a trail gets strong enough, other ants are attracted to it and will follow this trail toward a food destination. The more ants follow a trail, the more pheromone is accumulated and in turn the trail becomes more attractive for being followed. This is known as the autocatalytic process. Since long paths take more time to traverse, it will require more ants to sustain a long path. As a consequence, short paths will eventually prevail, see Figure 2.1c.

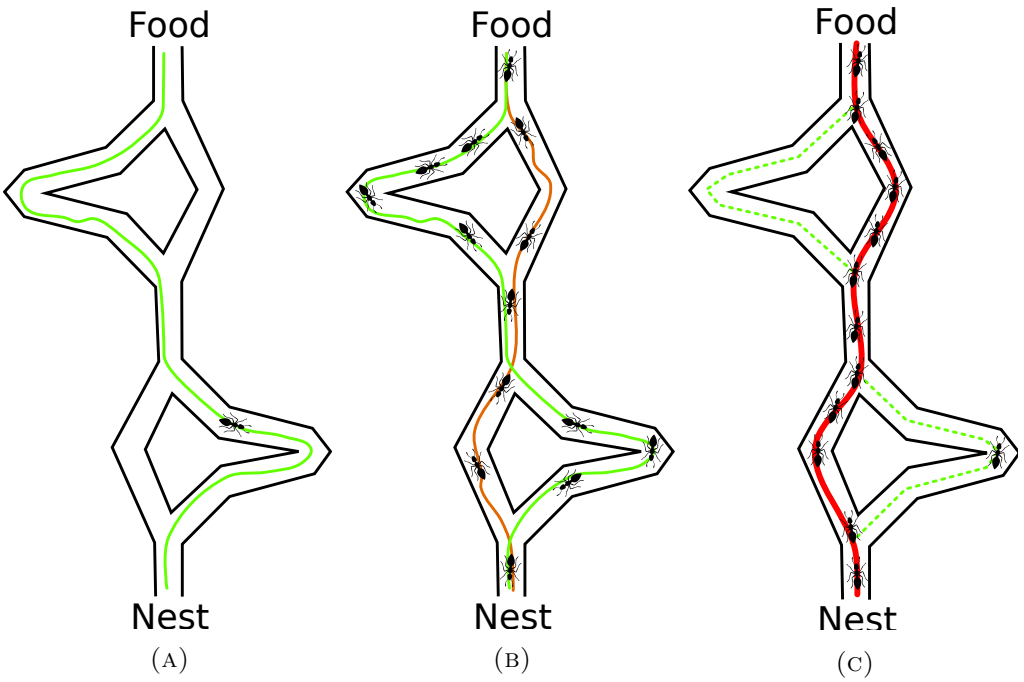


FIGURE 2.1: A: The first ant finds a food-source, than the ant reaches the Nest, leaving a pheromone trail. B: Other ants follow the one of four possible paths. C: The ants are following the shortest path.

2.1.2 Foraging Behaviour of Honeybees

Foraging honeybees display two types of behaviour, i.e., recruitment and navigation. In order to recruit other colony members for food sources, honeybees inform their nest mates of the distance and direction of these food sources by means of a waggling dance performed on the vertical combs in the hive. This dance (i.e., the bee language) consists of a series of alternating left-hand and right-hand loops, interspersed by a segment in which the bee waggles her abdomen from side to side. The duration of the waggle phase is a measure of the distance to the food. The angle between the sun and the axis of a bees waggle segment on the vertical comb, represents the azimuthal angle between the sun and a target location, i.e., the direction in which a recruit should fly (see Figure 2.2a). Other members of the colony can adopt the "advertisement" for a food source. The decision mechanism for adopting an "advertised" food-source location by a potential recruit is not completely understood. It is considered that the recruitment amongst bees is a function of the quality of the food source. Different species of social

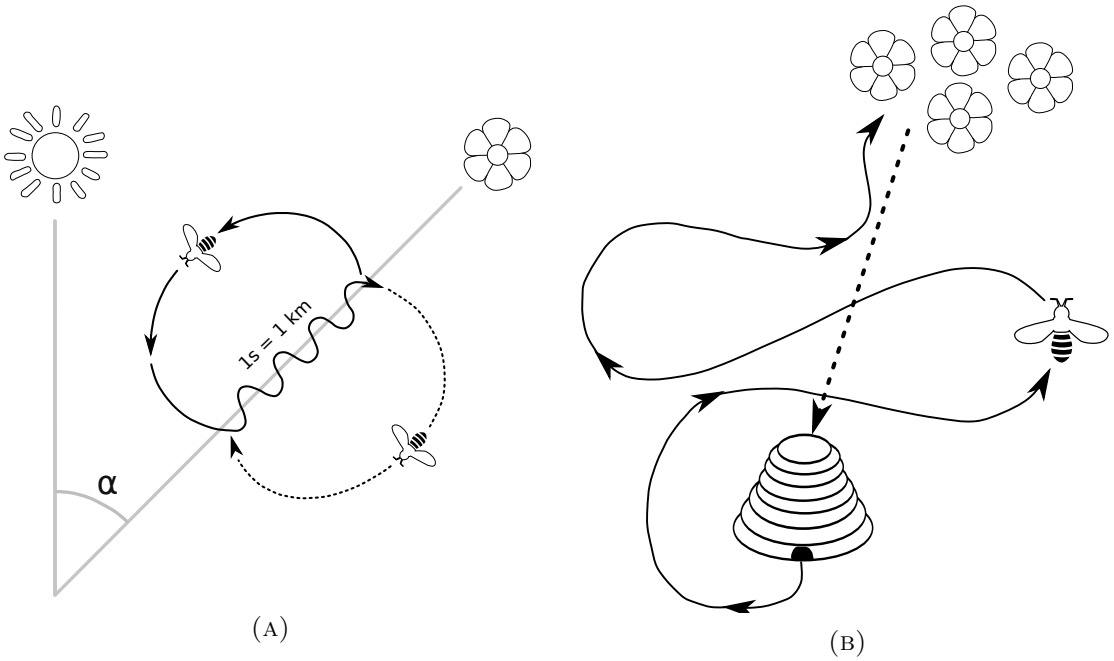


FIGURE 2.2: A: Shows the bee-dance. The angle relative to the sun indicates the direction and the duration of the wiggle informs of the distance to the food source. B: Shows the path integration to estimate the shortest path back to the hive.

insects, such as honeybees and desert ants, make use of non-pheromone-based navigation. Non-pheromone-based navigation mainly consists of Path Integration (PI), which is the continuous update of a vector by integrating all angles steered, and all distances covered [24]. A PI vector represents the insects knowledge of direction and distance towards its destination. To construct a PI vector, the insect does not use a mathematical vector summation, but employs a computationally simple approximation [24]. Using this approximation, the insect is able to return to its destination directly. More precisely, when the path is unobstructed, the insect solves the problem optimally. However, when the path is obstructed, the insect has to fall back on other strategies such as exploration or landmark navigation [23, 25] to solve the problem. Obviously, bees are able to fly, i.e., when they encounter an obstacle, they can mostly choose to fly over it. However, even if the path is unobstructed, bees tend to navigate over the entire path using landmarks. The landmarks divide the entire path into segments and each landmark has a PI vector associated with it. This behaviour decreases navigation errors and ensures robustness.

2.1.3 Pheromone Signalling Behaviour of Honeybees

The queen bee selection mechanism in honeybee colonies is used to orchestrate the colony by assigning responsibilities to each individual. [85] explains the process of larvae differentiation in beehives as an example of such orchestration. Bees have developed a special hormonal system to ensure every beehive has a queen, which maintains the stability of the colony and orchestrates the behaviour of all other bees. Throughout its life, a queen bee stimulates a pheromone called Queen Mandibular Pheromone (QMP),

which makes the worker bees aware of its presence as queen. This hormonal mechanism works as follows: the worker bees lick the queen bee and pass the pheromone on to the others. This pheromone is stimulation worker activities e.g. cleaning, building and foraging, but it also suppresses the urge to rear new queens. If the queen is old or sick (low pheromone) or is dead (no pheromones), the workers are alarmed and driven to rear new queens, they start feeding a few larvae with so called royal jelly containing royalactin protein [85]. The workers will also start building a so called queen cells around the larvae, in this specific case these queen cells are called an emergency cells. Royalactin protein induces the differentiation of honeybee larvae into a new queen. The queen which hatches first will kill the other queens in their cells, if two queens hatch at the same time they will fight to the death. This means as long as worker bees keep receiving the pheromone, they will be aware that there is a queen bee to orchestrate the colony and are not likely take action towards rearing a new queens. The algorithm BeePCo proposed in this Thesis is based on this principle, for more detail see Chapter 4.

2.2 Neural Networks

Artificial Neural networks (ANN) are a common machine learning concept with the ability to learn linear and non-linear relationships between input and output data. They are used in a varity of research field e.g. vision, speech recognition and playing games. This Section describes the basic idea of neural networks (Subsection 2.2.1), illustrates network topologies relevant for this thesis (Subsection 2.2.2), shows how these networks are trained (Subsection 2.2.3) and finally goes in more detail about special network cells which enable the network to learn outputs from a sequential input data (Subsection 2.2.4). All those techniques are employed in the distance-based multi-robot coordination on pocket, proposed in Chapter 7.

2.2.1 Basic Concept

ANNs are derived from biological neural networks that constitute human/animal brains. They are composed of artificial neurons also known as “nodes”. These neurons are connected via weighted edges, where the weight indicates the importance of the connection between nodes inside the network. Depending on the network type, the connections are either directed or non-directed (see Subsection 2.2.2). Usually an ANN has three types of neurons: input-, hidden- and output-neurons. The input neurons/nodes are the nodes located in the first layer of the network, the output-nodes are in the last layer and the hidden-nodes are in between those two (Figure 2.3a). The network receives the input-data as an array of numbers at the input layer, defined as $x(n)$, with n as the size of the array. Each input is multiplied by its weight. The weighted inputs are than summed up and a bias is added, to avoid a sum of zero. Finally, this sum is than transformed, by a mostly not linear activation function, in a so called activation value. The activation is than propagated in this fashion through the network until it reaches the last layer to

represent the estimated output. This process is illustrated in Figure 2.3b. There are

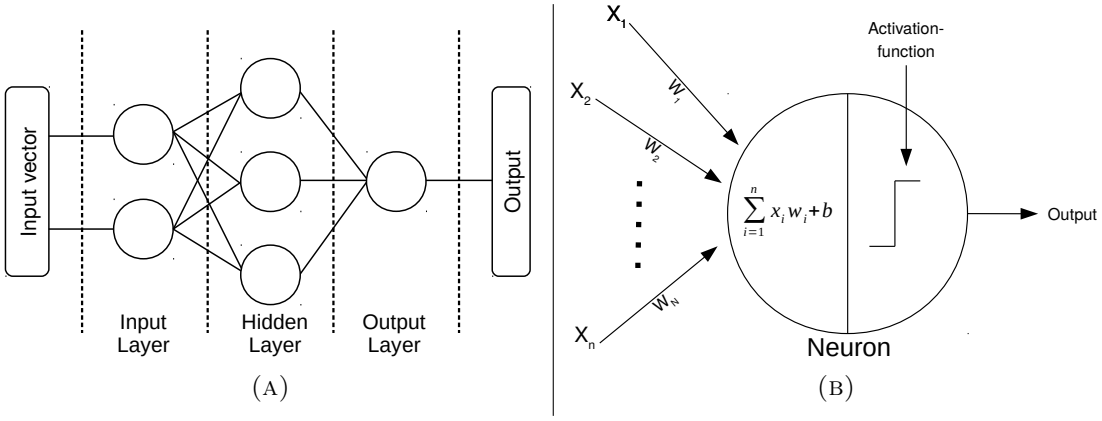
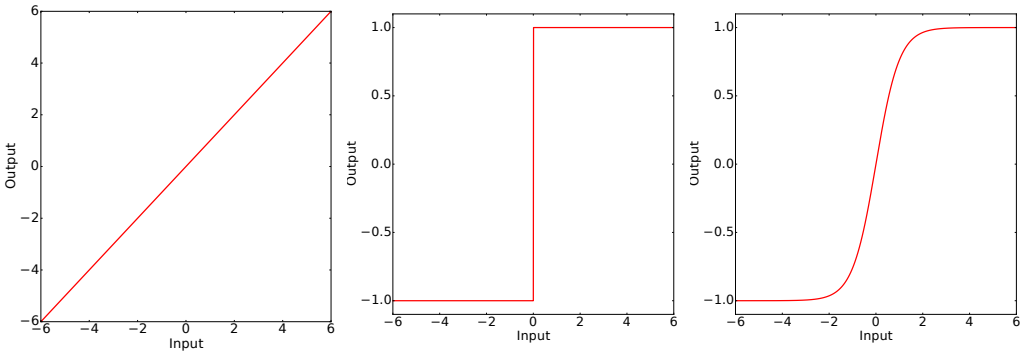


FIGURE 2.3: A: Shows the general structure, the input-layer and the output-layer of an ANN. B: illustrates in more detail the operations inside a singular neuron.

different kinds of activation functions, depending on the networks purpose, they can e.g "squash" the input to a output between -1 and 1 or transform the value to a binary output of zero or one. Most commonly used activation functions are threshold, linear and tan hyperbolic sigmoidal functions(nonlinear), shown in Figure 2.4. The goal of the learning is, to adapt the weights in a way that the input data corresponds to a desired output, this is described in Subsection 2.2.3.



(A) Linear (B) Threshold (C) tan hyperbolic sigmoidal

FIGURE 2.4: Example of activation functions.

2.2.2 Topologies

There are many different possible network configuration, this section is focusing on the topologies important in the context of this thesis.

Feed-Forward Network

A feed-forward neural network (FNN) is a directed graph where the signal can only travel from the input layer to the output layer without any recurrent links. The data is flowing throw the network with the steps described in Section 2.2.1. Figure 2.5 shows a simple example of a FNN network.

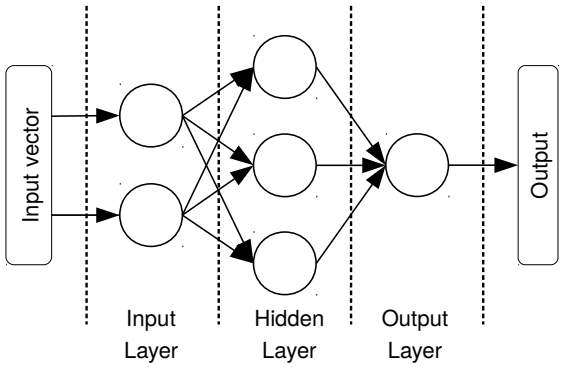


FIGURE 2.5: Feed-Forward Neural Network

Recurrent Neural Network

A recurrent neural network (RNN) is similar to a FNN, but besides the normal forward links it can also have links where the output is connected the input of the same node or nodes from previous layers. These recurrent connections allow the network to hold information across inputs. These connections can be thought of as similar to memory allowing the network to remember the previous activation value. This can be useful if the network has to handle sequential data e.g. in text auto completion, guessing the next word after a sequence of input words. It is common in RNNs to use special memory cells which have a specific mechanisms to protect their internal memory from losing essential previous inputs, allowing them to handle longer sequences. The most common cells are Long Short-Term Memory (LSTM) [55] cells and Gated Recurrent Units (GRU). In course of this thesis we employ a recurrent network with LSTM-cells, the basic principle of these cells is described in Section 2.2.4.

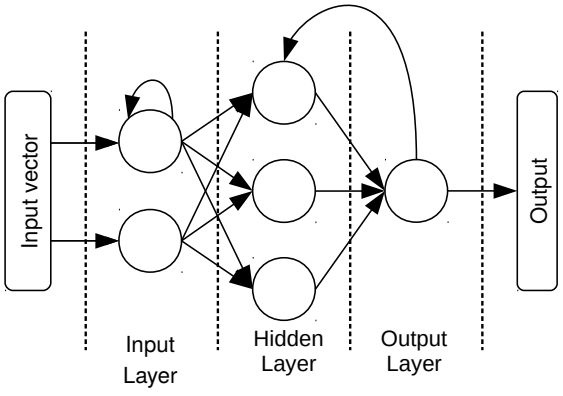


FIGURE 2.6: Recurrent Neural Network

2.2.3 Training

There are many types of learning for neural network, this Section is focusing on supervised learning. In supervised the training set contains input-data with their desired output. The aim of this training is to adjust the weights of the network so that the error between the networks and the actual output is minimised. The error of the network can

be seen as a optimisation function parametrised by the weights of the network. In order minimise this function, gradient descent optimisation algorithms are usually applied. These backpropagate the error back through the network to calculate the derivative of the function, with respect to the weights and biases, by applying the chain rule [5, 34]. This results in direction of the slope pointing away from the minimum (local). The weights are updated in the opposite direction of the slope to minimise the error. The step size is determined by a learning rate. For more detail about backpropagation see [92].

2.2.4 Long Short-Term Memory

LSTM-cells are capable of storing information, allowing the network to remember information many steps back. The cells are able to make decisions about when to store and read these informations, this is accomplished by opening or closing gates inside the cell. The gates are analog with the ability to parity "open" and "close". Different to normal memory, LSTM-cell data is differentiable, and therefore suitable for backpropagation. The gates mentioned, react on the signal received and open/close depending on the passed information. This depends on weights which will adjusted during the learning process. Figure 2.7 illustrates the structure of a LSTM-cell. The cell contains three gates for controlling the information flow. These are input-, forget- and output-gate. Mathematically these gates returns a numerical value between zero and one, which is than multiplied with information to partially or completely allow or deny the flow of information. The three gates have the following purpose:

- **Input-gate:** Determines to which extent a new value flows into memory.
- **Forget-gate:** Controls the importance of the current value in memory.
- **Output-gate:** Determines the flow of the resulting activation value.

For more information about LSTM see [55].

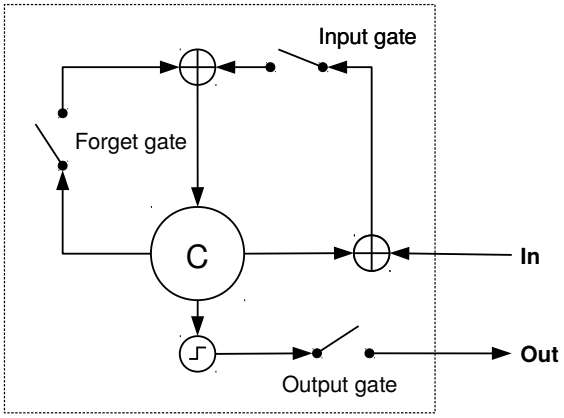


FIGURE 2.7: LSTM-cell

2.3 Reinforcements Learning

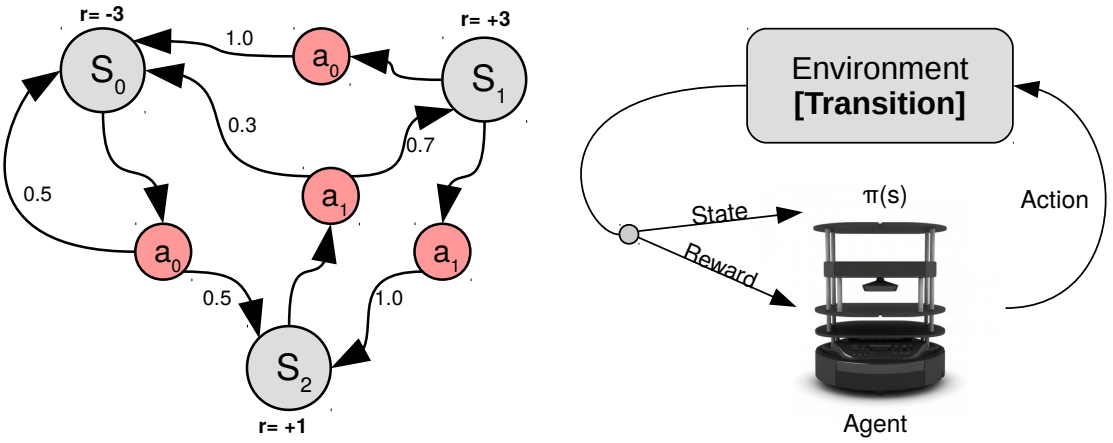
Reinforcement Learning (RL) is an area in machine learning focusing of maximising an agent sum of rewards, based on its actions in an given environment. The environment is usually formulated as a Markov Decision Process (MDP), but different to other techniques reinforcement learning doesn't require knowledge about the MDP itself. This makes it suitable for bigger and complex environments, where designing of a MDP before-hand is not feasible. This Section will give an general description of reinforcement learning and the Markov Decision Process in Subsection 2.3.1 and will go in more detail about a reinforcement learning related technique called Deep Reinforcement Learning in Subsection 2.3.2. This builds a foundation for the proposed technique introduced in Chapter 7.

2.3.1 Overview

Markov Decision Process (MDP) is the basic model Reinforcement Learning (RL) is building on, it is defined as follows:

- $\mathbf{S}$: Set of possible states.
- $\mathbf{A}$ or $\mathbf{A}(\mathbf{s})$: Set of actions or set of actions available in state $\mathbf{s}$
- $\mathbf{T}(\mathbf{s}, \mathbf{a}, \mathbf{s}') = \mathbf{P_r}(\mathbf{s}'|\mathbf{s}, \mathbf{a})$: Probability to transition to state $\mathbf{s}'$ under the condition of being in state $\mathbf{s}$ and taken action $\mathbf{a}$.
- $\mathbf{R}(\mathbf{s})$, $\mathbf{R}(\mathbf{s}, \mathbf{a})$ or $\mathbf{R}(\mathbf{s}, \mathbf{a}, \mathbf{s}')$: The reward a agent receives for entering a state $\mathbf{s}$, the reward a agent receives for being in a state $\mathbf{s}$ and taking action $\mathbf{a}$ or the reward an agent receives for selection action $\mathbf{a}$ in state $\mathbf{s}$ and transition to state $\mathbf{s}'$.

A MDPs actions can either be stochastic, where the agent has a certain probability to transition to a certain state $'\mathbf{s}$ by execution action $\mathbf{a}$ in $\mathbf{s}$, this means executing an action can have multiple outcomes and all the transition probabilities of a state action pair have to add up to 1.0. The alternative are deterministic actions, where the transition probability is 1.0 and the agent will always transition to the same state $\mathbf{s}'$ by execution the action $\mathbf{a}$ in state $\mathbf{s}$. The actions available can depend on state and restricts them to a subset of $\mathbf{A}$. Rewards can be immediate, but also delayed. Delayed means that the agent only receives a reward after executed a sequence of actions and reached a certain state. A good example is a path finding problem where the agent only gets a reward after it arrived at the goal. The aim of reinforcement learning is to learn a policy $\pi(\mathbf{s})$ which returns the optimal action $\mathbf{a}$ given the current state $\mathbf{s}$. In order to obtain this policy, a reinforcement agent is continuously interacting with its environment. It's executing actions and observing the resulting rewards. In order to act near optimally, the agent must reason about the long term consequences of its actions (i.e., maximize future income), although the immediate reward associated with this might be negative. The advantage of reinforcement learning approaches is that they are able to learn a policy



(A) (B)

FIGURE 2.8

without knowing the reward and transition function. These approaches can generally be divided in two categories: 1) Model-based approaches, which build models for reward- and transition-function based on the gathered observation and applying value/policy iteration to solve the problem. 2) Model-free approaches which derive the policy directly from the gathered data without building a model before hand. One of the most popular model-free approaches is Q-learning, it's also the basic idea behind Deep Reinforcement Learning, which is employed in this work and described in the following Subsection 2.3.2.

2.3.2 Deep Reinforcement Learning

As mentioned before, Q-learning [100] is a model-free approach with the aim to learn a optimal action-selection policy by interacting with the environment. Q-learning in particularly provides the expected utility-value for every state-action pair. After the learning is finished the optimal policy is simply the selection of the action with the highest utility value in a given state. The function returning the utility is call the Q-function. In the general the environment is parametrised by ε. During training, the agent will select an action a_t, at time-step t, transiting it from state s_t to s_{t+1} and receiving the reward r_t. This experience $e_t = (s_t, a_t, r_t, s_{t+1})$ is stored in the replay buffer of the agent. This data-storage is parametrised as D and mostly limited to a fixed size of N. This means the oldest experience is dropped when the storage-size exceeds N. The selection of the action can either be random or based on the current Q-function. A common selection-strategy is ϵ-greedy, where the agent selects a random action with the probability of ϵ and a value-function based action with the inverse chance of $1 - \epsilon$. Typically ϵ is initialised with a high value and reduced over time. This allows for a better exploration of the action space in the beginning. In the update-phase the agent will than draw a random mini batch of experiences from the replay-buffer in order to update the Q-function. In Q-Learning, $Q^*(s, a)$ is defined as the optimal value function for calculating the expected reward, for a given state-action pair, it's defined by the

recursive Bellman equation:

$$Q^*(s,a) = \mathbb{E}_{s' \sim \varepsilon}[r + \gamma \max_{a'} Q^*(s',a')|s,a] \tag{2.1}$$

Meaning, the Q-value of the state-action pair s and a, is the resulting reward r plus the maximal future reward, where all the consecutive actions are chosen optimally. The future reward is reduced by the factor $\gamma \in (0,1]$. One way to solve the problem, is to use the Bellman equation as in iterative update $Q_{i+1}(s,a) = \mathbb{E}[r + \gamma \max_{a'} Q_i^*(s',a')|s,a]$. Works proposing this technique converge to a optimal action-value function, $Q_i \to Q^*$ for $i \to \infty$ [100]. But this approach doesn't scale well. Instead [40] proposes to employ an approximator in a form of a neural network parametrised θ, to provide a value function Q to substitute the optimal Q-value-function so that: $Q(s,a;\theta) \approx Q^*(s,a)$. This network is also referred as Deep-Q-learning Network (DQN). It's trained in each iteration i by minimising a loss function $L_i(\theta_i)$,

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim p(\cdot)} \left[(y_i - Q(s,a;\theta_i))^2 \right] \tag{2.2}$$

where y_i is derived from the Bellman equation as $y_i = \mathbb{E}_{s' \sim \varepsilon}[r + \gamma \max_{a'} Q(s',a',\theta_{i-1})|s,a]$ and $p(s,a)$ is a provability distribution over states s and actions a. Differentiating the loss function with respect to the weights θ of the network, results in the following gradient used to update the weights:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim p(\cdot); s' \sim \epsilon} \left[\left(r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i) \right) \nabla_{\theta_i} Q(s,a;\theta_i) \right] \tag{2.3}$$

There are different methods to define a condition which determines the end of training. The simplest one is to set a fixed number of episodes, where an episode is a limited by a number of time-steps. In this case the agent interacts with the environment until the time-step limit N is passed or the agent reaches a terminal state, this is repeated for M episodes. The disadvantage of this approach is that this condition doesn't verify how good the policy is performing. Therefore a better method is validate the current policy every x iterations on a number of important scenarios. If the policy performs as expected or the accumulated reward doesn't increase any more, the training can be stopped. Depending on the use-case using the reward can be quite noisy. An other method is to check the Q-value of a set of states. The states are randomly chosen in the beginning and are frequently checked for the expected maximal Q-value returned from the current network. If the accumulated Q-value is not increasing over a longer period of time, the training can be stopped, since this is most likely an indication that the Q-function is close to optimum. Algorithm 1 illustrates the described learning process inside a simulation with M episodes and T time-steps per episode. For further details about Deep Q-learning Reinforcement, see [40, 74].

Algorithm 1: Deep Reinforcement Learning

Initialise the replay buffer D with size N
Initialise network with random weights θ
for episode = 1, M **do**
  observe state s_1
  for t = 1, T **do**
    chose random value for z
    if $z < \epsilon$ **then**
      select random action a_t
    else
      $a_t = \max_a Q(s_t, a; \theta)$
    end if
    Execute action a_t and observe r_t and the next state s_{t+1}
    Store experience $e_t = (s_t, a_t, r_t, s_{t+1})$ in D
    $s_t = s_{t+1}$
    Sample mini-batch of experiences (s_j, a_j, r_j, s_{j+1})
    Set $y_j = \begin{cases} r_j, & \text{for terminal } s_{j+1} \\ r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta), & \text{for non-terminal } s_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$ according to equation 2.3
  end for
end for

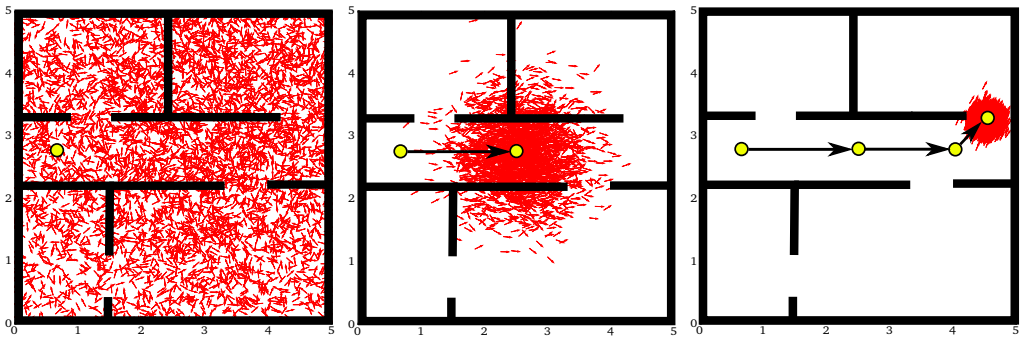
2.4 Monte Carlo Localisation

Monte Carlo Localisation (MCL) is a particle filters also known as Sequential Monte Carlo (SMC) methods, commonly used in robotics for localisation in known maps ([6, 43, 72]). A slightly modified version of this approach is applied during this thesis described in Chapter 7. Particle filters are designed to provide a conditional probability of states based on noisy observations. They present a distribution by a set of particles drawn from this distribution instead of presenting the distribution in parametric form. This form of representation is able to represent a much wider space of distributions, than for example Gaussians. Each particle $x_t^{[m]}$, in a set of size M, represents a possible state at time t. At each time step particles are updated by the control u_t (e.g. velocity) and afterwards weighted by its probability $w_t^{[m]} = p(z_t|x_t^{[m]})$ based on the observation z_t. The weight $w_t^{[m]}$ represent the relevance of its corresponding particle. Subsequently the filter draws M particles from the current set, and the probability of drawing a specific particle is defined by its normalised weight. This increases the probability to converge towards particles representing the actual state, each particle $x_t^{[m]}$ has the probability of its normalised weight $\frac{w_t^{[m]}}{\sum_{i=1}^{M} w_t^{[i]}}$ to be drawn. This means, higher weighed samples are more likely been drawn and the set is converging step by step towards particles which are more likely to represent the actual state. Algorithm 2 shows the mentioned process of the Monte Carlo Localisation (MCL). Figure 2.9 shows a simple MCL application in action. The red arrows represent the particles and the yellow circles show the actual path of the

Algorithm 2: Monte-Carlo Localisation

1: **MCL**(X_{t-1}, u_t, z_t, m)
2: $\bar{X}_t = X_t = \emptyset$
3: **for** m = 1 to M **do**
4: $x_t^{[m]}$ = **sample_motion_model**(u_t, $x_{t-1}^{[m]}$)
5: $w_t^{[m]}$ = **measurement_model**(z_t, $x_{t-1}^{[m]}$, m)
6: $\bar{X}_t = \bar{X}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
7: **end for**
8: **for** i = 1 to M **do**
9: draw i with probability $w_t^{[i]}$
10: add $x_t^{[i]}$ to X_t
11: **end for**
12: **return** X_t

robot in the real world. The robot in this Scenario knows the map and measures the environment with a 360 degree laser-scanner. In the beginning the particles are initialised randomly in the whole room (Figure 2.9a), but the more the robot moves and sensor-data is gathered certain particles are getting more and more probable (Figure 2.9b) until the cloud is converging to the actual position (Figure 2.9c). For further detail see [102].



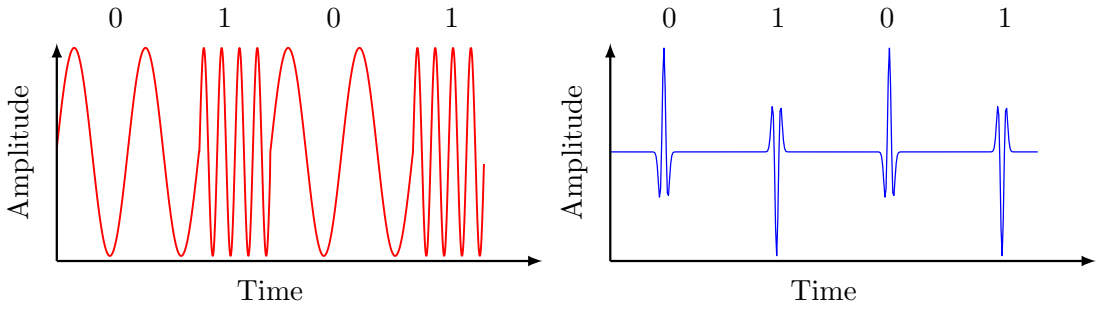
(A) (B) (C)

FIGURE 2.9: Monte Carlo Localisation: (A) shows the initialisation of the particles, (B) illustrates the particle states after the first moves and sensor updates and (C) presents the converging towards the actual robot position.

2.5 UWB Distance Estimation

This section gives an general overview over radio based distance calculation, more specifically for Ultra Wideband (UWB). Distance calculation in this context means to estimate the relative distance between two or more devices, it's mostly used for localisation. Multiple devices are placed at fixed positions in a environment, a mobile device is than tracked by estimating the distance to the fixed devices and applying simple triangulation. Radio waves are electro magnetic waves with a frequency between 3 kHz and 300 GHz. These waves are commonly used in wireless communication techniques e.g. WiFi (Wireless Fidelity) or bluetooth. Using wireless communication for distance calculation has different reasons: they are widely spread, low in cost, light weight and have the ability to penetrate walls. There have been a number of different ranging method proposed, one of the most well know methods is based on the fact that signal strength decreases over distance; this technique is called RSS (Received Signal Strength). RSS a simple and affordable technique, but it suffers from high inaccuracy especially indoors [12]. More accurate methods are time based and are mostly referred to as TOF (Time of Flight) or TOA (Time of Arrival). The basic idea of these methods is the employment of communication to calculate the distance between the two devices. The devices are sending messages back and forth, the distance is than approximated by multiplying the duration of the messaging process by the speed of travel (mostly light speed). The amount of messages necessary can be reduced by synchronising the device clocks, but this process is rather complicated especially with mobile devices. Common communication techniques like WiFi or Bluetooth are classified as narrow band radio transmission. Narrow band means that the frequency spectrum (bandwidth) in a channel of those media is quite small, common WiFi-standards have a bandwidth of 20-40 mHz. They use a sinusoidal wave as carrier wave and modulate its frequency to encode information, see Figure 2.10a. One mayor issue in narrowband systems is problem of multipath propagation where multiple replicas of the wave are reflected in the environment, overlapping or shifting of the original signal, therefore influencing the TOA-detection and distance accuracy on the receiver side, since start and beginning of the signal are harder to detect. There are techniques trying compensate these issues [56], but they require highly computational and complex algorithms. Ultra wideband techniques resolve this problem natively, instead employing a carrier wave they encode information in very short signal pulses, spreading the radio energy over a wide frequency band, therefore the name Ultra wideband (UWB). The short bursts of signals, with sharp rises and drops, makes the signals' starts and stops inherently easier to measure and the short duration of the signal reduces the probability of overlap (Figure 2.10b). These properties are the reason why UWB-solutions are gaining on popularity in TOA-implementations, there are less sensible to noise and provide a high accuracy.

The TOA-implementations itself are divided into two categories: time synchronised transmitter and receiver and unsynchronised transmitter and receiver. In the first scheme the clocks of all devices have to be in sync. The sender will then send a message



(A) Narrowband: Frequency modulation. (B) UWB: Signal pulses.

FIGURE 2.10: Signal encoding

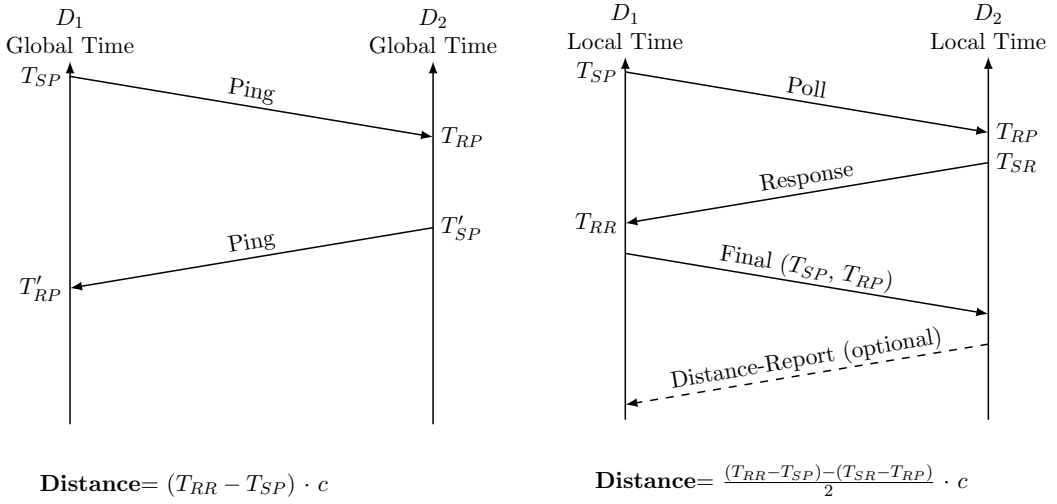
(ping) at a known time or will embed the transmit time into the message. By knowing the transmit time T_{SP} and the receive time T_{RP}, the recipient is now able to calculate the distance by multiplying the difference by c (speed of light):

$$D = (T_{RP} - T_{SP}) \cdot c \tag{2.4}$$

The drawback of this implementation is the difficulty of the time synchronisation. It is therefore mostly used for global position information, where most of the devices are positioned in known locations and physically wired together to provide the synchronisation. The mobile device is able to calculate the global time by knowing the transmit intervals of the stationary devices. The unsynchronised implementation avoids the synchronisation issues by sending a messages back and forth between two devices to acquire their distance information, this is known as two-way ranging. In this case if two devices, for example D_1 and D_2, require their current distance, a message (poll-message) is sent out by D_1 at its local time T_{SP} and is received by D_2 at its local time T_{RP}. D_2 response with a message at time T_{SR} which is received by D_1 at time T_{RR}. Lastly D_1 transmits the recorded times T_{RR} and T_{SP} to D_2. By subtracting the processing time of D_2 from the from the total time, D_2 is now able to calculate the distance as follows:

$$D = \frac{(T_{RR} - T_{SP}) - (T_{SR} - T_{RP})}{2} \cdot c \tag{2.5}$$

To reduce the message count, D_2 can now send the calculated distance back to D_1. The whole process is illustrated in Figure 2.11b. Additional methods like A0A (Angle of Arrival) [4] and TDOA (Time Difference of arrival) [52] are only suitable for global positioning with fixed devices.



Distance= $(T_{RR} - T_{SP}) \cdot c$

Distance= $\frac{(T_{RR}-T_{SP})-(T_{SR}-T_{RP})}{2} \cdot c$

(A) Time of Arrival (TOA) with synchronised transmitter and receiver.

(B) Time of Arrival (TOA) with unsynchronised transmitter and receiver.

FIGURE 2.11

2.6 Quadrotor Control

The work proposed in Chapter 7 is new multi robot coordination method based on local-sensing. This method is executed and tested on pocket drones or quadrotors, which are specified as Micro Aerial Vehicle (MAV). The following section will give an overview over dynamics and a typical feedback control system applied on MAVs, these principles are important for the later coordination of the drones. The material in this section is based on the technical report by Randal Beard [10].

2.6.1 MAV Coordinate Frames

In order understand dynamics described in Subsection 2.6.2 this section will introduce common coordinate frames used in the context of MAVs. The inertial frame $\mathcal{I}$ is a earth fixed with a arbitrary origin, most of the time the origin is defined as the start or home position of the MAV. Its x-axis points north, the y-axis points east and the z-axis points into the earth. The coordinate system of a MAV is described by the body frame $\mathcal{B}$, where the origin is located at the centre of mass of the MAV. The x-axis points towards the front of the MAV, the y-axis points to the right of the MAV and the z axis points downward from the MAV. The translation between the two is given by a rigid body transform ${}^{\mathcal{I}\mathcal{B}}T$ (Figure 2.12).

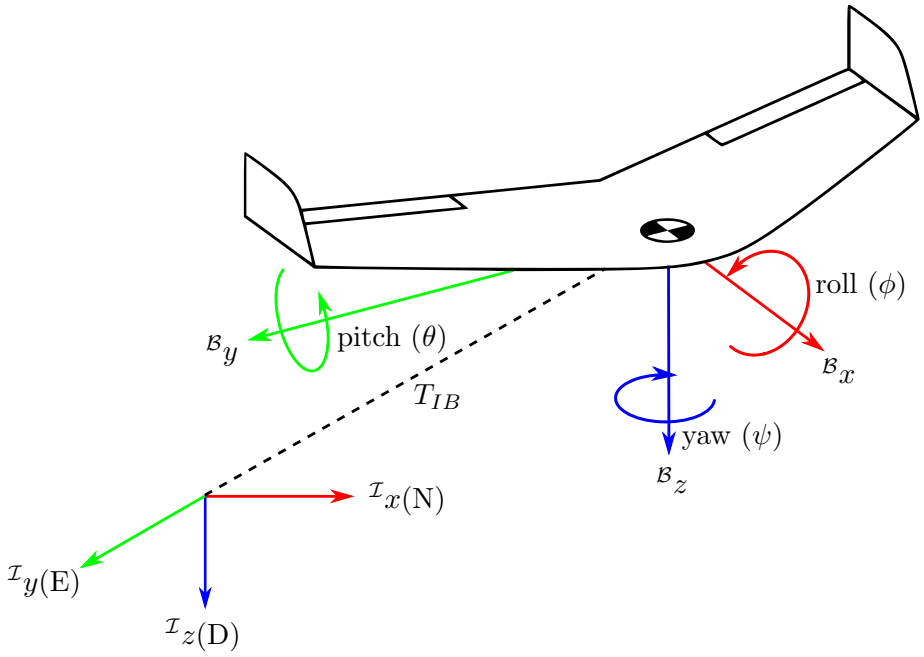


FIGURE 2.12: MAV Coordinate Frames.

Some works, particularly in the field of MAV control, introduce an additional frame to separate translation and rotation. For example, the vehicle frame is defined with its origin at the MAVs center of gravity and its axes aligned with the corresponding axes of the inertial frame.

2.6.2 Quadcopter System Model

Small quadrotor or also known as quadcopter are categories as Micro Aerial Vehicle (MAV), they are usually equipped with four equally spaced motors with fix pitch rotors to provide the lift. The common body shape is mostly squared with the main control unit (flight controller) placed in the center of gravity. The fight controller includes the processor and sensors required for the stable control of a quad rotor, discussed in Subsection 2.6.3. A Quadrotor's movement has 6 degrees of freedom, this means it can move freely along its three principle axis x, y and z. Additional it is also able to rotate around x, y and z, which are respectively called roll-, pitch- and yaw-axis, their rotation angle is parametrised as ϕ, θ and ψ (see Figure 2.12). The upward force F_i each motor M_i generates is controlled by altering the speed of the motor (Figure 2.13). The sum of these forces combined, make up to total force the craft is able to generate:

$$\mathbf{F} = F_1 + F_2 + F_3 + F_4$$

Increasing forces for different pairs of motors introduces torque around the center of mass (Figure 2.14), e.g. torque around the pitch-axis θ is given by:

$$\tau_\theta = l(F_3 - F_2 - F_1 + F_4)$$

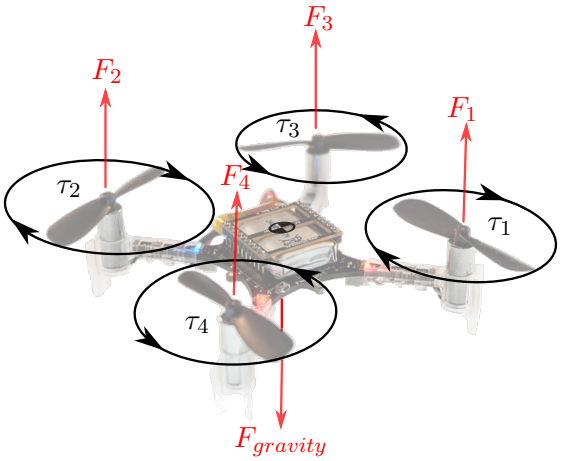


FIGURE 2.13: Quadrotor motor torque and upward-downward forces.

With l defined as the distance from the centre of mass to the motor. Similarly the torque around the roll-axis ϕ is defined as:

$$\tau_\phi = l(F_2 - F_1 + F_3 - F_4)$$

When a propeller is moving through the air it's exposed to drag which induces a yawing torques $(\tau_1, ...\tau_4)$ on the quadrotors body in the opposite direction of the propellers rotation. If all propellers would rotate in the same direction this yawing torque would cause the body to rotate in the other direction. Thus the motors are configured in a way to counteract this effect, motors $M1$ and $M3$ spin counter-clockwise and $M2$ and $M4$ spin clockwise. In this configuration one motor pair is cancelling out the yawing torque of the other and vice versa. The total yawing torque is defined as:

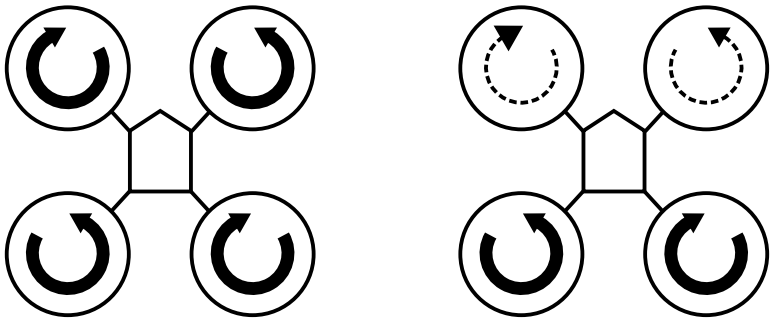
$$\tau_\psi = \tau_2 + \tau_1 - \tau_3 + \tau_4$$

The yawing torque of both pairs is in a equilibrium and the quadcopter will hold its current heading, when $\tau_\psi = 0$. The copter is in a stable hover if yaw-, pitch- and roll-torque are equal to zero: $\tau_\theta = \tau_\phi = \tau_\psi = 0$ and the total force $\mathbf{F}$ generated by the motors is large enough to counteract gravity. The forces and torques can be computed by taking various electromechanical and aerodynamic properties into account, however this is not necessary for control. Instead, given that produced lift and drag is proportional to the angular velocity of the motors, forces and torques can be express as:

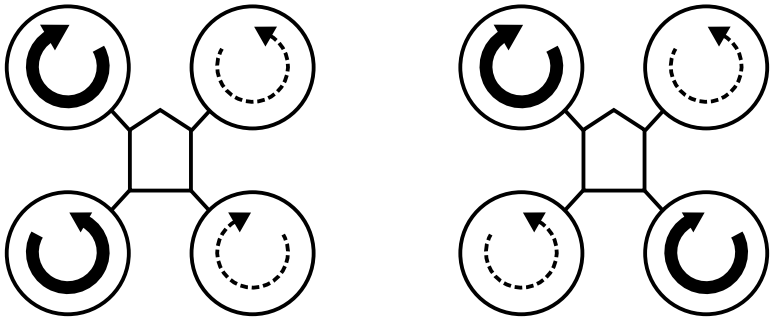
$$F_i = C_1\omega_i^2 \tag{2.6}$$

$$\tau_i = C_2 F_i \tag{2.7}$$

where ω_i is the angular velocity of motor i and C_1 and C_2 are constants that model the rotor characteristics, these can be determined experimentally. This gives the complete



(A) Vertical Climb (B) Pitch and Translate Forward



(C) Roll and Translate Right (D) Yaw Left

FIGURE 2.14: Quadcopter control: this diagram shows how varying the speeds of each motor results in a corresponding movement, thick arrows indicate increased speed. Note the coupled rotational and translations movements on the pitch and roll axes.

torque model as:

$$\begin{bmatrix} \tau_\psi \\ \tau_\theta \\ \tau_\phi \end{bmatrix} = \begin{bmatrix} -C_M & C_M & -C_M & C_M \\ -l & -l & l & l \\ -l & l & l & -l \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} \tag{2.8}$$

Additionally, gravitational force is acting on the quadrotor. The gravitation force in the vehicle frame is given by:

$$^{\mathcal{V}}F_g = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix}$$

where g is the acceleration due to earth gravity in m/s^2. In order to apply the gravitational force to the quadcopter model, it has to be transformed to the body frame:

$$^{\mathcal{B}}F_g = ^{\mathcal{V}} F_g\mathbf{R} = \begin{bmatrix} -mg\ sin\theta \\ mg\ cos\theta\ sin\phi \\ mg\ cos\theta\ cos\phi \end{bmatrix}$$

where $\mathbf{R}$ is a 3×3 rotation matrix of the rotation group SO(3), θ and ϕ are pitch- and roll-angles from the vehicle frame to the body frame. The forces can be decomposed in

force elements along each axis (F_x, F_y, F_z) by applying the following transformation:

$$m \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \mathbf{R} \begin{bmatrix} 0 \\ 0 \\ \mathbf{F} \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \tag{2.9}$$

In order to determine the angular accelerations, the moments of inertia have to be known, these represent the resistance to rotational acceleration of the body. They can be calculated by approximating the quadcopter as a dense spherical of mass m at the center of drone with a radius r and model the motors as point masses at a distance of l from the copter's centre. Under the assumption that the quadrotor is symmetric in all three axes an inertial matrix can be given by:

$$\mathcal{J} = \begin{bmatrix} j_x & 0 & 0 \\ 0 & j_y & 0 \\ 0 & 0 & j_z \end{bmatrix}$$

The inertia for the solid sphere and point masses is given as:

$$j_x = \frac{2mr^2}{5} + 2l^2m$$

$$j_y = \frac{2mr^2}{5} + 2l^2m$$

$$j_z = \frac{2mr^2}{5} + 4l^2m$$

Given the above the angular acceleration can be approximated by:

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} j_x^{-1}\tau_\phi \\ j_y^{-1}\tau_\theta \\ j_z^{-1}\tau_\psi \end{bmatrix} \tag{2.10}$$

Decomposing equations into their separate components gives the following dynamic model:

$$\begin{aligned} m\ddot{x} &= -\mathbf{F}(cos(\phi)sin(\theta)cos(\phi) + sin(\phi)sin(\psi)) \\ m\ddot{y} &= -\mathbf{F}(sin(\phi)sin(\theta)cos(\phi) - cos(\phi)sin(\psi)) \\ m\ddot{z} &= -\mathbf{F}(cos(\theta)cos(\psi)) + mg \\ \ddot{\psi} &= \frac{1}{J_x}\tilde{\tau}_\psi \\ \ddot{\theta} &= \frac{1}{J_y}\tilde{\tau}_\theta \\ \ddot{\phi} &= \frac{1}{J_z}\tilde{\tau}_\phi \end{aligned}$$

This simplified model is sufficient to model the motion of the quadrotor and helps to understand control applied in this thesis. It shows how the multi-rotor is able to accomplish its six Degrees of Freedom (DOF), linear movement in $x,\ z,\ y$ and rotational movement in $\psi,\ \theta,\ \phi$ with only four controllable actuators, motors $M1,\ M2,\ M3,\ M4$. Keeping this in mind a quadrotor is under actuated, meaning the six DOFs can't be controlled independently. If for example, a quadrotor is flying along the x-axis it has to be pitched forward to generate the required upward-forward thrust it requires.

2.6.3 Feedback Control

The previous section described the dynamics of a multi-rotor system, this section will give an basic overview over the feedback loop control typically applied in these kind of systems. In control theory there are two fundamental types of control loops: open-loop and closed loop control. Open loop control applies an action to a system without monitoring the output of this system. A simple example would be toaster, the controller executes the heating-action for a constant time, but it usually doesn't control the temperature or the color of the toast. A closed-loop control, also known as feedback control, on the other hand is exactly doing that. It compares the current system output of the system to a reference value, based on the difference between reference and actual value it will intensify or reduce the action. A common example is a thermostat in a normal household, it uses temperature sensor to measure the current state of the system and applies this information to maintain the room temperature at a desired level. Feedback control is formalised as follows: The current state $y(t)$ of the system at time t is estimated by the system sensors resulting in the measurement $z(t)$. These measurements are most of the time subject to noise parametrised by $\delta(t)$. Aim of the controller is to minimise the difference/error $e(t)$ between the measurement $z(t)$ and the desired state $x(t)$. The controller employs $e(t)$ to calculate a new action $u(t)$ which is passed on to the actuator. The action value will than, based on the systems dynamics, result in a new system state $y(t)$. This process can be seen as transfer function between $u(t)$ and $y(t)$ and is often called plant in control theory literature. The outcome of the plant is not only affected by the action, most of the time it is also influence by external disturbances $\epsilon(t)$. In context of MAV control $\epsilon(t)$ is i.e. the wind. The whole process is continuously repeated, the desired value $x(t)$ is not fixed and can be changed during execution. Figure 2.15 illustrates the flow of a general feedback controller. There are different variation of feedback control implementations, they differ mostly in the way they determine the action $u(t)$. One of the most popular mechanisms is the Proportional Integral Derivative (PID) controller. Like the name suggest its output is a combination of three terms. The proportional term is the easiest one and calculates the output proportional to the error as:

$$K_P e(t)$$

The constant K_P, the proportional gain, amplifies or reduces the direct influence of the error. A high value leads to a fast response, but can cause in overshooting of the

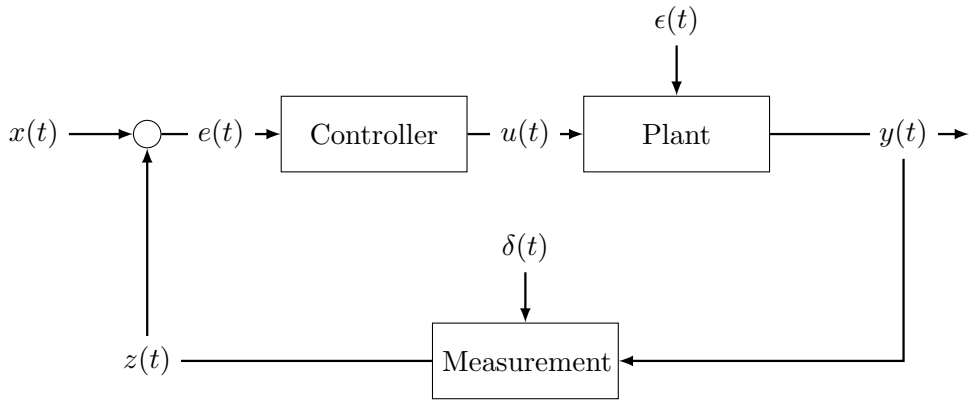


FIGURE 2.15: Block diagram of a typical feedback control problem

desired value and an oscillating behaviour. Lower gains can have an opposite effect an slow down the response or lead to almost no reaction, due to its inability to overcome external disturbances. The integral term (I-term) is a accumulation of pass errors, meaning if an error persist for a while, this terms will increase over time until it has overcome the disturbance. It is defined as follows:

$$K_I \int_0^t e(\tau) d\tau$$

An example scenario in the context of MAVs, would be a case where a drone wants to fly to the left but due to wind its staying at its position or drifting to the right. In this case the I-term will grow and raise the command-value until it will overcome the external influence. K_I is the integral gain factor to increase or reduce the effect of the I-term in the control. Combining only the proportional and integral term can in some use-cases be sufficient, those type of controllers are referred to as PI-controller. In other use cases this might not be enough and causes a slow response or a high frequents oscillation. The final term to compensate these issues is the derivative term (D-term), given by:

$$K_D \frac{de(t)}{dt}$$

The derivative term takes the error's change over into time account. This means, if the current error is less than the previous one, the control is going in the right direction and the D-term will start reducing the control-value to avoid overshooting. The effect of the reduction depends on the slope of the error change. This works also in the reverse direction, if the error is increasing the derivative term will boost the control-command for a faster reaction time. The complete PID-controller is given by the sum of the three terms:

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau) d\tau + K_D \frac{de(t)}{dt}$$

The performance of a PID-controller is highly dependent on the tuning of the control gains $K_P,\ K_I,\ K_D$. This is often done in a experimental fashion, but there are many

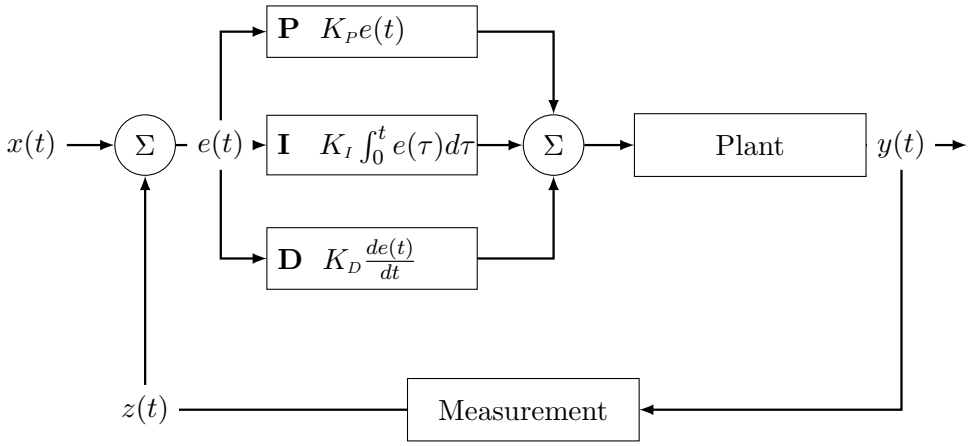


FIGURE 2.16: Block diagram of a Proportional Integral Derivative (PID) controller

heuristic or automatic methods for estimating valuable gains.

2.6.4 Quadrotor Angular Velocity Control using PID

The following section presents how the PID-controller is employed in the field of MAVs. General speaking there are three different state-properties to control: position, velocity and angular velocity. Angular velocity control is the most important one of the three, it operates on the lowest level and is essential for a stable flight, it's the foundation for position and velocity control. Position and velocity control are optional and can only be provided if the MAV possesses special sensors e.g. GPS, motion capture or optical flow sensors (see Subsection 2.6.5). The sensors providing the feedback measurements for angular velocity are called gyros and are necessary on any multi-copter hardware, they measure the rotation rate in degrees per second for the three principle axis: roll ϕ, pitch θ and yaw, it's represented by:

$$z(t) = (\dot{\phi}, \dot{\theta}, \dot{\psi})$$

The input for the controller is therefore defined as follows:

$$y(t) = (\dot{\phi}, \dot{\theta}, \dot{\psi}, th)$$

with $(\dot{\phi}, \dot{\theta}, \dot{\psi})$ as the desired angular rates and th is a baseline throttle command. The output is the throttle commands for all four motors:

$$u(t) = (\omega_1, \omega_2, \omega_3, \omega_4)$$

The throttle baseline th is not a desired set point, it is the value all motors should have, when all angle velocity inputs are zero and the drone is perfectly stable. The output of the controller is added to this baseline-value in order to achieve the desired angle-rate. The system described above is a Multiple Input Multiple Output control (MIMO) system, where a PID-controller is only designed as a Single Input Single Output (SISO)

mechanism. In order to solve this problem, common MAV angular velocity controller consist of a set of PID-controllers, one for each degree of freedom, parametrised as: PID_ϕ, PID_θ, PID_ψ. For each PID-controller the error is the difference between the desired angular velocity and the actual value.

2.6.5 Cascaded PID Control

Position and velocity control suffer from the same issue, sensors or techniques available to provide the required feedback have a relative slow update rate. Typical sensor or techniques are e.g GPS, Visual SLAM and Visual Flow. This makes these feedback system the bottle neck of the control-loop, since the motor can react much faster. Meaning, if the position or velocity controller would sit at the lowest level of the control hierarchy the control is only able to react to disturbances every time a new feedback comes in, but the faster a error can be detected the better, especially in a highly dynamic system like a multi-rotor platform. A typical solution for such a problem is the cascade control approach, an example can be seen in Figure 2.17. In this approach there are multiple levels of control, the lowest level are mostly controllers with a fast reaction time provided by their sensors. In the case of an quadrotor MAV this would be the angular velocity control, Internal Measurement Units (IMU) required for the feedback can have a usual update rate between 200 to 1000 Hz and can therefore detect rapid changes.

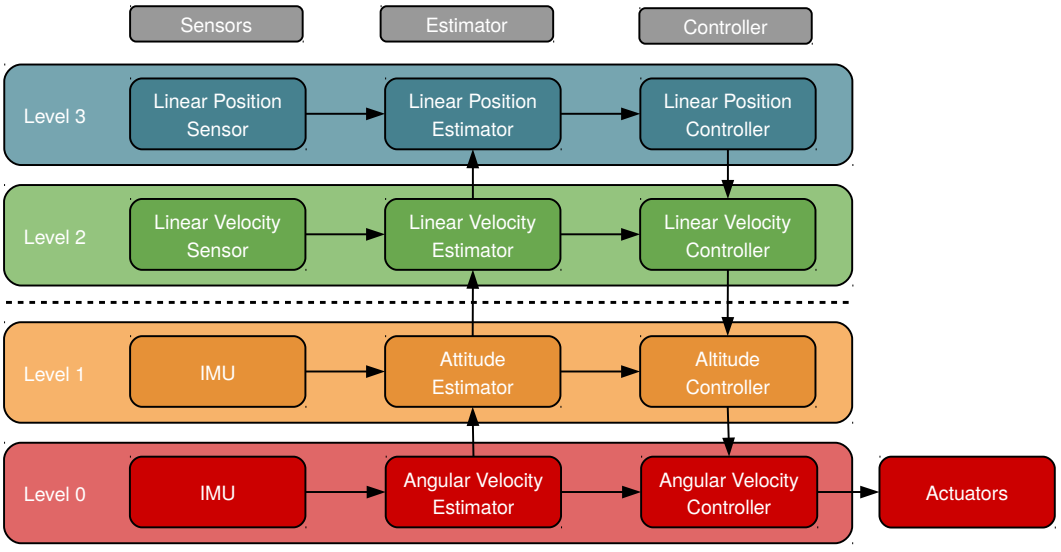


FIGURE 2.17: The general high-level system architecture for an autonomous MAV.

Controllers with lower refresh rate like position control sit on top of the angular velocity controller and pass down set points, these are than maintained by the low-level controller, until a new set point is handed down. This allows the system to be much more responsive to external disturbances. The higher level controller can consider the lower level controller as a quasi-static system i.e. one that responds almost instantly and holds the desired commands. This drastically simplifies the complexity of the controllers at a higher level while still accounting for fast dynamics and the handling of disturbances at

a lower level. A drawback of this kind of approach is that it requires tuning for the PID-gains at all levels. In respect to the position- and velocity-control this introduces four new PID-controllers for each of those control-problems, controlling the position/velocity on the x-axis, y-axis, z-axis and the yaw-rotation. But the high level controller don't require intensive tuning if the low-levels are performing as expected, most of the time a general PI- or even P-controller is sufficient. Cascaded PID-controller are therefore tuned from the bottom-up.

2.7 Conclusion

In this chapter we introduced the background relevant to this thesis. We introduced the behaviours occurring in social insects swarms, on which the related work (Chapter 3) and our proposed approaches are basing on (Chapter 4, Chapter 5 and Chapter 6).

In order to build the required knowledge foundation for the proposed technique introduced in Chapter 7, we introduced machine learning techniques of neural networks, reinforcement learning, in particular deep reinforcement learning and the filtering technique of the monte carlo localisation. Finally, we showed techniques necessary for the coordination on our real-world platform, in form of quadrotor control and radio-based distance estimation.

Chapter 3

Related Work

This chapter shows the related work relevant to this thesis. The work proposed in Chapter 4 and Chapter 5, can generally be categorised as bio-inspired coverage approaches. Area coverage is usually divided in different types of coverage, Section 3.1 provides a general overview over types and common approaches. Since we are proposing ant- and bee-inspired approaches to solve the problem of coverage, we highlight similar solutions used in optimization and robotics in Section 3.2 and 3.3. Section 3.4 goes in further detail into the coverage approach called StiCo, since this approach serves as foundation and comparison for techniques introduced later in the thesis. In Chapter 7 we propose a framework to provide bio-inspired coordination on pocket drones, since this also includes collision avoidance, we provide a general overview over the work in this area in Section 3.5. The chapter is concluded in Section 3.6.

3.1 Coordination and Coverage Techniques

The concept of coverage is a metric for evaluating robotic systems and was first introduced by Gage [47]. Gage defines three basis types of coverage: Blanket coverage, where the objective is to achieve node formation which maximizes the total detection area; barrier coverage, which aims to minimize the probability of undetected intrusion through the barrier; and sweep- or repetitive-coverage with the goal to cover all accessible interest points in an given environment over time, while maximizing the rate of visits over all points and minimizing the total distance travelled by all robots. Blanket coverage is most common for the deployment of mobile sensor networks in an unknown environment, the sensor nodes are initially placed in a compact configuration, where the nodes are trying to spread out such that the area covered by the network is maximized. One example scenario for a specific use case is a hazardous material leak in a damaged structure. Mobile sensor nodes equipped with chemical sensors spread out from a initial position to gather information about location and concentration of the hazard. Due to the fact that the communication infrastructure could be damaged the nodes have to insure their own network structure, even if single nodes get lost or destroyed. Another popular approach in this field is the potential field technique, first introduced by

Khatib [63]. In his approach nodes are treated as virtual particles, covered by a virtual force-field, which repels nodes from each other, the environment and obstacles to ensure that the nodes will quickly maximize the coverage area. In addition to these forces, nodes are subject to a virtual friction force. This force ensure that the network will eventually reach a state of static equilibrium. This techniques has also been applied for group formation problems for mobile robots [8, 94], which is to some extent similar to the deployment problem of mobile sensor networks. In robotics, repetitive-coverage can be described as a problem where a team of robots has to visit multiple Points Of Interests (POI) in a known environment frequently, to perform certain tasks [35, 76]. The goal of such a algorithm is to keep the average visit frequency over all POIs high, while achieving a minimal total travelled distance and a balanced workload over all robots. Typical real world use cases for such a problem are patrolling, lawn mowing and chemical spill clean up. Many approaches concerning multi-robot patrol partition the area into sub-areas divided between the robots. Inside such a sub-area, each robot applies a single robot patrol algorithm. Ahmadi and Stone [1] describe a negotiation-based approach for distributing the area between the robots and dealing with events such as addition or removal of robots to the environment. Jung and Sukhatme [57] introduce a region based approach for tracking tracking targets in a system with mobile robots and stationary sensors. Guo et. al [51] also divides the area between the agents while focusing on their localization and sensor capabilities. Another important form of multi-robot coverage is *Terrain coverage* or multi-robot exploration. It can be defined as a problem where a robot tries to visit each and every location in a continues bounded unknown environment by avoiding obstacles and perform defined tasks [26, 46, 78]. A Terrain coverage algorithm must generate a coverage path, which is a chain of motion steps for a robot, the optimal coverage path takes minimal time and guarantee to cover the entire terrain and perform the task efficiently. Most approaches divide the in the environment into grid cells and explored one cell at the time until the whole area is covered. One of the first was Spanning Tree Coverage (STC) which solves single robot coverage optimistically [45]. It is proved that STC is a polynomial time coverage that divides the terrain into cells, creates a spanning tree and the robot circumnavigates around it. The same idea was applied by Hazon and Kaminka on a multi-robot system [53]. Several authors proposed marked based approaches in multi-robot exploration, in which robots are making bids on a sub task of a exploration attempted [93, 98, 110]. These bids are based on values such as expected information gain and travelled cost to a particular location. This seems to minimize the costs while maximizing the benefit. These approaches rely heavy on a reliable communication link and the bidding-strategy of each agent.

3.2 Ant Inspired Techniques

Ant-based multi-robot coverage is highly inspired by the notion of stigmergic communication introduced by Dorigo [28, 30–32] in 1992. The basic idea underlying this form of communication is that pheromones are used as a medium for transmitting messages among artificial ants. During the past few years, variants of Dorigo's method, known as Ant Colony Optimization (ACO), have been developed, and it has been shown that it allows for very efficient distributed control and optimization in a variety of problem domains e.g. scheduling problems, routing problems, assignment problems et. al. (Dorigo et al. [29]). Wagner et al. [105] were the first, who invested stigmergic multi-robot coordination for covering/patrolling the environment. In their approach, the robots were supposed to be able to (1) deposit chemical odor traces and (2) evaluate the strength of smell at every point they reach. Based on these assumptions, they used robots to model an unmapped environment as a graph, and they proposed basic graph search algorithms (such as Depth-First-Search and Breadth-First-Search) for solving robotic coverage problems. Many other researchers used this graph-based modeling scheme in order to design solutions for multi-robot patrolling/covering problems, e.g., Elor and Bruckstein [36], Glad et al. [48], and Yanovski et al. [109]. In contrast to the mentioned graph-based techniques, a geometrical framework can also be used for addressing the swarm robotic coverage problem. One of the most important geometric techniques is Voronoi-based coverage that has been introduced for solving robot coverage problems (e.g., Cortes et al. [27], Schwager et al. [96]). These Voronoi-based techniques aim at devising coverage algorithms, which work according to the following basic rule: Each vehicle moves toward the center of its Voronoi region. Based on this rule, many researchers have proposed modified covering approaches, which are adaptable to changes in the environment and are provably convergent (e.g., Schwager et al. [95]; Breitenmoser et al. [14]). However, all these geometrical algorithms require a group of robots with the capability of direct communication and in most of the cases also need very complex mathematical computations (e.g., calculating margins and center of mass for an individual Voronoi region), which limits their potential real-world usage. Another related research topic is focused on the "real" implementation of stigmergic communication in real-world environments. For example, chemical substances such as ethanol (C2H5OH) are used instead of natural pheromones by Fujisawa et al. [44].

Works like [75] by Dorigo et al. use the idea of robotic chain from Goss et al. [49]. In this approach the robots itself are used as the pheromone trail substitute. An agent can have two different roles: chain member or explorer, the role determines a led light color on the robot. In the beginning of the approach, the robots start exploring the area around a homing beacon, if a certain distance threshold to the beacon or certain time is passed a robot switches to the chain member role. When a robot is in the chain member role it holds its position and other explorer agents can use this chain member as an additional beacon. The distance threshold has to be smaller than the sensor range

of the robot, this allows the robot traverse the chain from start to finish. This principle allows ant like exploring and foraging, but is limited by the size of the swarm.

StiCo (Stigmergic Coverage) introduced by Bijan Ranjabar et al. uses fluorescent or "glow in the dark"-foil to simulate pheromones. In this solution each robot is equipped with a led-light and when the robots move across the foil, they leave a glowing trail. Duo to the properties of the foil the glow intensity fades over time, which symbolises the decay of the ant pheromone. In StiCo the robots communicate indirect through the marking of trails and by using a simple circular movement strategy it achieves a scalable, robust and efficient coverage behaviour. This technique can achieve a voronoi-like coverage, since this technique is part of the approach proposed in Chapter 5, StiCo is explained in greater detail in Section 3.4.

These mentioned systems are categorised as swarm approaches since they are highly distributed and every agent makes autonomous decisions based on their local information, gathered from local sensing and communication. The general idea of swarm intelligence is that the objective is not coded into each agent, rather the desired behaviour emerges consequently from the interaction between the agents and their interaction with the environment. This makes their development and evaluation process quite difficult. Therefore, works like Winfield et al. [108] or Kazadi [61] introduce and examine the concept of swarm engineering and conduct case studies in this field to apply the traditional analyses, design and test phases to swarm development, in order to work towards dependable robotic swarms. Most swarm approaches are simple and homogeneous behaviours, executed on each agent, which makes them highly redundant and therefore robust. Robustness is usually defined as the ability to provide the demanded functionality when single agents of the system fail. Since the failures in these system are mostly base on hardware malfunctions, Winfield et al. apply in [107] a Failure Mode and Effect Analysis (FMEA), where they define and identify hazards (points of failure) and propose methods to identify the optimal swarm size to provide the required reliability. These approaches require always a estimated probability for the points of failures and an estimate of the minimal number of agents required to accomplish the task, which is sometime difficult to determine.

3.3 Bee Inspired Techniques

Bee Colony Optimization (BCO) was introduced independently by Lemmens et al. (Lemmens et al. [70]; Lemmens and Tuyls [71]) and by Karaboga et al. (Karaboga [59]; Karaboga and Basturk [58]). Unlike ACO (which is only inspired from the notion of stigmergic communication), scientists are inspired by various behaviors of bees: foraging behavior in Lemmens et al. (Lemmens et al. [69]), Beehive protocol (Wedde et al. [106]), BeeSensor (Saleem and Farooq [91]), bees mating procedure (Senthilkumar and Chandrasekaran [97]; Sahoo et al. [89]), and pheromone signaling mechanism (Caliskanelli et al.[22]). Karaboga et al. [58, 59] introduced the optimisation algorithm Artificial

Bee Colony (ABC) in which bees represent the search agents and their environment represents the potential solutions. In their work, the high-quality candidate solutions represent a pollen source which encourages further exploration of the region by additional bee agents. In the networking context, protocols have been developed in which network packets are treated as biologically inspired agents [2, 60]. In the Beehive protocol (Wedde et al. [106]), packets search for efficient routes through an IP network in a process modelled after the foraging behaviour of bees. Similar work targeted specifically at Wireless Sensor Nodes (WSN) is BeeSensor (Saleem and Farooq [91]) in which routing is performed via classes of packets following different types of bee behavior: for example as scouts and foragers. The redundancy introduced by BeeSensor is capable of increasing the proportion of delivered packets compared to common methods like Ad-hoc On-demand Distance Vector-Routingalgorithmus (AODV) [77], although it experiences increased latency due to the possibility for bee packets to select suboptimal routes during exploration. A general framework through which a set of biological agents can attempt to simultaneously satisfy multiple possibly conflicting objectives (such as latency, energy efficiency, and delivery success in a WSN) is provided in MONSOON (Boonma and Suzuki [13]). Their previous work has also mapped the bee colony model more directly to WSN hardware, with individual nodes representing individual bees, status within the hive corresponding to node responsibilities, and signalling chemicals corresponding to data packets. Recent work has applied bee protocols specifically to WSN load balancing (Senthilkumar and Chandrasekaran [97]), which is inspired by the bees mating procedure. This approach focuses on cluster set-up communication overheads by restricting the communications with bee mating election algorithm. Removing the redundant communications inside the cluster increases the successful delivery ratio while decreasing the latency. Caliskanelli et al. explore the pheromone signaling mechanism in honeybee colonies in Caliskanelli et al. ([18][22]) to solve the load balancing (i.e., to distribute the network load among processing elements) and redundancy control issues in largescale WSNs. Caliskanelli and Indrusiak improved their parameter-rich technique in Caliskanelli and Indrusiak ([22]) by tuning its parameters and modifying their initial work (Caliskanelli et al.[19]). Later on, they applied pheromone signaling process on WSRNs (Wireless Sensor Robot Networks) in [20]. Another biological behaviour adopted for exploration and mult-robot coordination is the recruiting and navigation strategy used in a bee colonies. The former is used to distribute knowledge to other member of the colony. More specifically by direct messaging (dancing), agents are able to communicate distance and direction towards a target [104]. The latter is applied to efficiently navigate through an unknown environment. Bees use a strategy called Path Integration (PI). They compute their present location from their past trajectory continuously and, as a consequence , can return on a direct path back to the starting position (Lamrinos el al. [68]). A popular bee inspired optimisation algorithm is the Artifical Bee Colony (ABC) introduced by Karaboga [3]. He defines three types of information which is exchanged during the direct communication (bee dance): The direction of the

target position, the distance and the quality or pay-off. Also he divides the bee colonies in two types of robot/bees:

- Scout bees: Carries out a random search for target positions and return to the hive to inform the other bees about the gathered information.
- Onlooker bees or unemployed bees: Interpret the information of the dance and decide the best target position to go for next.

ABC was later used for an algorithm for path planning of a robot by for example by Savasani and Jhala [79] or Saffari [87]. These algorithm are proposed to minimizing the travelled time and distance.

3.4 StiCo

This section describes StiCo (Stigmergic Coverage), an ant stigmergy (Section 2.1.1) inspired coverage approach. StiCo was introduced in [82], it's highly robust, adaptive, easy to implement and serves as foundation and comparison for the in this thesis proposed algorithm called HybaCo (Section 5).

3.4.1 StiCo Principle

The StiCo approach follows the principle of indirect, stigmergic coordination to establish a simple but efficient coverage of the environment. In contrast to the classical stigmergic coordination in Ant System (AS) [28], where 1) agents have a tendency to move straight with minor deviations, and 2) traces act as sources of attraction, in StiCo robots orbit in circles, instead of moving straight, and the traces have repulsive characteristics instead of attracting the agents. These two differences turn the path finding characteristic of AS into efficient area coverage of StiCo. The robots are equipped with two simple sensors (in the front-left and front-right directions like an ant antenna), capable of detecting immediate traces. Each robot orbits in a circle with a predetermined radius. Based on the circling direction (CW or CCW), one sensor would be considered as the interior sensor and the other as the exterior one. When the interior sensor detects pheromone, the robot changes its circling direction immediately as shown in Figures 3.1(a) - 3.1(c) and Algorithm 3. Otherwise, if exterior sensor detects pheromone, the robot continues rotating in the same direction until it doesn't detect pheromone anymore. The amount of pheromone deposited by each robot can practically be adjusted based on pheromone evaporation rates, in a way that robots do not collide with their own pheromones. For further information on StiCo principle see the work by [81]. The robustness, scalability and functional extensibility (see the work by [82]), make StiCo an interesting alternative to Voronoi-based and graph-based multi-robot coverage approaches which currently are dominant in the field. Moreover, because of these features StiCo has a broad application potential and can be used for various monitoring, rescue, and patrolling missions.

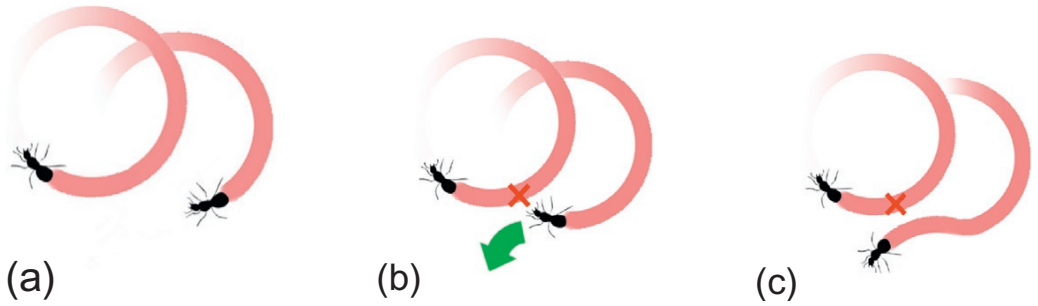


FIGURE 3.1: StiCo coordination principle: (a) robots circle around. (b) the right robot detects pheromone. (c) the right robot changes circling direction.

Algorithm 3: StiCo Algorithm

```
Require: Each robot can deposit/detect pheromone trails
  Initialize: Choose circling direction (CW/CCW)
  loop
    while (no pheromone is detected) do
      Circle around
      deposit pheromone
    end while
    if (interior sensor detects pheromone) then
      Reverse the circling direction
    else
      while (pheromone is detected) do
        Rotate
      end while
    end if
  end loop
```

3.5 Collision Avoidance

As mentioned before we propose a framework to provide bio-inspired coordination on pocket drones in Chapter 7, since this also includes collision avoidance, we provide a general overview over the work in this area. In the field of multi-robot systems, collision avoidance is usually defined as the task of avoiding the collision with static and dynamic object. Where the avoidance of static object can generally be solved with traditionally planning, is the avoidance of dynamic objects more complex. The native approach is to observe consecutive obstacle positions to extrapolate future trajectories. Dynamic object are usually defined as a velocity obstacle (VO [41]), which is a geometric representation showing all velocities, which will eventually result in a collision. Robots are pro-active and neglecting this fact can lead to inefficient trajectories or oscillation, Van den Berg et al. [103] introduces the concept of reciprocal velocity obstacle (RVO) to address reciprocity: it considers robots as pairs and splits their responsibility of the collision avoidance, between each other. Over the years different variations have been developed to address different problems of RVO, these approaches include e.g. HRVO [99] and CALU [54].

Current research to reduce the computational overhead aims to imitate the behaviour of the previous behaviours through deep-neural networks in a supervised learning setting (e.g. [73]), but this is mostly applied in simulations or full-observable environments, where the global positions of all robots is given.

3.6 Conclusion

This chapter shows related work relevant to this thesis. We provide an general overview over coordination, collision avoidance and coverage techniques and detail the strongly related ant- and bee-inspired approaches. We goes in further detail into the coverage approach called StiCo, since this approach serves as foundation and comparison for techniques introduced later in the thesis.

Chapter 4

Bee Pheromone Based Coverage

This Chapter introduces a novel multi-robot coverage approach inspired by the pheromone signalling behaviour of honeybees, described in the preliminaries Section 2.1.3. The aim of this technique, named BeePCo (Bee Pheromone Coverage), is to provide a simple, scalable algorithm for finding a static arrangement of robots that maximises the total detected area, also known as blanket coverage. Each robot in this team acts individually on its local information to improve the performance of the whole collective. This can e.g. be applied in the context of mobile Wireless Sensor Networks (WSN). Additionally, it serves as a foundation for a surveillance and exploration approach illustrated in Chapter 5. The remainder of this chapter is structured as follows. Section 4.1 introduces the biological principle on which the proposed coverage approach is inspired. Additionally, it summarises a pheromone based WSN load balancing approach, we can utilise, if necessary. Section 4.2 covers the proposed pheromone signalling based coverage algorithm for Multi-Robot Systems (MRSs). Section 4.3 describes the employed simulator. It continues with the experimental setup and results in Section 4.4 and finishes with the conclusion in Section 4.5.

4.1 Introduction

In previous work on Pheromone Signalling (PS) [18, 21, 22], bee pheromone signalling is employed for service load-balancing in WSNs. The algorithms aim is to maximise the area, a certain network service is provided by the wireless sensor nodes, while minimising the number of active nodes, in order to save energy. Since the proposed multi-robot coverage can be applied in context of WSNs and its signalling procedure is quite similar to PS's, we integrate PS's principle into BeePCo to allow service load balancing, if required. The load-balancing in already evaluated in [18, 22] and this chapter is therefore mainly focusing on the multi-robot coverage.

The bee-inspired coverage algorithm, *BeePCo*, described in this section is a completely decentralised approach that has low computational overhead and employs direct local communication. Changes in pheromone levels are used by many social animals to orchestrate the colony by assigning responsibilities to each individual. Roberts [85]

explains the process of larvae differentiation in beehives as an example of such orchestration. Bees have developed a special hormonal system to ensure every beehive has a queen, which maintains the stability of the colony and orchestrates the behaviour of all other bees. Throughout its life, a queen bee stimulates a pheromone called Queen Mandibular Pheromone (QMP), which makes the worker bees aware of its presence as queen. This hormonal mechanism works as follows: the worker bees lick the queen bee and passes the pheromone, to the others. If there is no pheromone passed through the worker bees, they will consider the queen to be dead. Consequentially, workers will select a larva to be fed with large amounts of the royalactin protein. Royalactin protein induces the differentiation of honeybee larvae into a queen. If worker bees keep receiving the pheromone, they are aware of the queen bee's presence and will take no action towards raising a new queen. This behaviour is described in greater detail in Section 2.1.3.

The proposed coverage technique is inspired by the pheromone propagation, and PS's load-balancing takes inspiration from the differentiation between queen and worker bees. In the context of load-balancing, the role of a queen bee denotes a robot that is responsible for managing the execution of all required services and a worker-robot idles to save energy.

Throughout this thesis we will refer these robots as Queen Robots (QR) and their responsibility (service) is to sense their surrounding area, e.g. for exploration or intrusion detection purposes. The basic strategy of the algorithm is based on the periodic transmission of pheromone by QRs, and its retransmission, by recipients, to their neighbours. The pheromone level of each robot decays over time and distance to the source. All robots accumulate the pheromone received from other QRs and if at a particular time the pheromone level of a robot is below a given threshold this robot will differentiate itself into a QR and execute the service.

4.2 Pheromone Signalling Based Coverage Technique

This section focuses on the coverage performance of BeePCo and disregards the load balancing for now. All experiments are therefore executed with an "infinitely" high pheromone threshold to ensure the robot's activity. Although the coverage is not particularly benefiting from service load-balancing, it is still described for the sake of completeness and to provide a base for future applications.

The pheromone propagation via licking in a bee-hive, is in the proposed BeePCo algorithm substituted by local messages between neighbouring robots. Each pheromone message includes the robot-identifier i and the position P, where the it is coming from. The pheromone intensity is derived from the relative distance between sender and receiver. By accumulating all received pheromones, a robot is able to calculate the pheromone levels in the environment. The passed position information can either be global or in the local frame of the receiver.

The level of pheromone indicates how well a certain area is covered. Areas in the environment having a lower level of pheromone, at a given time, demonstrate a lower robot density as opposed to other parts. This means that areas with low pheromone level have either low coverage, or are not covered at all. The agents in BeePCo are therefore repelled by pheromone in order to move towards areas with a low density.

The proposed algorithm consists of the five following parts, which are executed on every robot of the MRS: the differentiation cycle, movement process, wall detection and the decay of pheromones are time-triggered. The remaining pheromone propagation occurs in an event-triggered process.

The first time-triggered part, referred to as the differentiation cycle (Algorithm 4), is part of the load-balancing and is executed by each robot of the MRS, every T_{QR} time steps. On execution, the robot checks its current pheromone level (H_{sum}) against a predefined level ($threshold_{QR}$), the robot will act as a Queen Robot (QR) and provide the required service, if the level is lower than said threshold. H_{sum} is the sum of all individual pheromone clouds H_i , with i being a unique id of a specific robot. H_{sum} can be formalised as:

$$H_{sum} = \sum_{i}^{n} H_i$$

The summed up pheromone is only used for load balancing and its level is decayed over time. QRs transmit pheromone, to its network neighbourhood to make its presence felt. Each pheromone message of a robot s is represented as a three-dimensional vector $pm_s = \{M_h,\ P_s(t),\ id\}$. The first element M_h denotes the distance to the source in hops. Hops indicate the number of transmissions of a message so far. This means that a pheromone message has a hop count of zero at its source. The pheromone dosage of a message is inversely proportional to the euclidean distance between the senders position $P_s(t)$ and the receiver position $P_r(t)$, at time t. This means the closer the sender (s), the more intense the calculated pheromone dosage H_s, which is defined as follows:

$$H_s = \begin{cases} \dfrac{H_{range} - \|P_r(t) - P_s(t)\|}{H_{range}}, & \text{if } \|P_r(t) - P_s(t)\| < H_{range} \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

Where H_{range} is defined as the maximal range a pheromone can travel and $P_r(t)$ as the position of the receiver (r). By using this equation H_s will always be $H_s \in [0, 1]$. The last element (id) of the message is an unique identifier of the robot.

The event-triggered part of BeePCo deals with the propagation and information extraction of the pheromone messages. The purpose of propagation is to extend the influence of a QR, beyond its transmission range. Propagation is not a periodic activity, it is triggered every time a robot receives a pheromone dose. The pseudocode is given in Algorithm 5. Messages with a higher hop count than $threshold_{hopcount}$ are not propagated, this reduces the pheromone influence and prevents an infinite loop of message propagation. In the case that the hop count of a message, of sender s, is below

Algorithm 4: Differentiation Cycle for Robot i

```
every T_QR do
  if (H_sum < threshold_QR) then
    QR =True
    broadcast pm_i = {0, P_i, i}
  else
    QR = False
  end if
```

the defined threshold ($threshold_{hopcount}$), the robot will calculate the pheromone level according to Equation 4.2, add it to total pheromone level H_{sum} and update the senders pheromone dosage H_s, according to Equation 4.2.

Algorithm 5: Pheromone Propagation Trigger Procedure of Robot i

```
pm is received
if (pm.M_h < threshold_hopcount) then
  s = pm.id
  update H_s according to Equation 4.2 using sender position P
  increase pm's hop count M_h by one
  save pm's position P as P_s
  broadcast pm
else
  drop pm
end if
```

The virtual pheromone represents the internal estimation of the covered areas in the environment. If an agent loses the direct/indirect connection to the robot covering an area, this estimation becomes more uncertain over time. In order to allow the sensing these areas again, the pheromone is decayed over time, similarly to pheromone seen in ant- and bee-colonies. Pheromone occurring nature, is gas which evaporates in an exponential fashion. Its evaporation rate is usually the pheromones half-life period ht, where ht is the duration it takes to evaporate half of the current dosage. Since a dosage is always reduced to its current half, in a constant time-period, the pheromone decay is exponential and can be formalised as follows:

$$H_i' = H_i \cdot 0.5^{\frac{\Delta t}{ht}} \tag{4.2}$$

With Δt being the time past since the last update. The pseudocode in Algorithm 6 shows the decay-cycle for agent i in a swarm of n robots.

Algorithm 6: Decay Cycle

```
1: Every T_QR Timesteps
2:   c = 0
3:   while c < n do
4:     H_c = H_c · 0.5^(Δt/ht)
5:     c = c + 1
6:   end while
```

The pheromones of each robot i are presented by H_i, BeePCo's purpose is to avoid the pheromone in order to move towards uncovered areas. This is done in periodic movement cycles shown in Algorithm 7. Since the pheromone decay is exponential, the level will never decrease to exactly zero, BeePCo has therefore a small movement threshold ($Movement_{threshold}$), if any H_i exceeds this threshold the robot will decide on a new movement action, otherwise it will stay at its current location.

Algorithm 7: Move Cycle

```
1: Every T_QR do
2:   if (pheromone present) then
3:     pheromone-guided moving decision
4:   else
5:     stop movement
6:   end if
```

The movement decision itself is done by accumulating direction pointing from the pheromone towards the robot. These two dimensional vectors are normalised and weighted by their current pheromone dosage, to intensify avoidance of closer clouds. The accumulated vector is than multiplied by the robots maximal velocity V_{max}, resulting in the final control command V_{xy}, representing the desired velocity on the x- and y-axis. As discussed before, the pheromone can't have a higher dosage than one, but if the sum of all vectors is higher than one, BeePCo will normalise the sum, to limit V_{xy} to the maximal velocity. Additionally, the algorithm only considers pheromones with a higher dosage than $Threshold_{movement}$. Algorithm 8 shows the pseudocode of this procedure.

If the virtual pheromones evaporates and if the robots doesn't receive new messages it surveils the current position until new connections are established. This happens when a robot finds an area, which is not in the sensor range of any other robot. Additionally BeePCo has a last, time triggered routine, to avoid collision with walls. This is also essential to maximise the sensor coverage. Robots without such a system would collide with the walls and therefore waste almost half of their sensor area, since the area is cut of by the wall. The mentioned routine handles walls like an other robot, this has the advantage that it can employ the same motion circle.

Algorithm 8: Moving Decision for Robot i

Every T_{QR} **Timesteps**
  initialise $V_{xy} = (0, 0)$
  initialise $s = 0$
  while $s < n$ **do**
    if $H_s > Threshold_{movement}$ **then**
      $V_{xy} += H_s \cdot \frac{P_i - P_s}{\|(P_i - P_s)\|}$
    end if
    $s = s + 1$
  end while
  if $\|V_{xy}\| > 1$ **then**
    $V_{xy} = \frac{V_{xy}}{\|V_{xy}\|}$
  end if
  $V_{xy} = V_{xy} \cdot V_{max}$

If a wall is detected by the sensors, the robot estimates the closest position to the wall P_{wall} and calculates the pheromone level H_{wall} to this position, according to Equation 4.2. Position and pheromone level are stored and the next time the motion cycle is triggered, it will avoid the "wall"-pheromone in the same way it avoids the virtual pheromone coming from the other robots. The whole flow of wall detection is represented in the pseudo code of Algorithm 9.

Algorithm 9: Wall-Detection Cycle

Every T_{QR} **do**
  if (wall is detected) **then**
    estimate closest position to wall P_{wall}
    calculate H_{wall} according to Equation 4.2.
    save P_{wall} and H_{wall}
  end if

4.3 Simulation

This section provide a general overview over the employed simulator structure in Section 4.3.2 and describes its ant-pheromone simulation in Section 4.3.2. The ant-pheromone simulation is a requirement for the coverage approach StiCo, which serves as a comparison for BeePCo.

4.3.1 Structure

To evaluate the performance of BeePCo, we designed a three-tier simulation. The first layer being the physical layer, responsible for simulating simple inertia, acceleration, collision detection with the environment and the pheromone for StiCo. For more detail about the pheromone simulation see Section 4.3.2. The second layer is split into

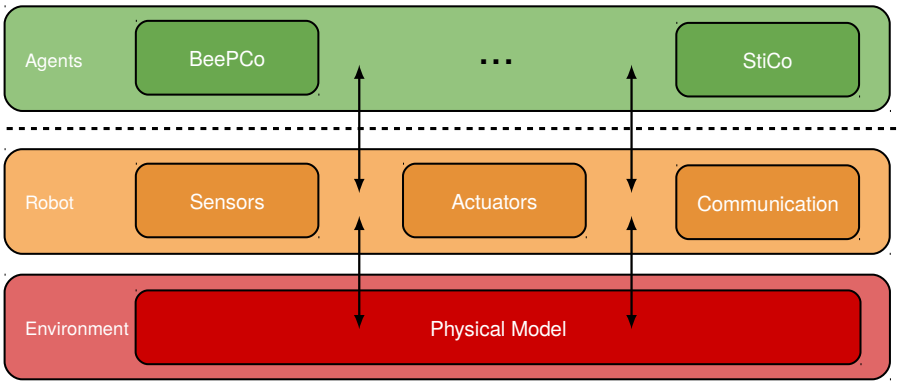


FIGURE 4.1: Simulator structure

communication-simulation and the robot-model. The former is providing the communication model, which allows communications for robots within each others communication range. The latter is the robot-model, enabling methods for reading sensor-data and use actuators of the robot, which are then accessed by the third and final agent layer. Agents using algorithms like BeePCo operate on this layer and employ the provided information to make movement and communication decisions.

4.3.2 Ant Pheromone Simulation

The ant-pheromone simulation proposed in [83], divides the environment into a grid, with each grid-cell representing the current pheromone level at this position, see Figure 4.2. While the robots are moving they expel pheromone, every T_s time-steps, increasing levels of cells at the back of the robot to the maximal level of P_{max}.

In order to simulate the decay of the pheromone, the simulation applies a linear reduction by the factor P_{decay}, this can be formulated as follows:

$$P'_{xy} = P_{xy} \cdot P_{decay} \cdot \Delta t$$

with P_{xy} being the pheromone level grid-cell x and y and Δt being the time past since the last update. All required parameters are summarised in the following table:

Parameter	Definition
P_{max}	Maximal limit of pheromone per cell
T_s	Update interval radiation
P_{decay}	Pheromone decay per second

TABLE 4.1: Ant-pheromone simulation parameters

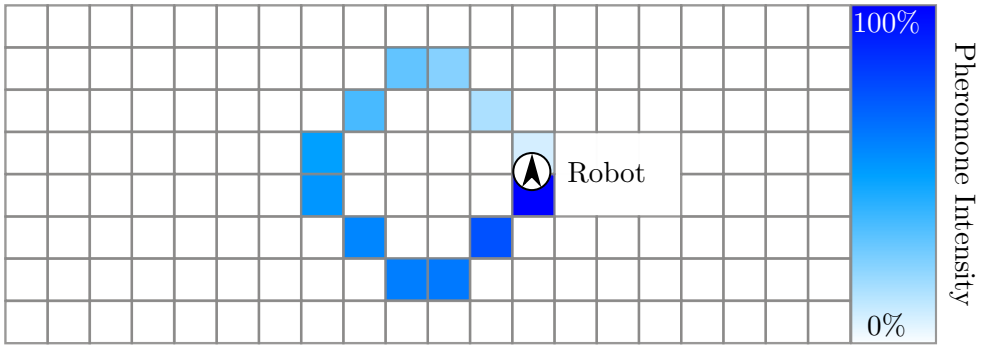


FIGURE 4.2: Simplified visualisation of the pheromone simulation. Each grid-cell represents the pheromone level at that position and the pheromone linear decayed over time.

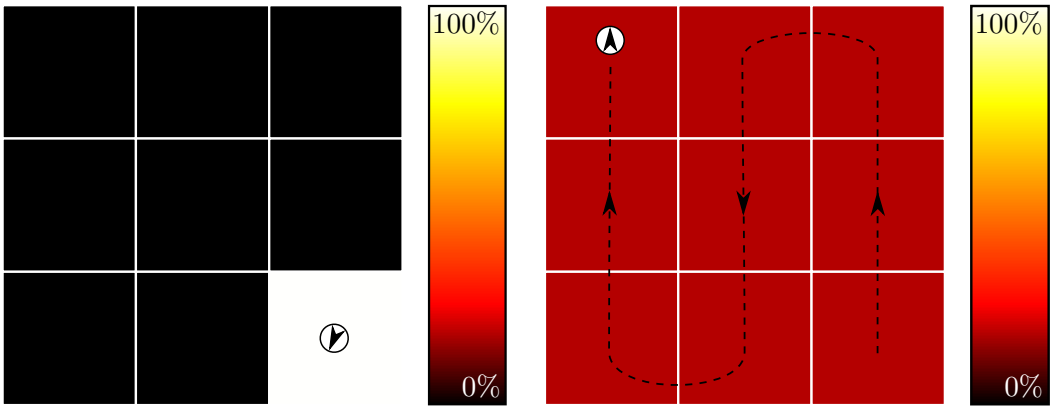
4.4 Evaluation Environment and Experimental Results

The following section describes the evaluation process and results. It starts by introducing the evaluation metrics in Section 4.4.1, continues with a description of the test environment and robot algorithm settings in Section 4.4.2 and concludes with the experimental results in Section 4.4.4.

4.4.1 Coverage Metrices

We are interested in the total sensor coverage, meaning the area which is detected by the sensors of all robots combined. Redundant sensed areas are only counted once, therefore the maximal sensor coverage is a configuration where there is no overlap. This can also be classified as 1-coverage, because each point has to be covered by just one sensor in order to extract the required information. This comes from the k-coverage terminology, where k is the number of sensor required to extract the information from a defined point. An example for a higher k-coverage problem would be triangulation for cell-phone localisation, where a phone has to be covered by three cell-phone towers in order to determine the position, this case is therefore defined as 3-coverage.

In order to calculate how evenly the environment is scanned by the robots, the environment is divided into a grid. In each time-step, the evaluation checks if a cell of the grid is covered by a sensor, if this is the case a counter related to the cell is increased by one. After a run, the evaluation is calculating the mean μ and standard deviation σ between these scan-counters. If the environment is scanned in perfect uniform fashion all cells have the same scan-count and the standard deviation is zero. In order to illustrated this, the thesis is employing a heat-map-visualisation. In this representation the total scan-count of each cell is divided by the number of time-steps, resulting in the percentage of the total time a certain cell is covered by a sensor. The produced image or heat-map, colours the cells in a gradient colour-scheme, with white symbolising 100% coverage and black 0% coverage. Figure 4.3a shows a simplified heat-map where a sensor covering one cell, is placed in a fixed location for the full duration of the experiment. Since the robot



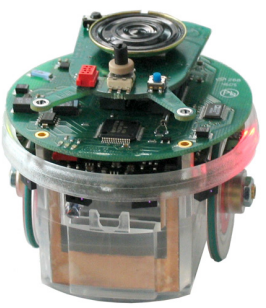
(A) (B)

FIGURE 4.3: Simplified heat-map visualisation: (A) shows a scenario where the robot is staying in one location for the whole duration of the experiment. (B) illustrates a scenario where the robot visited each cell the same amount of time.

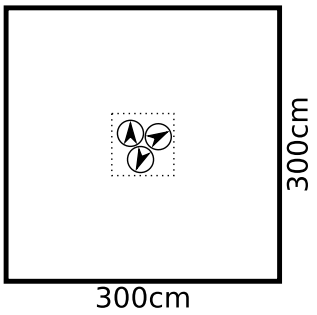
in this scenario is staying at one location, this location has 100% coverage over time and is therefore coloured in white, the remaining cells have a coverage of 0% resulting in a black colouring. Figure 4.3b illustrates the optimal case where each cell is visited the same amount of times, which results in an even colouring.

4.4.2 Environment

The test-area is of size 300 cm × 300 cm representing the robotics lab of the University. The area has a square shape and robots are initialised in the centre of the environment (Figure 4.4b). More complex areas are evaluated in Chapter 6. The simulated robots match the specification of E-Pucks, which are small low-cost robots with a diameter of 70 mm and a communication and sensing range of 25 cm (Figure 4.4a). Using these micro robots with a limited communication and sensing range has the advantage that the test environment can be relatively small in order to test the algorithm on a larger number of robots. The application can be easily scaled up to bigger robots with a higher communication and sensor-range, if the ratio between those properties (sensing and transmission-range) stays relatively the same. The sensor- and transmission range ratio in the scenario of the E-Pucks is quite interesting since it is 1:1, which makes a maximal sensor coverage more difficult. Because robots are not able to communicate with each other, if the distance between them is bigger than the sensor range. This means two robots can have a redundant sensor area without them knowing. For example, if the transmission range is double the length of the sensing range, robots would be able to communicate until sensor areas are perfectly tangential without overlap.



(A) E-Puck



(B) Environment

FIGURE 4.4: (A) shows an E-Pucks robot, modelled by simulation. (B) illustrates the squared test-environment with the start position in the center.

4.4.3 StiCo Parameters

StiCo is ant-pheromone coverage principle (Section 3.4), used as a comparison for BeePCo. In order to make StiCo perform optimally and compatible with BeePCo, its parameters are set as follows. StiCo is moving in circles with the radius r_s, to be able to scan the area inside this circle, the radius has to be smaller or equal the sensor range of the robot. Therefore we chose $r_s = 25\,\text{cm}$ to match the sensor range of the E-Puck.

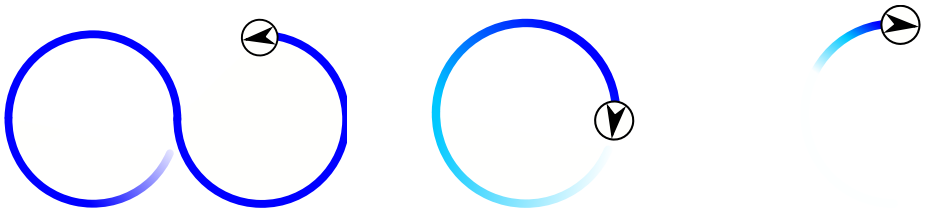
An other parameter is the decay rate (P_{decay}) of the pheromone, used in the simulation. As described before, StiCo moves in a circular fashion, marking it covered area with a pheromone trail. Optimally the pheromone evaporates in around the same time the robot finishes a complete a full revolution (Figure 4.5b). If the evaporation rate is too fast, the area is just partially or not marked at all (Figure 4.5c). Additionally, if the evaporation rate is too slow, uncovered areas could be surrounded by a trail long after the robot moved on (Figure 4.5a).

Since we know the maximal velocity of the robot ($V_{max} = 15\,\text{cm/s}$) and circumference of the circle a robot travels, we can calculate the duration required to complete a revolution as follows:

$$\underbrace{T_d}_{\text{duration}} = \underbrace{2r_s\pi}_{\text{circumference}} \; / \; \underbrace{V_{max}}_{\text{velocity}} \tag{4.3}$$

The simulation uses a linear model to represent the decay of the pheromone. Since we know that the pheromone radiated form the robot has the start value of P_{max}. We can generalise the required decay rate with linear equation:

$$P_s = -\frac{P_{max}}{T_d} \tag{4.4}$$



(A) (B) (C)

FIGURE 4.5: A: Shows a slow pheromone decay. B: Shows an optimal pheromone decay. C: Shows a fast pheromone decay.

By the setting the maximum pheromone level to one ($P_{max} = 1$), we derive a decay rate of -0.1 units/s. All the required parameters and their setting is summarised in Table 4.2.

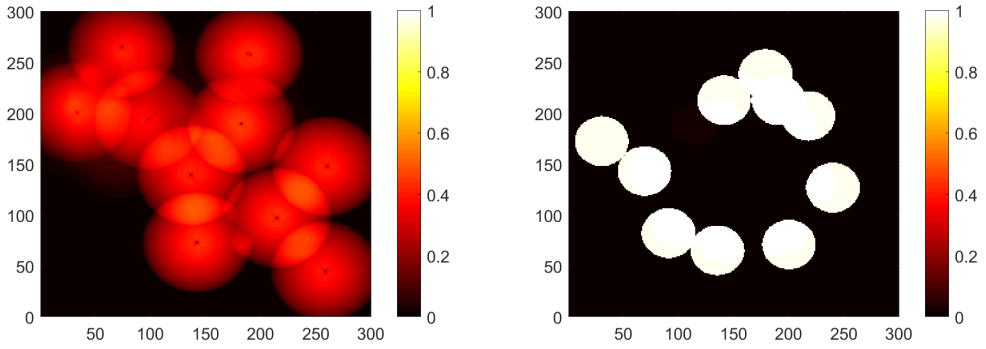
Parameter	Definition	Value
P_{max}	Maximal limit of pheromone per cell	1.0
r_s	Rotation radius	25 cm
P_{decay}	Pheromone decay rate	-0.1 units/s

TABLE 4.2: StiCo parameter settings

4.4.4 Experimental Results

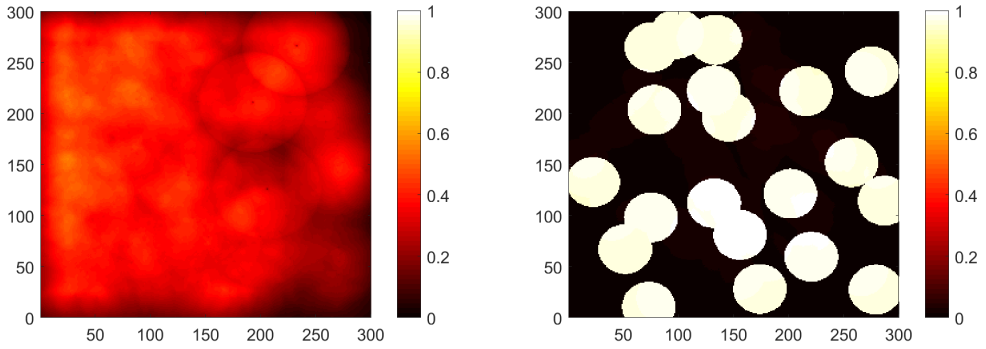
The main purpose of BeePCo is to provide blanket coverage, e.g. to find automatically configurations for WSN nodes. Blanket coverage is aiming to maximise the combined area which is sensed by all robot sensors combined, therefore it's trying to reduce the redundant areas. This Section evaluates the different performances and properties of BeePCo and StiCo in different scenarios, to see how it can be improved or combined. The results of this study is used in the HybaCo algorithm described in Chapter 5. We compare converging speed, maximal area covered and the distribution over time on different numbers of robots (10, 20, 30 and 40). We define the "distribution over time", as each cells total coverage time over the duration of a run, described in Section 4.4.1 and illustrated in Figure 4.3. BeePCo's performance is compared with the StiCo algorithm (Section 3.4). Additionally we compare both algorithms against the maximal possible sensor coverage with respect to the number of robots. This metric represents the optimal case where the robots' sensor range does not intersect with each other. This can also be referred to as potential coverage. We will refer the potential coverage as MaxCo. BeePCo and StiCo are configured as follows:

- ***BeePCo***: $Threshold_{hopcount} = 10$, $Threshold_{movement} = 0.001$, $Threshold_{QR} = 10,000$, $T_{QR} = 0.2$ sec and $ht = 0.4$ sec.
- ***StiCo***: $P_{decay} = -0.1$ units/s and $r_s = 25$ cm (See Section 4.4.3).



(A) StiCo (B) BeePCo

FIGURE 4.6: The distribution of robots in the arena using a MRS of 10 robots on StiCo and BeePCo algorithms.

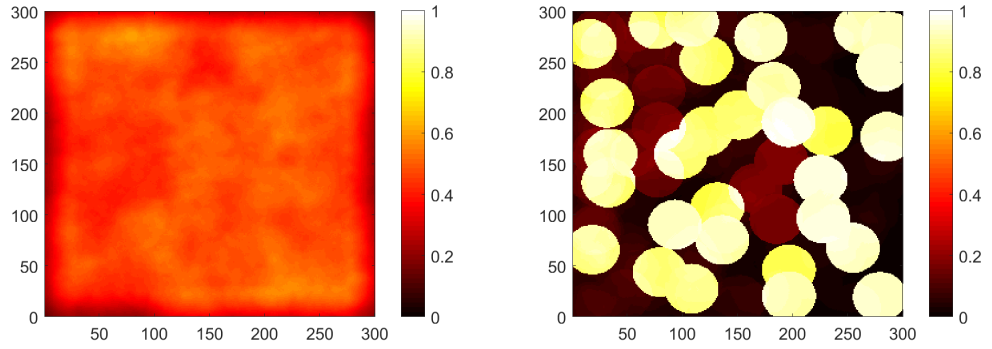


(A) StiCo (B) BeePCo

FIGURE 4.7: The distribution of robots in the arena using a MRS of 20 robots on StiCo and BeePCo algorithms.

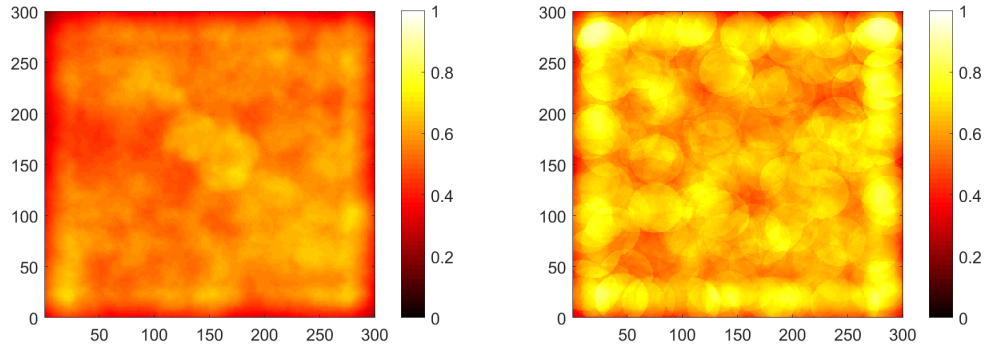
Figures 4.6, 4.7, 4.8 and 4.9 illustrate how evenly the area is covered over time, using 10, 20, 30 and 40 robots on a single run. The figure employs the colour scale described in Subsection 4.4.1, where a 100% covered cell is coloured white, a 0% covered cell is represented in black and the remaining percentages are shown in a gradient colour between those two extremes.

The more evenly the total area is coloured, the more uniform is the distribution of the robots positions over time. Besides showing the performance of StiCo and BeePCo in a single run, the heat-map visualises also how the number of robots influences the distribution or robot movement over time. In the case of StiCo, Figure 4.6 shows that the algorithm converges, in the case of 10 robots, to a steady state where the robots circle around the same position, shown as the red circles. This means the robots found a configuration where the rotation circles do not overlap and the robots are not influenced by each others pheromones. By increasing the numbers of robots in Figures 4.6a - 4.9a, the area is covered more and more evenly, and the average cell coverage also increases proportional to the amount of robots. This is due to the fact, that the increase in robots, increases the amount of pheromone trails in the environment. In order to avoid those



(A) StiCo (B) BeePCo

FIGURE 4.8: The distribution of robots in the arena using a MRS of 30 robots on StiCo and BeePCo techniques.



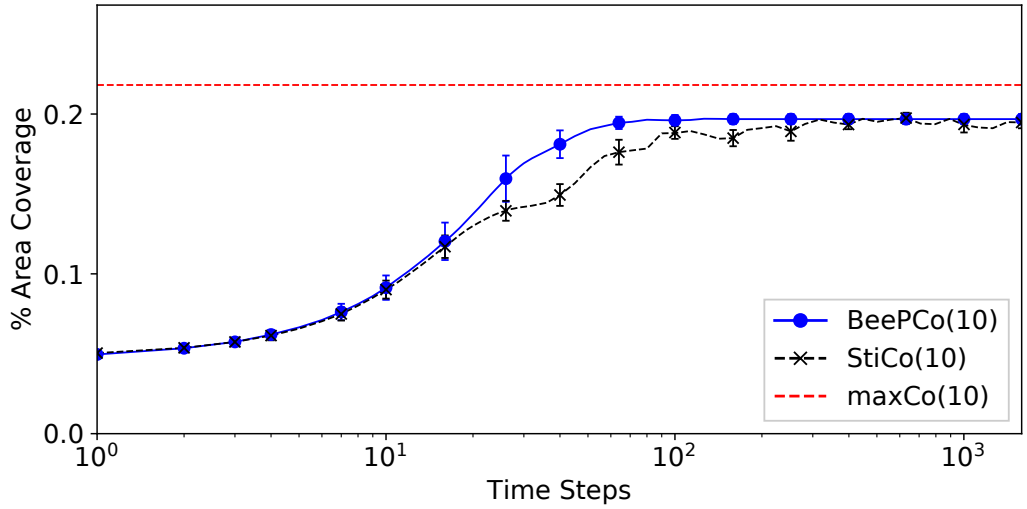
(A) StiCo (B) BeePCo

FIGURE 4.9: The distribution of robots in the arena using a MRS of 40 robots on StiCo and BeePCo techniques.

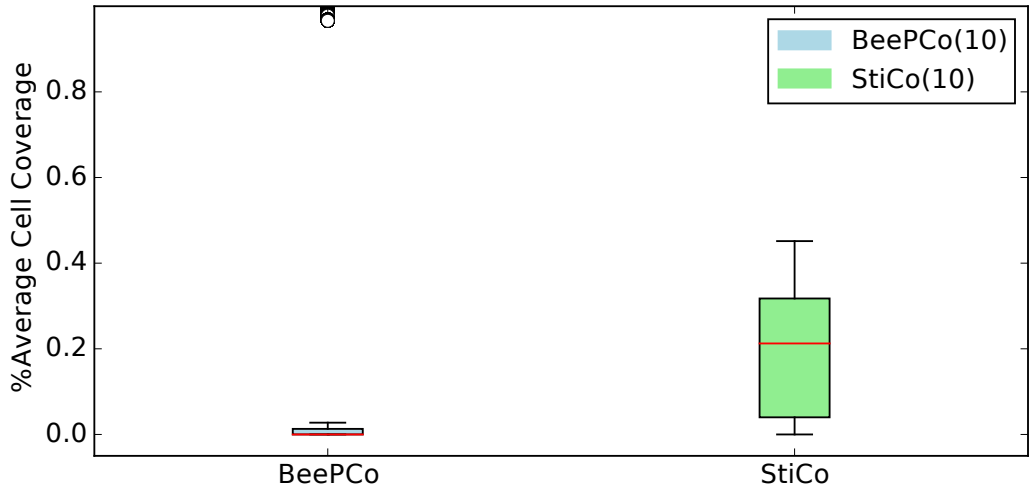
trails StiCo triggers its movement routines, described in Subsection 3.4, resulting in a more even spread through the environment.

The same effect can be seen in the BeePCo heat-map Figures 4.6b- 4.9b, but they also show that BeePCo is able to find a stable configuration for 10 and 20 robots. The corresponding Figures 4.6b and 4.7b illustrate that the algorithm converges to a configuration where the robots are not influenced by each other and stay in their positions, shown as white circles. This is due to the fact, that the robots have to move out of each others sensor range to maximise the covered area, resulting in a lost of connections since sensor- and transmission range are the same. Without communication links BeePCo will decay the virtual pheromone level until the movement routine is not triggered any more. The areas have mostly no overlap, which increases the maximal covered area, but the lack of movement doesn't allow an evenly scanned area like StiCo does.

Beginning with 30 robots (Figures 4.8b), BeePCo's agents are more and more influenced by each other and are starting to move, this can be seen by a little bit more variation in colour across the whole area. The last Figure 4.9b shows an extreme case



(A) The average percentage of the total area coverage, shown for each time step of the run (10 Robots).



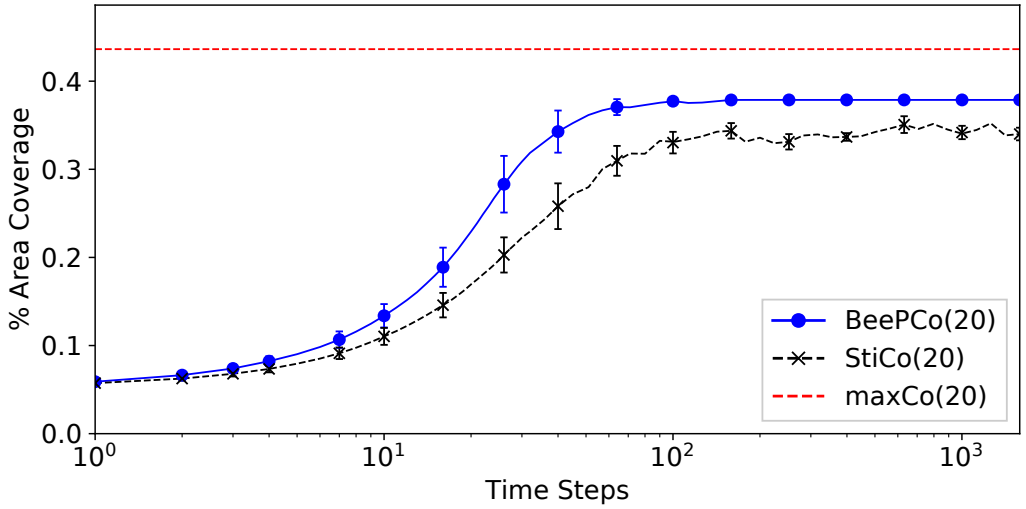
(B) Distribution over all grid-cells in the environment, showing how many percent of the total run time a cell is covered (10 Robots).

FIGURE 4.10

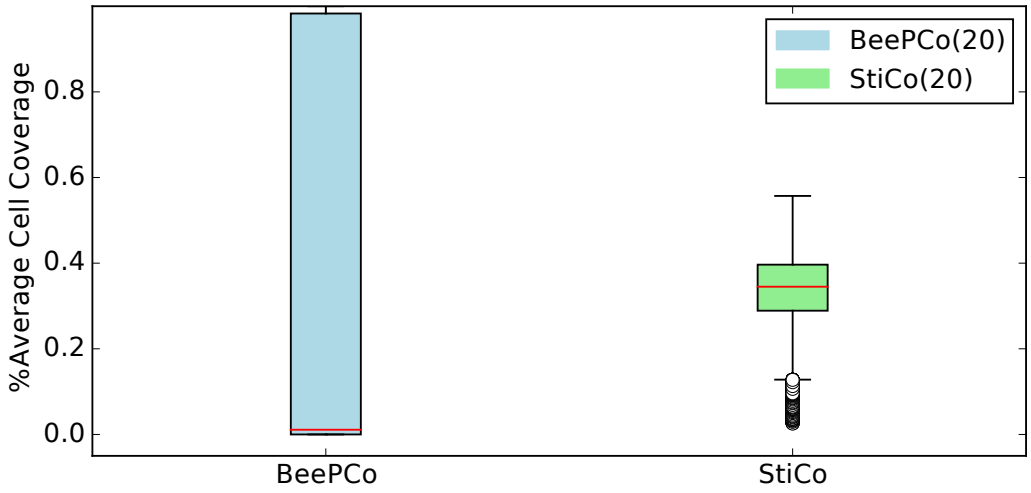
where the area size is too small for the number of robots, meaning it is not possible to distribute the robots in such a way, that there are no redundant sensor areas.

In the case of the 3m × 3m environment the maximal number of robots to allow no overlap is 35. Therefore a minimum of 5 robots will always have an overlap in the sensed areas. This results in a continues movement of BeePCo, in order to avoid the virtual pheromone. Figures 4.9b show a higher average cell coverage than its StiCo counterpart (Figures 4.9a), this is mostly due to BeePCo's direct and linear movement strategy against the circle path behaviour of StiCo.

Figures 4.10a - 4.13a show the progression of the covered area over the course of a run in respect to the number of robots, ranging from 10 to 40. The graphs show the mean and standard deviation at each time step over a total of 30 runs to ensure statistical



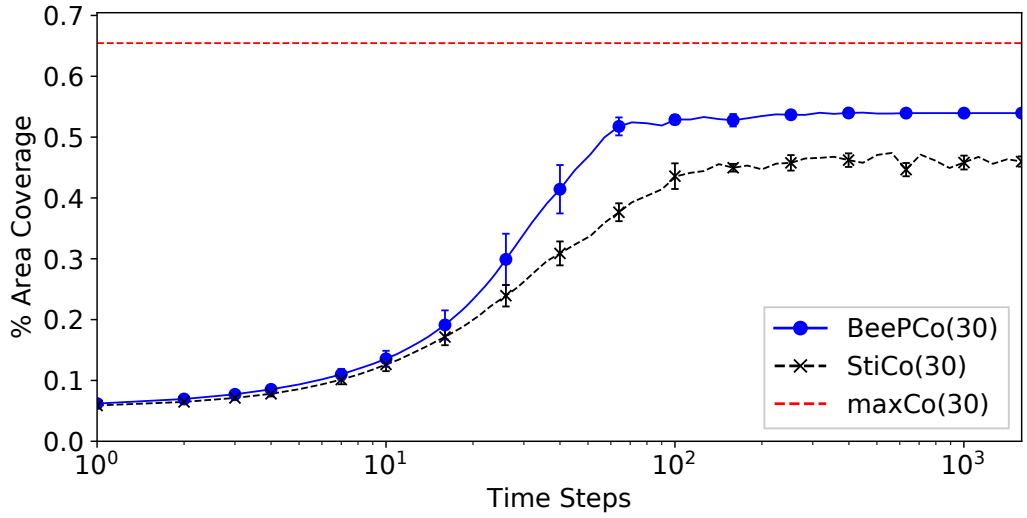
(A) The average percentage of the total area coverage, shown for each time step of the run (20 Robots).



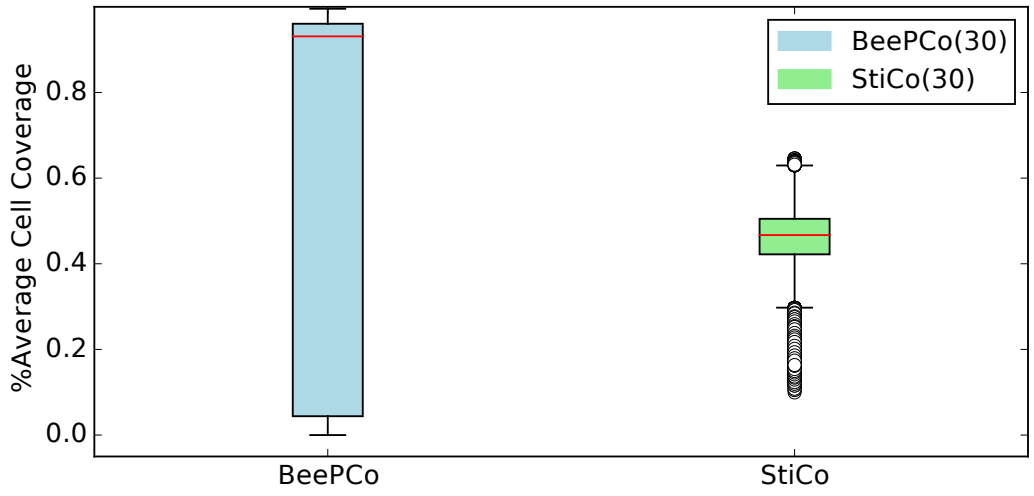
(B) Distribution over all grid-cells in the environment, showing how many percent of the total run time a cell is covered (20 Robots).

FIGURE 4.11

significance. It can be seen that the standard deviation (σ) over all runs, illustrated by the error bars, is negligible small for BeePCo and StiCo, which shows the reliability of both algorithms. In terms coverage speed, it can be seen that BeePCo converges faster towards its maximal coverage than StiCo, this effect is amplified by an increasing number of robots. Where the growth rate is similar in the setting of 10 robots (Figures 4.10a), the gap becomes significantly bigger with 20 to 40 robots. The reason for this is that BeePCo moves in a straight line versus the circular motion of StiCo, which allows an overall faster spread of the robots. As shown in the heat-maps, BeePCo converges in the setting of 10 to 30 robots, to a steady state, where the algorithm reaches its maximum coverage and the robots stop moving. This can be seen in Figures 4.10a- Figures 4.12a, since the area coverage stops changing, in contrast to the continues changes in StiCo.



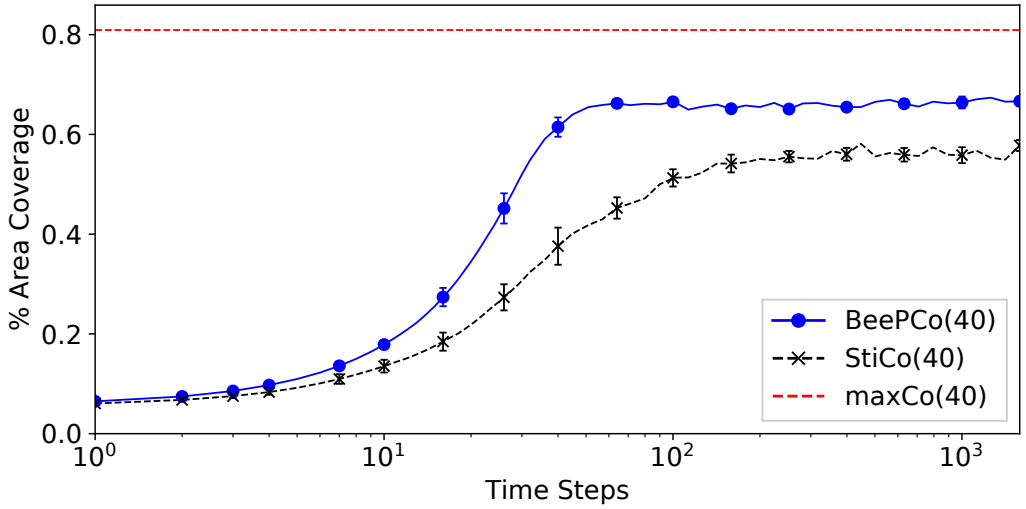
(A) The average percentage of the total area coverage, shown for each time step of the run (30 Robots).



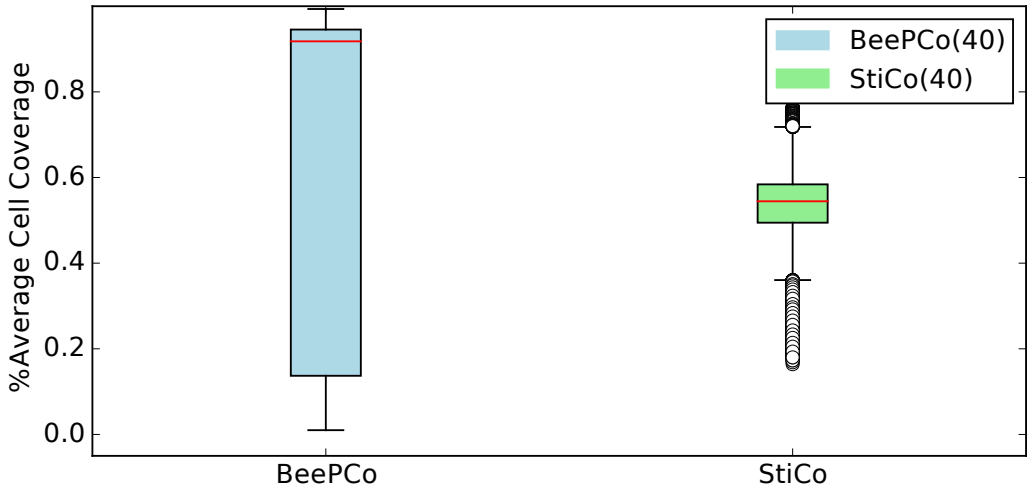
(B) Distribution over all grid-cells in the environment, showing how many percent of the total run time a cell is covered (30 Robots).

FIGURE 4.12

In high density case of 40 robots, the robots are forced, by the size of the environment, to have overlapping sensor and transmission areas. Therefore, at least some robots are moving, this triggers a chain reaction where one moving robot is driving in the transmission area of a stationary robot, subsequently triggering the moving strategy of that robot and so on. The continuous motion in BeePCo can be seen from Figure 4.13a. A summary over four different settings is presented in Figure 4.14, which shows in more detail how the number of robots increases the rate in which BeePCo's area coverage is growing. The rate between the 30 and 40 robot setting stays almost the same, this indicates subsequently that the growth rate is at its maximum. StiCo's area coverage rate stays for most experiments, except for the 10 robots setting, the same. In the last mentioned setting is the growth of the area coverage a little bit slower, since the



(A) The average percentage of the total area coverage, shown for each time step of the run (40 Robots).



(B) Distribution over all grid-cells in the environment, showing how many percent of the total time a cell is covered (40 Robots).

FIGURE 4.13

environment is less cluttered, so that a robot takes a longer to reach a position where it senses pheromone of other robots, this results in a slower spread overall. The difference between the maximal coverage achieved by BeePCo and StiCo, compared to the maximal possible coverage MaxCo increases with the number of robots (see Figure 4.14). This effect occurs because the environment becomes more cluttered and robots are getting surrounded by other robots, forcing them to choose a movement action, which will transition them to a state where their sensor areas overlap.

Figures 4.10b - 4.13b illustrates the distribution over the cell coverage, the same data represented in the heat-maps, in a box-plot format. As described in Subsection 4.4.1 the area is divided into a fine-grained grid. During the experiment runtime, an evaluation routine counts the time steps where a certain cell was covered by at least one robot

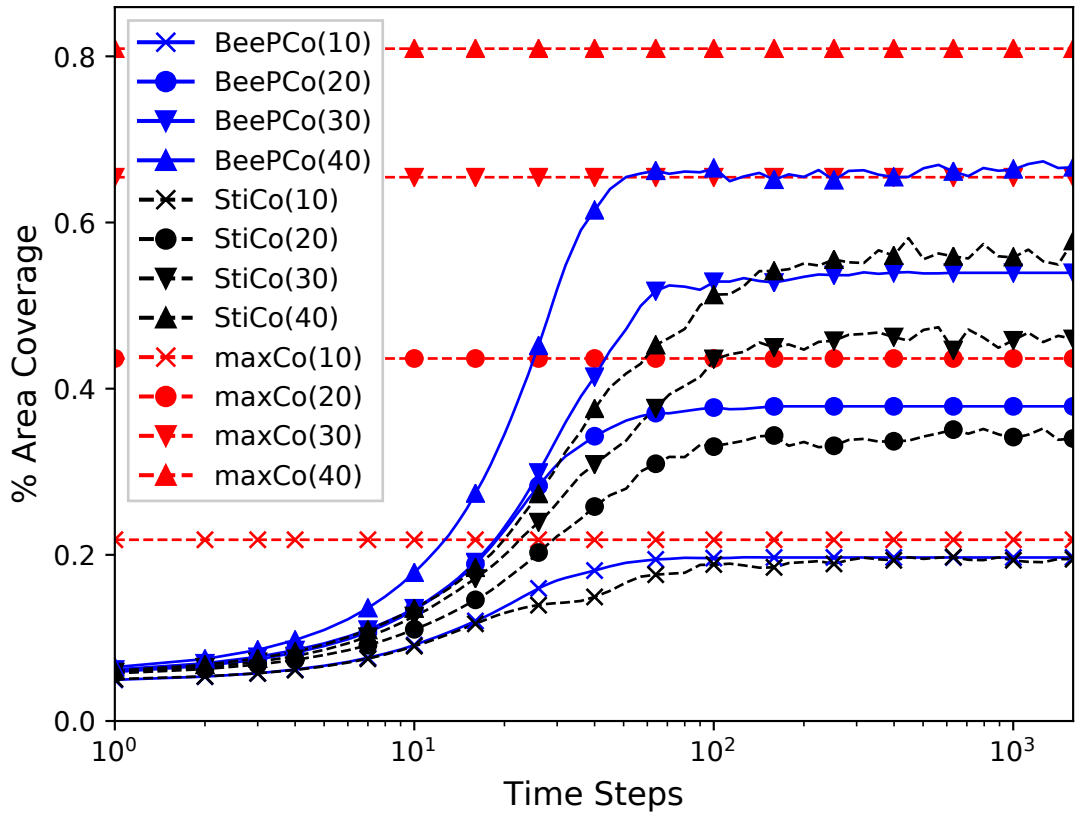


FIGURE 4.14: This plot shows the competing percentages of area coverage, using MRSs with 10, 20, 30 and 40 robots: StiCo and BeePCo

sensor. By dividing this counter, by the total number of time steps, we derive the percentage of the runtime, a specific cell is covered. In the box-plot format, the box shows the region where 75% of the data is located, the red-line marks the median and lines at the top and bottom of the box show the minimal and maximal values of the data set. Extreme outliers are represented by circles. The upper and lower boundary of the box are usually known as upper and lower quartile and the difference between those to values is refereed to as interquartile range.

Figures 4.11b and 4.12b also show that BeePCo converges for 20 and 30 robots to a steady state where it doesn't cover the cells evenly, shown by the large difference between the two quartiles. Figure 4.10b shows that with a robot size of 10, more than 75% of the cells have a run time coverage of around 0%. Figure 4.13b shows again that BeePCo covers the area more evenly in the 40 robot setting. The difference between the quantiles is only 20%, in comparison to the 90% of the other settings. StiCo shows in the configurations of 20, 30 and 40 robots an even coverage per cell, where the difference between the upper and lower quantile is between 10% and 15%, with an increasing median proportional to the number of robots, illustrated in Figures 4.11b - 4.13b. In the setting of 10 robots StiCo is not able to spread across the whole area, also shown in heat-map Figure 4.6a. This results in a more unevenly covered environment with a interquartile range of over 30%, see Figure 4.10b.

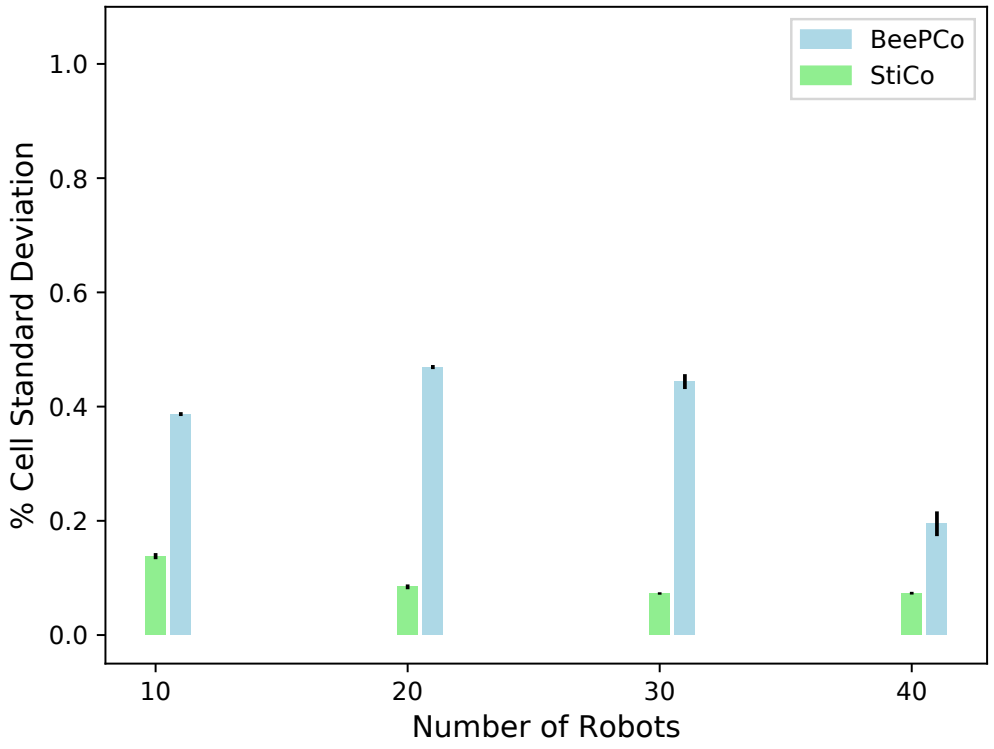


FIGURE 4.15: Standard deviation over all cells, for 10, 20, 30 and 40 robots.

The remaining Figure 4.15 shows additional how evenly the environment is covered, by illustration the standard deviation of the coverage between the cells. The lower the standard deviation the more even the environment is covered. As described before is BeePCo able to maximise the area coverage compared with StiCo, but is not covering the cells evenly over-time, which causes the higher standard deviation. In order to determine if the difference between the result is significant, we apply the Mann-Whitney U-Test and propose the required null hypothesis as follows:

> H_0: *StiCo's and BeePCo's cell coverage has the same standard deviation between the cells.*

Table 4.3 shows the result of the test for all four robot configurations. The p-value resulting from this test, is defined as the probability of the null hypothesis being satisfied. All results show a significant difference, since the p-values of the the Mann-Whitney U-test are under significance level $\alpha = 0.05$ and we can reject the hypothesis.

Number of Robots	p-Value	H_0
10	0.0004	reject H_0
20	0.0004	reject H_0
30	0.0004	reject H_0
40	0.0027	reject H_0

TABLE 4.3: Mann-Whitney U-test regarding the cell coverage standard deviation between StiCo and BeePCo. A $p < 0.05$ indicated a significant difference between the results and rejects the hypothesis.

4.5 Conclusions

In this chapter we are address the research question:

Research Question RQ1: To what extent can we devise appropriate algorithms for stigmergy-based/bio-inspired multi-robot exploration/coverage, in simulation and ground-based robots?

We devise the novel bio-inspired coverage algorithm BeePCo, based on the simple communication strategies seen in bees, to address the coverage problem in MRSs. We examine its coverage performance with the comparable coverage algorithm StiCo. We show that it is able to reach a high maximal sensor coverage and a fast converging speed. Since its simple rule based strategy is satisfied after its maximum sensor coverage is reached, it will eventually stop moving. This makes BeePCo attractive for blanket coverage use-cases e.g. in WSNs, but is therefore not suited for continues scanning scenarios. Because of its fast converging speed and its high maximal sensor coverage, we employ it as a foundation for a hybrid-pheromone coverage approach described in the following Chapter 5.

Chapter 5

Hybrid Bee and Ant Inspired Coverage

The previous chapter introduced the BeePCo algorithm for coverage and showed its strengths and weaknesses, compared it to and an ant pheromone based coverage approach called StiCo. This chapter elaborates over the differences between those algorithms and introduce the hybrid approach HybaCo, which combines the ant- and bee-pheromone algorithms to improve the performance with respect to converging speed and uniform sensing of the environment, also known as repetitive- or sweep-coverage. The remainder of this chapter is structured as follows: Section 5.1 illustrates the differences between the BeePCo and StiCo approaches. Section 5.2 explains the proposed bee and ant inspired HybaCo technique. The chapter continues with the experimental setup and results in Section 5.3. The results are further discussed and generalised in Section 5.4 and the chapter is concluded in Section 5.5.

5.1 Comparison Between StiCo and BeePCo

In order to understand the improvements by the hybrid approach over BeepCo and StiCo. Section 5.2 discusses the results and behaviours shown in the previous Chapter 4. Section 5.1.1 provides an overview of the differences between BeePCo and StiCo and Section 5.1.2 and 5.1.3 explain how these differences influence their performance.

5.1.1 Characteristics Differences Between BeePCo and StiCo

The major differences between BeePCo and StiCo are the communication- and movement principles each approach relies on. StiCo applies an indirect communication scheme, where the robots place physical pheromones in the environment to inform their cooperating agents of their recent visited positions. The robots move on a path described by a circle with a fixed radius. StiCo employs two sensors, one on the right and one on the left of the robot. If pheromone is detected on the outside of the circle-path, the robot will switch the rotation direction, e.g. from clockwise to counter-clockwise. If pheromones

are detected on the inside of the circle path, the robot will rotate until it doesn't sense the pheromones anymore. This behaviour results in spreading of the robots and can be used in the context of exploration and surveillance. For more detail, see Section 3.4.

In BeePCo, the communication is done in a direct fashion, which means the robots have a communication link to exchange pheromone messages between neighbouring robots. Similar to pheromones in a bee-hive, these pheromone-messages are propagated through the swarm to allow for an information exchange, over further distances than the communication range of a single robot. The propagation can also be classified as indirect communication. The messages itself can be seen as virtual pheromone and include a position, relative or global, indicating covered areas in the environment. Its intensity is decreased with increasing distance and time. Based on this gathered information, BeePCo is choosing the shortest path/straight line to the closest uncovered area. Overall the discussed features can be summarised in the following Table 5.1.

Property	*StiCo*	*BeePCo*
communication	Indirect (Pheromone)	Direct/Indirect (Messages)
movement	Circular	Vector-based

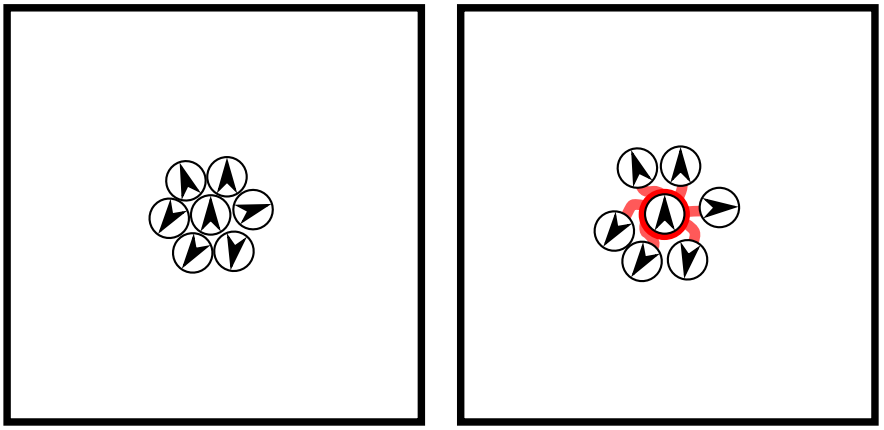
TABLE 5.1: Differences between *StiCo* and *BeePCo*

5.1.2 StiCo Coverage Performance

The experiments show that StiCo (Section 3.4) takes overall a longer time to converge to its maximal sensor coverage, mostly due to two main reasons: 1) StiCo moves in a circular fashion, which increases the path length in comparison to a straight line movement. 2) StiCo slows down in the beginning of a run, which is due to the additive pheromone and the initial configuration of the robots. Since the robots are initialised as a cluster, they start spreading pheromones, which surround the robots in the middle of the pack (see Figure 5.1). This triggers both line-detection sensors of those robots, resulting in a continuous switching of the circle direction. These robots are stuck in this behaviour until pheromones around them are evaporated.

StiCo's main advantage is that it is able to uniformly cover and sense the environment. How uniformly the environment is covered depends on the density of robots in the environment. In settings with a lower number of robots, StiCo converges to a state where each robot is disjoint from the others and each robot is surveilling an area around a fixed point in the environment. In the context of StiCo, are two robots disjointed, when their pheromone trails have no intersection and they are consequently not able to influence each other.

This maximises the sensor-coverage, but doesn't allow for a uniformly covered environment. With an increasing number of robots, the robots are starting to influence each other more, which introduces frequent avoidances, allowing StiCo to evenly cover the environment.



(A) Beginning (B) After a few time-steps

FIGURE 5.1: StiCo problem: Robot surrounded by pheromone.

5.1.3 BeePCo Coverage Performance

The experiments of BeePCo show that it converges quicker to its maximal sensor coverage than StiCo, which is due to the straight line movement compared to the circular movement of StiCo. BeePCo achieves higher coverage than StiCo, but converges to a steady state where the robots are not influenced by each other. This effect comes from BeePCo's notion to avoid the sensor-range of the other robots to maximise the total coverage. Since communication and sensor range are the same (in the considered case), the robots lose their communication links. After BeePCo looses all communication links it decays the virtual pheromone until it is not affected by it any more.

Compared to StiCo, BeePCo does require a much higher robot density to introduce a continuous movement, since it tries to reach the before described steady state. This property is sufficient in a context of blanket coverage, but is not suitable for an even coverage of the environment.

5.2 HybaCo: Hybrid Bee and Ant Pheromone Coverage

The goal of the Hybrid Bee and Ant Pheromone Coverage (HybaCo) is a simple merge of the two pheromone-based approaches BeePCo (Section 4) and StiCo (Section 3.4) to combine their strengths, while overcoming their individual weaknesses. In the previous Chapter 4, BeePCo showed it converges quickly and is able find configurations to maximise the coverage, but it requires communication links to provide a continues sweeping over the environment. Additional, StiCo is able to provide an even coverage with a lower robot density, but has a slower converging speed and lower maximal coverage.

Therefore, HybaCo aims to combine best of both worlds by preferring the BeePCo-principle as long as a robot still "senses" bee-pheromone. Allowing a faster spread in the beginning and a more calculated avoidance of densely packed areas. As soon as the bee-pheromone is decayed HybaCo switches to the StiCo-principle and employs its

circular motion and ant-pheromone to enable a more even spread of the robots. If a robot receives new bee-pheromone messages it will switch back to the BeePCo-principle, until the virtual pheromone is decayed and so forth. The pseudocode for HybaCo is shown in Algorithm 10.

Algorithm 10: Hyba-Co Algorithm

```
loop
  if Bee-pheromone present then
    Apply BeePCo
  else
    Apply StiCo
  end if
end loop
```

5.3 Experimental Evaluation

This section evaluates HybaCo in different scenarios. Section 5.3.1 explains the test-environment and configurations, Section 5.3.2 shows HybaCo's performance in respect to coverage, Section 5.3.3 analyses how evenly it is able to cover the environment over time and Section 5.3.4 illustrates how it utilises the underlying pheromone approaches, depending on the setting it is in.

5.3.1 Experimental Setup

HybaCo is designed to combine the high coverage and converging speed of BeePCo with the ability to evenly cover the environment of StiCo. This section analyses to what extent HybaCo succeeds in this. For a better comparison we evaluate the algorithm with the same number of robots and in the same environment used for the evaluation of BeePCo, described in Section 4.4. These parameters can be summarised by a square environment with a size of 300cm $\times$ 300cm and experiments with 10, 20, 30 and 40 robots. The evaluation is done in simulation and the employed simulator structure is described in Subsection 4.3. The simulated robots are based on the E-Puck robots and have a transmission- and sensor-range of 25 cm (see Section 4.4.2). We analyse HybaCo's maximal coverage ability, converging speed and coverage distribution over time. The maximal possible sensor coverage is in this section referred to as MaxCo and is defined as the maximal area all robots combined can sense. The distribution of the scanned positions over time is visualised by the heat-map representation, introduced in Section 4.4.1. This representation divides the environment in equally size grid-cells and applies a gradient color scheme, to each cell, according the percentage of total runtime, this specific cell is covered. The ligther the color the higher the percentage. To ensure statistical significance StiCo, BeePCo and HybaCo are each executing 30 runs, with a runtime of 6 minutes. The initial positions are varied between the runs, but are kept the same for all algorithms.

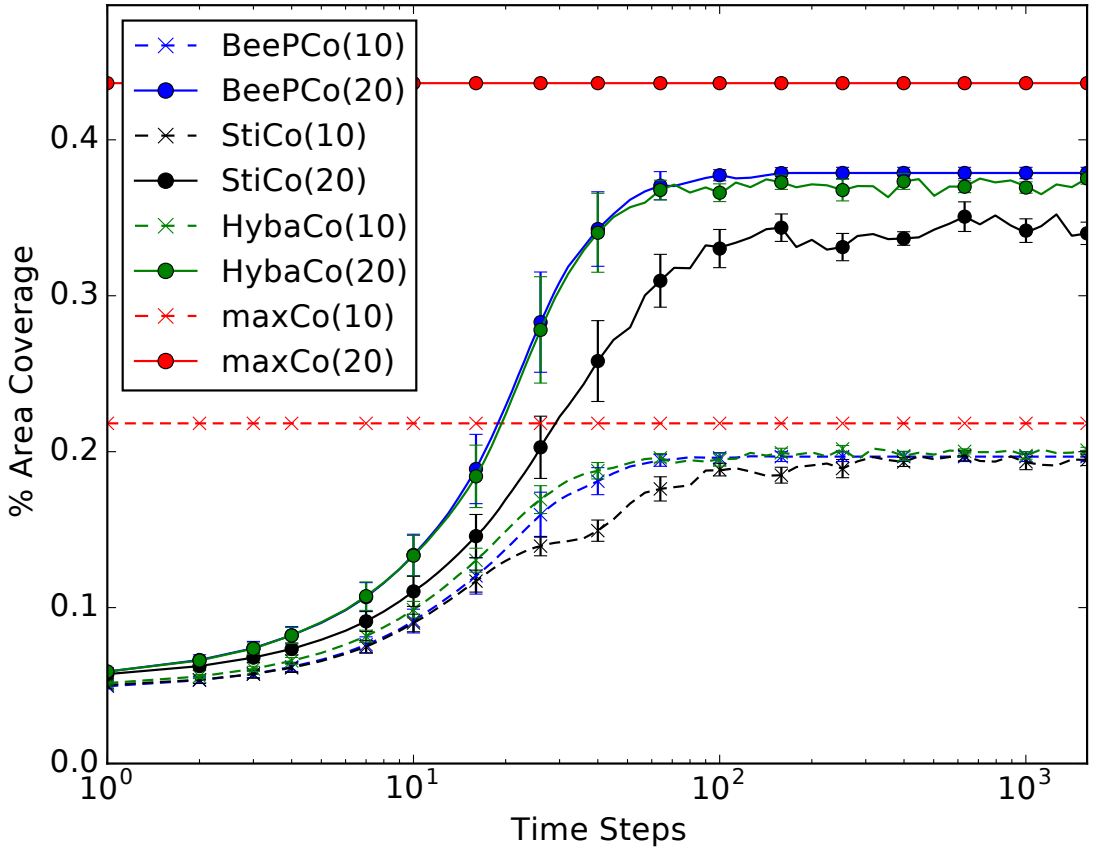


FIGURE 5.2: The percentage of area coverage using BeePCo, StiCo and HybaCo with 10 and 20 robots. The plot illustrate the mean and the error-bars the standard deviation over 30 runs.

Since BeePCo and StiCo are already examined in Section 4.4, they are mainly used as a reference in the following experiments. HybaCo uses the same parameters for the virtual bee pheromone and physical ant pheromone as the BeePCo and StiCo implementations it is compared with. The configuration of all three algorithms are illustrated in Table 5.2. For more detail about BeePCo's parameters, see Section 4.2 and for more detail about StiCo's parameter, see Section 4.4.3.

BeePCo	**StiCo**	**HybaCo**
$Threshold_{hopcount} = 10$ $Threshold_{movement} = 0.001$ $Threshold_{QR} = 10,000$ $T_{QR} = 0.2\,\text{sec}$ $ht = 0.4\,\text{sec}$	$P_{decay} = -0.1\,\text{units/s}$ $r_s = 25\,\text{cm}$	$Threshold_{hopcount} = 10$ $Threshold_{movement} = 0.001$ $Threshold_{QR} = 10,000$ $T_{QR} = 0.2\,\text{sec}$ $ht = 0.4\,\text{sec}$ $P_{decay} = -0.1\,\text{units/s}$ $r_s = 25\,\text{cm}$

TABLE 5.2: BeePCo's, StiCo's and HybaCo's configuration

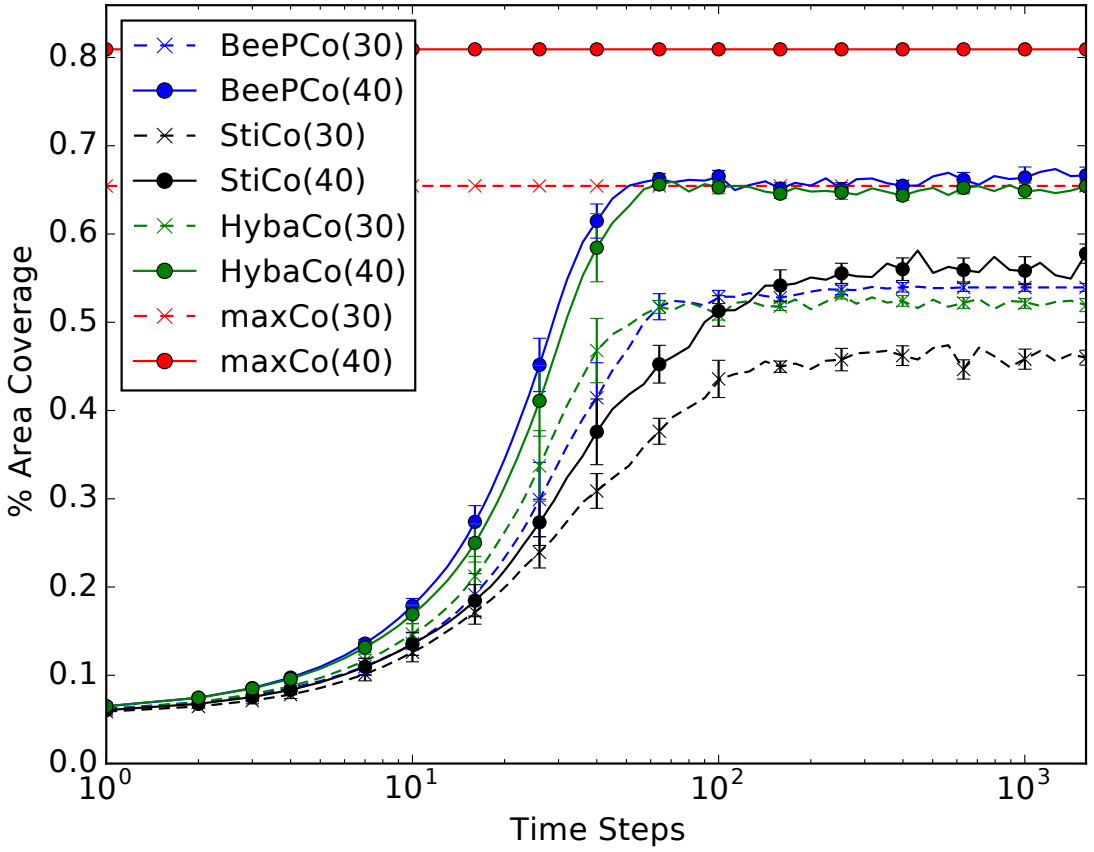
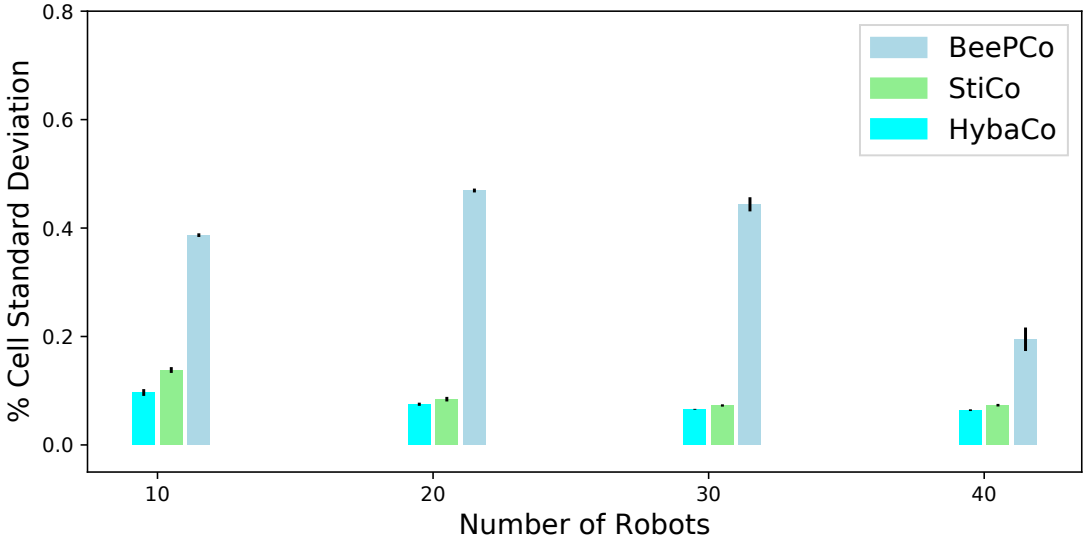


FIGURE 5.3: The percentage of area coverage using BeePCo, StiCo and HybaCo with 10 and 20 robots. The plot illustrate the mean and the error-bars the standard deviation over 30 runs.

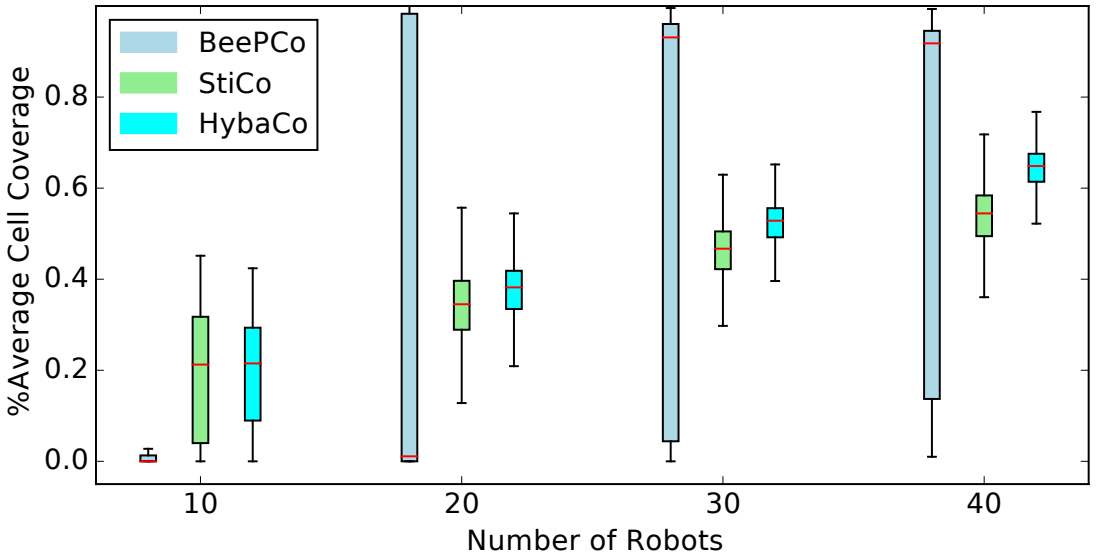
5.3.2 Sensor Coverage

Figure 5.2 and Figure 5.3 show the coverage progression of StiCo, BeePCo and HybaCo over the duration of a run. The graph shows the mean over all 30 runs and the error-bars illustrates the standard deviation. In the settings of 10 robots, it can be seen that HybaCo converges around 10 time-steps quicker than BeePCo, this difference is about 2 seconds of runtime (see Figure 5.2). The same holds for the setup of 20 and 30 robots, in the remaining setting of 40 robots. BeePCo has an advantage of 2 seconds over HybaCo, but these differences are insignificantly small and show that HybaCo is able to match the converging time of BeePCo. The convergence rate, defined as the area gained per time unit, is in both algorithms almost identical. After a small difference in the first 10 to 20 seconds, BeePCo's and HybaCo's plots show the same slope until they approach their maximum (Figure 5.2 and Figure 5.3).

With respect to maximum coverage, HybaCo and BeePCo converge to a similar level of covered area. Both approaches achieve in the settings of 10 and 40 robots a sensor coverage of 18%, respectively 65%. In the experiments with 20 and 30 robots, BeePCo has a slight advantage of 0.5% to 1% more coverage than HybaCo. The reasons for converging to the same coverage level are different for the scenarios of 10 and 40 robots.



(A) Standard deviation over the coverage per cell in the settings of 10, 20, 30 and 40 robots for BeePCo, StiCo and HybaCo.



(B) Distribution over the coverage per cell in the settings of 10, 20, 30 and 40 robots for BeePCo, StiCo and HybaCo.

FIGURE 5.4

In the setting of 10 robots, the environment is less densely packed with robots, allowing BeePCo and HybaCo to find a state where the coverage is maximised and the robots are not interfering with each other. This results in both cases to approximately the same of coverage, shown in Figure 5.2.

In the setting of 40 robots it's exactly the opposite. The robot density is high and a few robots will always be in communication range of one another. Robots in communication range propagate virtual pheromone, which triggers BeePCo's movement strategy. This movement propagates through the whole swarm, resulting in a continues motion. Since HybaCo prefers BeePCo's movement strategy, and BeePCo is generally

active (setting of 40 robots), is has approximately the same overlap as BeePCo with 40 robots (Figure 5.3).

In the cases of 20 and 30 robots, BeePCo is able find a steady state, where the robots are not interfering with each other, but HybaCo's continuous motion, a consequence of ant-pheromone strategy, increases the probability of overlapping sensor areas. This probability increases with a rising robot density, explaining the small coverage difference, for 20 and 30 robots, between HybaCo and BeePCo (Figures 5.2 and 5.3).

5.3.3 Distribution Over Time

In terms of distribution over time, HybaCo is able to improve upon the performances of StiCo and BeePCo. The heat-maps (Figure 5.5) show that it overcomes the steady states of StiCo and BeePCo in the 10 robot scenario. In the 10 robots setting StiCo and BeePCo are converging to a state where they patrolling/observing the same positions. This is shown by the red circles in Figure 5.5a and the white circles in Figure 5.5b. In this setting HybaCo shows an even distributed coverage over the environment (Figure 5.5c). This is also illustrated in the box-plot Figure 5.4b and standard deviation shown in Figure 5.4a, where the interquartile range and the standard deviation are smaller compared to StiCo's. The heat-map Figure 5.5c with 10 robots illustrates that HybaCo scans the environment more evenly, but still has some uncovered areas. Robots in this setting are able to find areas where they are not influenced by bee or ant pheromones.

Starting with 20 robots HybaCo is able provide an even scan over the whole environment, while improving on StiCo's standard deviation about 3% to 5% (Figure 5.4a) and improving its interquartile range, shown in the box-plot Figure 5.4b, by approximately 5%. With an increasing number of robots HybaCo is able to improve the average count each cell is covered, while still maintaining the low deviation between the cells counts (Figure 5.4a). This is also shown in the heat-map visualisation where colouring is getting brighter proportional to the robot count.

The overall explanation for the HybaCo performance is that by applying the principle of BeePCo it is able to spread faster in a formation where the coverage is maximised. After this phase, HybaCo's bee-pheromones are decayed and it switches to the circular motion of StiCo. By marking the covered area with the physical ant-pheromone a robot indirectly informs nearby by robots about its presence. This introduces an additional separation of the robots and an even spread across the environment. If robots move in each others communication range, HybaCo applies the linear movement of the BeePCo-principle to separate the robots faster, and in a more effective manner, compared to StiCo. This is particularly helpful when a group of robots collides.

One of the main goals of HybaCo is to achieve the even distributed coverage of StiCo, while improving on the maximal sensor coverage by applying the BeePCo-principle. As described before, HybaCo shows a slight improvement in regards to standard deviation between the cells (Figure 5.4a) in all settings and it improved in regards to the maximal coverage for the setting of 20 to 40 robots. In order to see if the improvements are

significant we apply the Mann-Whitney U-test and compare the cell standard deviation and the average cell coverage between StiCo and HybaCo. For the former we test the hypothesis:

> H_0: *StiCo's and HybaCo's cell coverage has the same standard deviation between the cells.*

The p-value resulting from this test, is defined as the probability of the null hypothesis being satisfied. The test shows for all four setting a value $p < 0.05$ (see Table 5.3), which shows significant difference between the results and we can reject the hypothesis in all setting.

Number of Robots	p-Value	H_0
10	0.0004	reject H_0
20	0.0341	reject H_0
30	0.0011	reject H_0
40	0.0015	reject H_0

TABLE 5.3: Mann-Whitney U-test regarding the cell coverage standard deviation between StiCo and HybaCo. A $p < 0.05$ indicated a significant difference between the results and rejects the hypothesis.

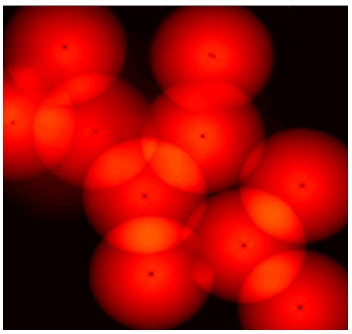
In regards to the average cell coverage (Figure 5.4b), we apply the hypothesis:

> H_0: *The average percentage a cell is covered during a run is the same for HybaCo and StiCo.*

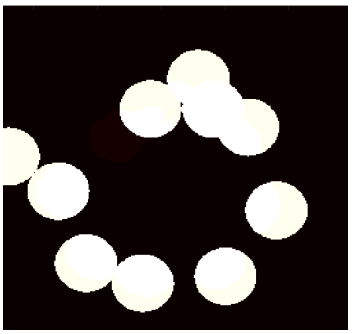
As shown in Figure 5.4b are the results significant different for 20 to 40 robots and are drawn from a similar distribution for the setting of 10 robots. This observation is also confirmed with the Mann-Whitney U-test, the results of the test are illustrated in Table 5.4. This results confirms that that HybaCo is able to achieve a higher sensor coverage than StiCo, in denser robot setting.

Number of Robots	p-Value	H_0
10	0.6588	don't reject H_0
20	0.0104	reject H_0
30	0.0004	reject H_0
40	0.0004	reject H_0

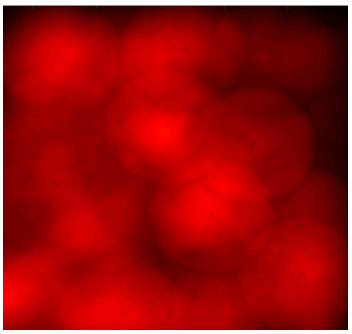
TABLE 5.4: Mann-Whitney U-test regarding the average cell coverage between StiCo and HybaCo. A $p < 0.05$ indicated a significant difference between the results and rejects the hypothesis.



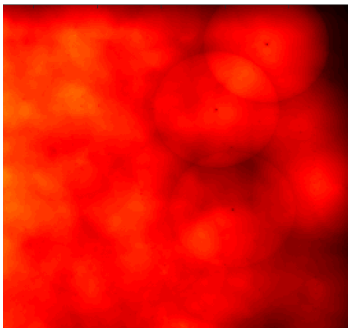
(A) StiCo - 10 Robots



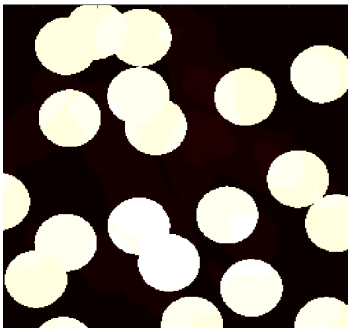
(B) BeePCo - 10 Robots



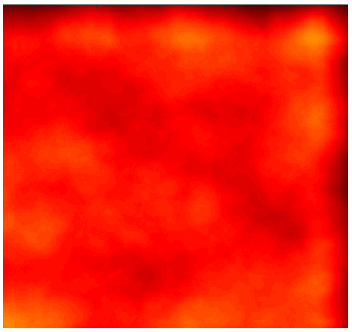
(C) HybaCo - 10 Robots



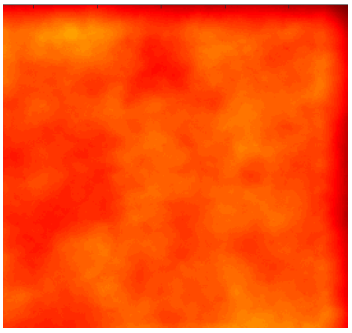
(D) StiCo - 20 Robots



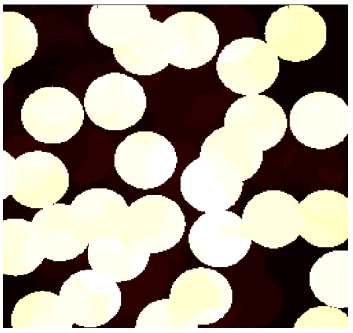
(E) BeePCo - 20 Robots



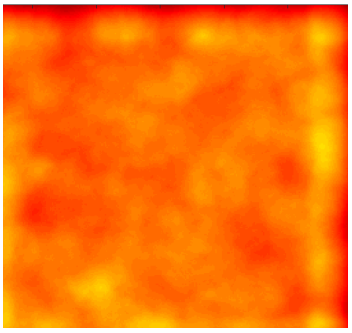
(F) HybaCo - 20 Robots



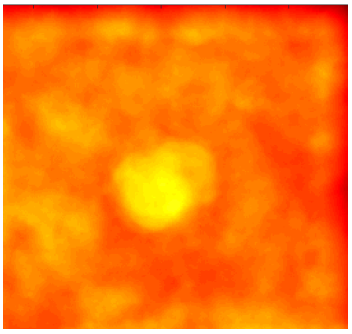
(G) StiCo - 30 Robots



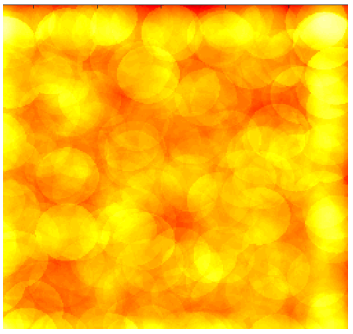
(H) BeePCo - 20 Robots



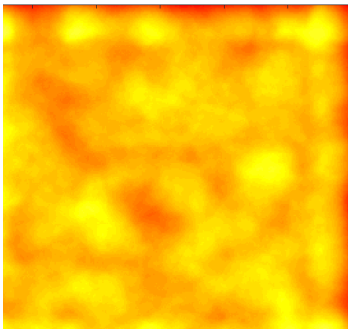
(I) HybaCo - 30 Robots



(J) StiCo - 40 Robots



(K) BeePCo - 40 Robots



(L) HybaCo - 40 Robots

FIGURE 5.5: The distribution over time, using 10, 20 ,30 and 40 robots. The coverage is illustrated in a gradient color scheme. A cell is coloured white, if it is covered 100% of the run time and coloured black for a 0% coverage. The intermediate percentages are coloured accordingly to the scale at the top of the figure.

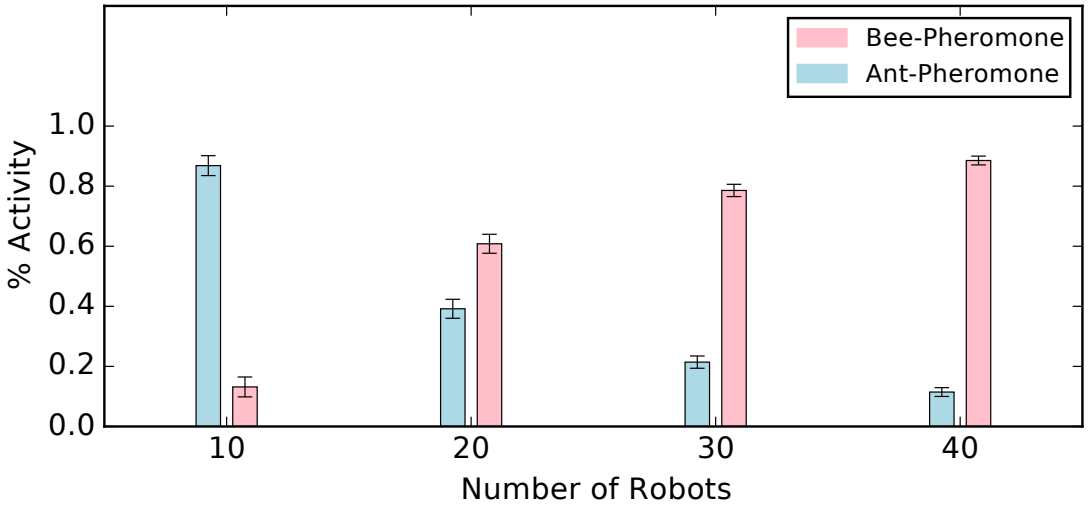


FIGURE 5.6: The percentage of the total run-time, the bee- and ant-pheromone algorithms are active.

5.3.4 Ant and Bee Pheromone Usage

HybaCo combines the ant-pheromone technique of StiCo together with the bee-pheromone technique of BeePCo to surveil the environment in a continuous fashion. The previous Sections 5.3.2 and 5.3.3 showed that HybaCo is able to match and outperform BeePCo and StiCo in maximal coverage and distribution over time. This section analyses how the robot density or number of robots influences the usage of the ant-and bee-pheromone approaches inside HybaCo.

Figure 5.6 illustrates the total run-time of each approach in HybaCo, with respect to the number of robots. Figure 5.7 shows the time of operation, of each approach, over the course of a run. As described in Section 5.2, the pheromone approaches HybaCo relies on, are mutually. Specifically, the ant-pheromone approach is deactivated while the bee-pheromone approach is active and vise versa.

The experiments show that HybaCo's usage of the underlying pheromone approaches depends on the robot density in the environment, the higher the density, the higher the total operation-time of the bee-pheromone approach. In the case of 10 robots, HybaCo uses the bee-pheromone approach around 15% of the total run-time, this number increases proportional to the number of robots and reaches a maximum of 85% in the setting of 40 robots (see Figure 5.6).

Figure 5.7 presents typical runs of a robot in regards to different swarm sizes, it shows which pheromone approach is active at a specific point in time. In the beginning of all four scenarios (10 - 40 robots) the robots are applying the bee-pheromone principle. This is because the robots are initialised in a cluster, allowing all robots direct communication with at least one cooperating robot. After the initial spread, controlled by the bee-pheromone approach, the robots start switching to the ant-pheromone technique.

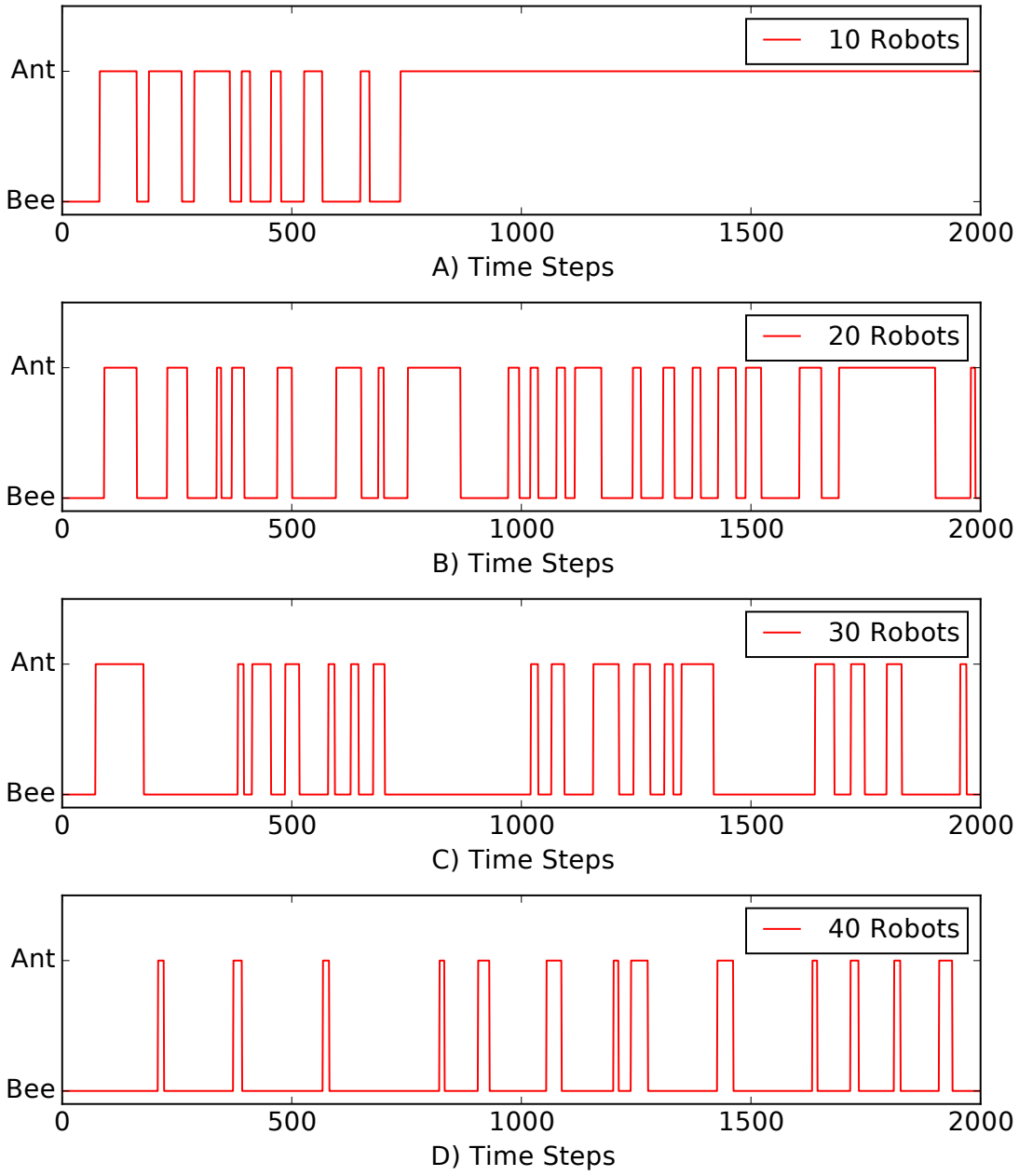


FIGURE 5.7: Activation-times of ant- and bee-pheromone principles during a single run of HybaCo.

An increasing number of robots, or higher robot density, reduces the duration and frequency the ant-pheromone principle is applied. Where HybaCo is almost fully relying on the ant-pheromone approach during the 10 robots setting (Figure 5.7a), it employs it short and sparsely in the 40 robots setting (Figure 5.7d).

This is due to the high density, which forces the robots closer together and therefore increasing the probability to activate of the internal bee-pheromone approach. If the environment is less densely packed, the robots are able to keep a safe distance between each other, by only using the ant-pheromone approach.

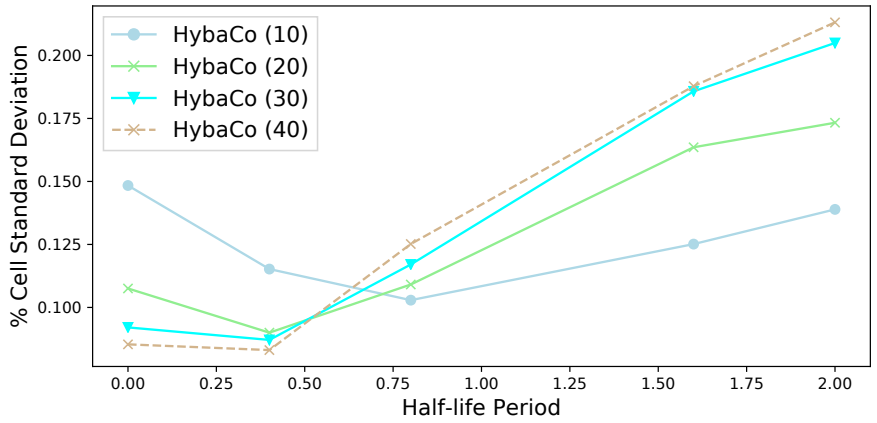


FIGURE 5.8: Illustrates the different standard deviations, between the cells, for different settings of HybaCo's half-life period parameter ht.

5.3.5 Pheromone-Decay Parameter Sensitivity and Tuning

The previous Sections 5.3.2 and 5.3.3 showed that HybaCo is able to outperform StiCo, the ant-pheromone principle it's based on. This is accomplished by additionally utilising the bee-pheromone communication technique BeePCo, the whole combination of both techniques is described in Section 5.2 in more detail. As described in Section 5.2, relies the BeePCo-principle, used in HybaCo, on different parameters to tune the algorithms overall performance (Section 4.2). This section will analyse the influence of the pheromone decay/half-time parameter ht, since its the most important parameter, because it determines the bee-pheromone algorithms duration, activated by the virtual pheromone. The half-time period of the pheromone is defined as the duration required to evaporate 50% of the current dosage.

Since the pheromones dosage decreases expectationally (Equation 4.2, Section 4.2) it can never reach exactly 0%. Because of this reason we define a the threshold value $Threshold_{movement}$, this threshold determines the percentage, which is consider as negligible. During all experiments we keep the threshold constant at $Threshold_{movement} = 0.001$ and change only the half-time period. The same result can be achieved by keeping the rate constant and changing the threshold, as long as the total evaporation time is kept the same. We define the evaporation time, as the duration to decay the pheromone from a 100% dosage to a dosage below the threshold $Threshold_{movement}$. The evaporation time can be calculated by solving the Equation 4.2, introduced in Section 4.2, for Δt:

$$\Delta t = h_t \frac{\ln(Threshold_{movement})}{\ln(0.5)} \tag{5.1}$$

In order to analyse how the virtual pheromone half-time/decay rate influences the algorithm's performance and how sensitive the parameter is, we employ scatter plots for different fitness functions of the algorithm. In order to illustrate how even the environment is covered, we use the standard deviation of the coverage between the cells

and to compare the maximal area the robots are covering, we compare the average sensor coverage over the duration of a run.

The experiments are done in the previous settings of 10, 20, 30 and 40 robots, and we examine their result for the half-time (ht) settings of: 0, 0.4, 0.8, 1.6 and 2.0 seconds. The range and step-size between the settings are sensibly chosen, in order to examine a significant increase in the parameter setting, since smaller step-sizes show no or minimal effect. Preliminary analysis showed that the performance is linear decreasing after the decay-rate exceeds a certain limit, which is the reason for the limited parameter range. The effect can be seen in Figures 5.8 and 5.9 and is further discussed in the following paragraphs.

All setting combinations are run 50 times in a 3m $\times$ 3m environment described in Section 4.4. A half-time period of 0 means that the virtual bee-pheromone evaporates directly and is not detectable, HybaCo is relying in this case exclusively on the underlying ant-pheromone principle (StiCo). This parameter setting is meant as a performance base-line, since HybaCo uses the additional BeePCo algorithm to improve on StiCo's performance. The 2 seconds half-time period on the other side of the spectrum, means that the pheromone dosage halves every 2 seconds and lingers therefore much longer in the system.

Figure 5.8 and Table 5.5 illustrate the different performances with regards to the standard deviation of the coverage between the cells (Section 4.4), the smaller the standard deviation the more equally the environment is covered. Starting with a half-time period of 0 seconds, the deviation decreases with increasing parameter value until a certain value is passed, the deviation is linearly increasing after this point. The deviation minima is sightly dependent on the robot density. For the denser setting of 20 - 40 robots, is the deviation increasing after parameter value of 0.4 seconds and for the 10 robots setting, it increases after 0.8 seconds.

		Number of Robots			
		10	20	30	40
Half-life Periode	0.0 sec	0.1484	0.1075	0.0921	0.0853
	0.4 sec	0.1212	**0.0960**	**0.0931**	**0.0831**
	0.8 sec	**0.0969**	0.1031	0.1109	0.1209
	1.6 sec	0.1191	0.1576	0.1801	0.1821
	2.0 sec	0.1329	0.1673	0.1989	0.2071

TABLE 5.5: Illustrates the different standard deviations, between the cells, for different settings of HybaCo's half-life period parameter ht.

A similar effect can be seen in respect to the coverage in Figure 5.9. With a half-life time of 0 seconds, the robots take longer to converge to the maximum coverage and it converges to the lowest coverage compared to all other settings, with a half-time value of 0.4 seconds, HybaCo converges quicker and reaches a higher coverage. For $ht > 0.4$ the

maximal sensor coverage becomes less stable and shows occasionally drops, this effect increases proportional to the ht value.

The reason for the improvement from $ht = 0$ to $ht = 0.4$ can be explained by the fact the a half-time period time of 0 means that the virtual pheromone evaporates immediately and is therefore not considered for the movement of HybaCo, it basically operates like StiCo. In the $ht = 0.4$ setting, HybaCo uses the additional information of the pheromone to spread quicker and more efficiently, as described in Sections 5.3.2 and 5.3.3. The linear degrease in performance for the settings $ht > 0.4$ is caused by the fact that the pheromone consist longer in the virtual view of the robot and the robot considers the area of the pheromone as covered, when it didn't receive an update of the robot who placed it. This means there is more pheromone to avoid, this forces more robots to the same pheromone free spaces, which results in less equally covered environment and more overlap of the sensor areas. If ht is smaller, the virtual pheromone evaporates quicker and the robots will explore these spaces again.

The analysis shows that the ht parameter influences HybaCo's performance in a stable and predicable fashion. In regards to the tuning, we can use the standard deviation as a fitness function and increase the ht parameter in a gradient decent fashion until we reach the minimum.

5.4 Discussion

Generally speaking HybaCo is able to match and outperform the two pheromone-based approaches StiCo and BeePCo it relies on. The performance itself is strongly influenced by the number of robots in respect to the size of the environment. Additionally, the performance is also effected by the communication range of the underlying pheromone principles, since both approaches trigger an avoidance behaviour if robots are in each others transmission ranges.

For the bee-pheromone principle (BeePCo) is the transmission range limited by the maximal range of the wireless communication device e.g. blue-tooth, WiFi and in our experiments infra-red transmission (25 cm). The underlying ant-pheromone principle (StiCo) is communicating indirectly via pheromone trails. Since StiCo is moving in circles, the communication range is the diameter of the circle.

Both algorithms aim to disjoint their transmission areas, to maximise the total sensed area. Since the sensor- and transmission range of the robots, in the considered test-scenarios, are the same.

In order to generalise the result of the experiments we define the robot density as the combined transmission area of all robots, divided by the two dimensional area size of the environment. For x robots with with a communication range of r, in a environment of size $n \times m$ this can be formalised as:

$$density = \frac{xr^2\pi}{nm} \tag{5.2}$$

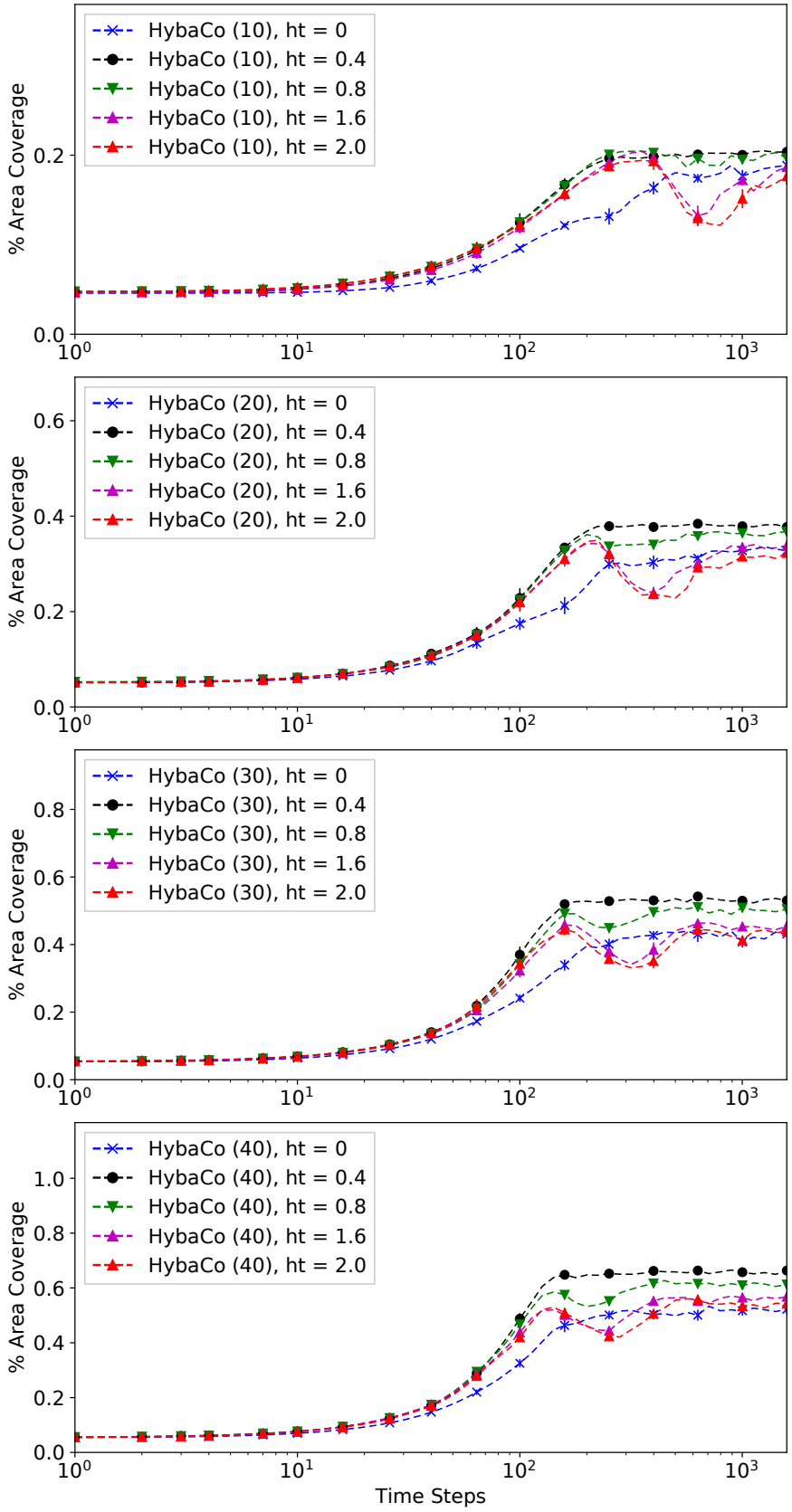


FIGURE 5.9: Illustrates HybaCo's sensor coverage performance for 10, 20, 30 and 40 robots over a run, with respect to different settings of HybaCo's half-life period parameter *ht*.

The experiments show that the initial rate, in which the algorithm gains coverage, reaches its maximum at 20 robots. This effect is caused by the number of robots and not by the environment size. Since a higher robot count increases the velocity in which the BeePCo principle avoids the virtual pheromone, the graphs show that this velocity reaches its maximum at a configuration of 20 robots (Figures 5.2 and 5.3).

A property affected by robot density, is the difference between the maximal achieved coverage and the maximal potential coverage. The experiments with 10 robots (21% density) show a maximal coverage of 2% under the potential coverage, where a density of 85% (40 robots) shows a difference of around 13%. This is because, a lower density gives the robots more options to avoid each other, where a denser environment forces the robots to move towards walls or other robots, which reduces the overall sensor coverage (Figures 5.2 and 5.3).

But this has also a positive effect on HybaCo, because it introduces more movement into the swarm, leading to a full coverage of the environment. Starting with a robot density of 40% (20 robots), it is able achieve uniform coverage, where each cell of the environment is covered for a similar duration. Higher robot densities than 40% affect the frequency a cell is scanned, but the deviation for the cell-scan-duration stays between 5% to 8% (Figure 5.4a).

This means HybaCo is able to continuously scan/sweep the environment if the density is over 40%. This is around 10% lower than StiCo, which required 30 robots to achieve even coverage. Lower densities than 40% are therefore sufficient, if a full coverage is not necessary and the main goal is the maximisation of the total sensor area.

5.5 Conclusions

This chapter addresses research question RQ1 and RQ2:

Research Question RQ1: To what extent can we devise appropriate algorithms for stigmergy-based/bio-inspired multi-robot exploration/coverage, in simulation and ground-based robots?
Research Question RQ2: What are the advantages of direct and indirect communication and to what extent can they be combined?

We compare BeePCo's direct communication with StiCo's indirect communication and analyse their differences in maximal sensor coverage and distribution of their coverage in Section 5.1. We show that StiCo's communication principle incentivises the robots to evenly spread across the environment, but doesn't use the sensor coverage to its full potential and shows inefficiency spreading behaviour in clustered scenarios. BeePCo is able to provide a fast spread and high maximal coverage, but is not able to cover the environment evenly.

We introduce the hybrid bee and ant-pheromone based coverage approach HybaCo with the aim to combine the strengths of the underlying principles StiCo and BeePCo in Section 5.2.

The experiments in Section 5.3, show that HybaCo is able to match the convergence speed and sensor coverage of BeePCo, while outperforming the well-distributed coverage over time of StiCo. Additionally we show how the number of robots affects the usage of the underlying pheromone principles and the performance overall. Finally, Section 5.4 discusses how the results can be generalised in respect of the robot properties and environment size.

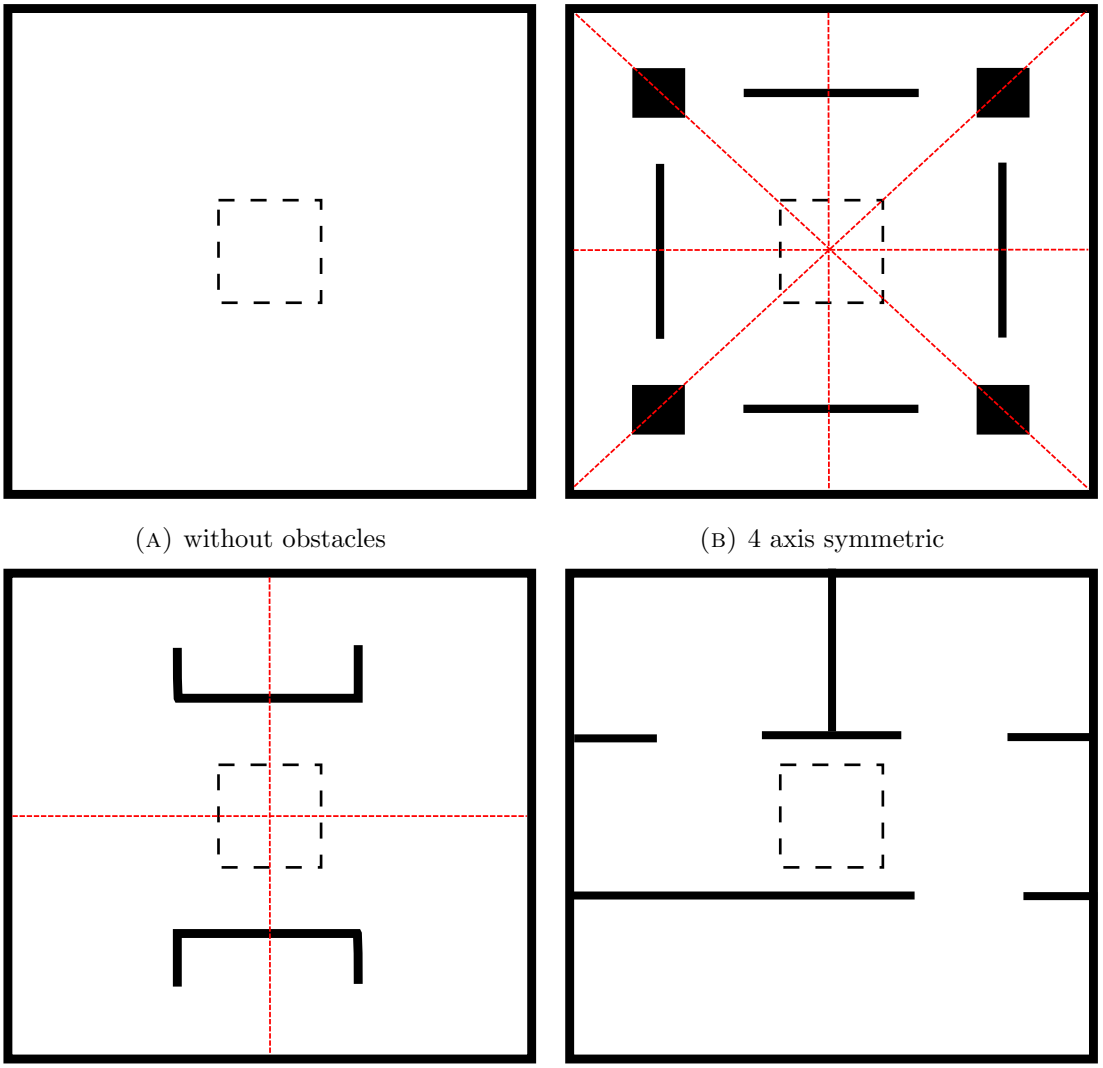
Chapter 6

Insect-Inspired Multi-Robot Coverage in Complex Environments

The previous Chapters 4 and 5 introduced and evaluated the pheromone based coverage approaches BeePCo and HybaCo. In order to investigate the general behaviour with respect to swarm size, the experiments were conducted in a simplistic square arena. This chapter extends these experiments and examines the stability and robustness of BeePCo and HybaCo in more complex environments. Section 6.1 describes the employed environments and their properties. Section 6.2 shows the experimental setup and results. The results are further discussed in Section 6.3 and the chapter is concluded in Section 6.4.

6.1 Environments

The chosen environments illustrated in Figure 6.1 are designed to analyse to what extent the algorithms performances are effected by the shape of the environment. Figure 6.1a shows the basic environment used in the previous Chapters 4 and 5 with a size of $300\,\text{cm} \times 300\,\text{cm}$ without any obstacles, it functions as a benchmark in the following experiments. The robots start positions are in all environments randomly initialised in a center square of the size of $50\,\text{cm} \times 50\,\text{cm}$, illustrated by the black dashed lines in Figure 6.1. The coverable area, is defined as the area that the robot sensors are able to cover. Meaning, the total area of the environment minus the area of all obstacles combined. To make the result between the environment more comparable we keep the coverable area constant for all environments at $90.000\,\text{cm}^2$ (area without obstacles).



(A) without obstacles

(B) 4 axis symmetric

(C) U-shapes

(D) Floor-plan

FIGURE 6.1: The four environments used in the experiments. The red lines show the symmetry axes of the environments and the black dash-lines illustrate the square the robots are initialised in.

Table 6.1 illustrates the different environments, with their total area, the combined area of the obstacles and the area coverable by the sensors.

Environment	Total Area	Obstacles Area	Coverable Area
Without obstacles	90 000 cm^2	0 cm^2	90 000 cm^2
U-shapes	91 500 cm^2	1500 cm^2	90 000 cm^2
Floor-plan	92 500 cm^2	2500 cm^2	90 000 cm^2
4-axis symmetric	95 000 cm^2	5000 cm^2	90 000 cm^2

TABLE 6.1: Environment areas

The second environment (Figure 6.1b) has mirror symmetries along the horizontal, vertical and the two diagonal axes, allowing the robots the same space and obstacle configuration in each octant of the environment. The symmetry axes are shown as red lines

in Figure 6.1. The obstacles are walls and pillars surrounding the initial starting positions of the robots. The walls and pillars are configured in a way that they allow small gaps to the outer realm of the environment. The environment is designed to analyse to what extent small gateways influence the even coverage of the complete environment. The symmetry of the environment makes each path through the gaps between the obstacle equally attractive. During this chapter we refer to this environment as "4-axes symmetric environment".

The third environment (Figure 6.1c) has symmetries across the horizontal and the vertical axes. The obstacles are u-shaped walls placed at the top and the bottom of the environment. The walls are rotated in a way that parts of the environment are hidden from the robots in their initial positions. The purpose of this configuration is to analyse whether the algorithms are able find the hidden space, or if they leave undiscovered gaps. We refer to this environment as "U-shapes environment".

The final environment seen in Figure 6.1d represents a floor plan, with two equally sized rooms at the top and one long room at the bottom. This environment shows how separated areas and an asymmetric environment effect the even coverage of the algorithms.

6.2 Experimental Evaluation

This section evaluates BeePCo's, StiCo's and HybaCo's performances in the environments described in Section 6.1. Section 6.2.1 explains the test-environment and configurations, Section 6.2.2 shows the performances with respect to coverage and Section 6.2.3 analyses how evenly the environments are covered over time.

6.2.1 Experimental Setup

In order to investigate to what extent the shape of the environment influences BeePCo's and HybaCo's performances, we maintain the parameters and configurations employed in the previous experiments (Sections 4.4 and 5.3). This includes the number of robots (10, 20, 30 and 40), the coverable area of 90.000 cm^2 (Section 6.1) and the simulated robots. The robots are based on the micro robots named E-Pucks with a communication and sensor range of 25 cm (see Section 4.4.2). E-Pucks use infra-red emitters and receivers for communication. Since this technology uses light for communication, robots are not able to communicate through walls or other non-transparent obstacles. The evaluation is done in simulation and employs the three tier simulation structure described in Subsection 4.3. We analyse HybaCo's and BeePCo's maximal coverage, convergence speed and coverage distribution over time in relation to the different environments. Since HybaCo is based on StiCo's and BeePCo's pheromone principles (Section 3.4), we additionally employ StiCo to analyse the performance differences between HybaCo and the algorithms it is based on.

To ensure statistical significance all algorithms and environments combinations are repeated 30 times with a runtime of 6 minutes.

Deploying three algorithms, four environments and four different group sizes we consider 64 different experimental settings. To reduce the number and the redundancy in plots in this section, we show the major influences of the environments on a representative subset (Section 6.2.2 and 6.2.3). The complete set of plots can be seen in Appendix A. In order to make the results comparable to previous experiments, we apply the same parameters for the virtual bee pheromone and physical ant pheromone approaches discussed in Sections 4.4 and 5.3. The parameters of the three algorithms are illustrated in Table 6.2:

BeePCo	**StiCo**	**HybaCo**
$Threshold_{hopcount} = 10$ $Threshold_{movement} = 0.001$ $Threshold_{QR} = 10,000$ $T_{QR} = 0.2\,\text{sec}$ $ht = 0.4\,\text{sec}$	$P_{decay} = -0.1\,\text{units/s}$ $r_s = 25\,\text{cm}$	$Threshold_{hopcount} = 10$ $Threshold_{movement} = 0.001$ $Threshold_{QR} = 10,000$ $T_{QR} = 0.2\,\text{sec}$ $ht = 0.4\,\text{sec}$ $P_{decay} = -0.1\,\text{units/s}$ $r_s = 25\,\text{cm}$

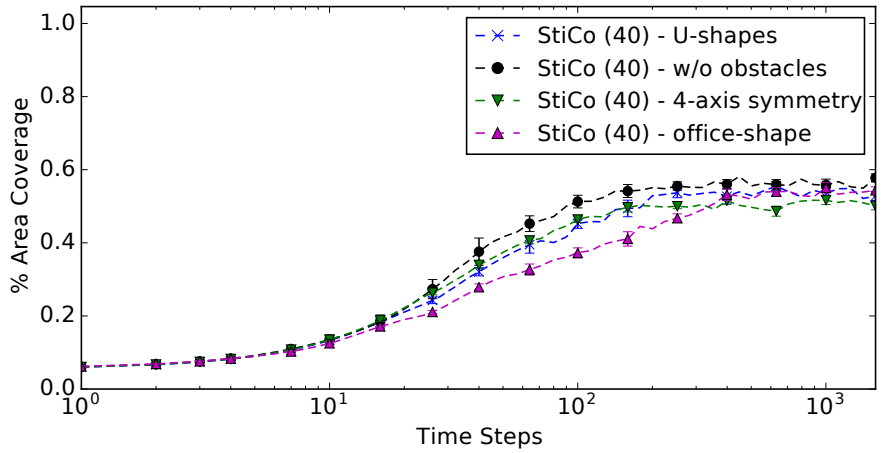
TABLE 6.2: Algorithm parameters

6.2.2 Sensor Coverage

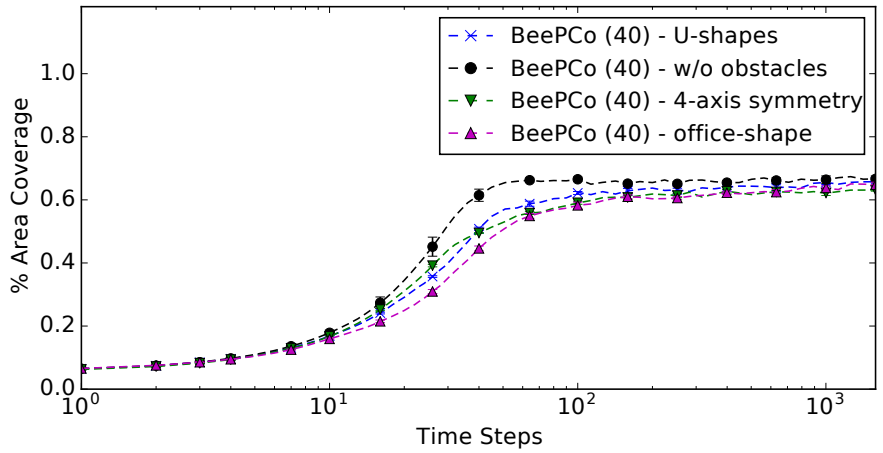
Sensor coverage is defined as the combined area, a group of robots is able to observe. This section analysis how obstacles in the environment influence the performance of BeePCo, HybaCo and StiCo.

Comparing BeePCo and HybaCo, it can be seen, that both algorithms achieve a similar performance in the obstacle related environments (see Figure 6.3). A comparison between the environments shows that BeePCo's and HybaCo's coverage is not significantly affected by obstacles in the environment (see Figure 6.2). The coverage decreases around $1\% - 4\%$, with the highest drop in the environment with the four symmetry axes (Figure 6.1b). This is due to two reasons: 1) The combined area of the obstacles is the highest (Table 6.1) and 2) the obstacles are close to the border of the environment. The former reason increases the probability of a collision with an obstacle and the latter complicates the avoidance of other robots in the outer realm. These factors force robots in positions where the sensor areas between robots overlap, or the sensor area is restricted by an obstacle. The obstacles in the other two environments (Figures 6.1c and 6.1d) are wider spread, resulting in a smaller drop of $1\% - 3\%$. StiCo is affected in the same way with a maximal drop of around $2\% - 6\%$ depending on the environment (Figure 6.2). Figure 6.2 and Figure 6.3 show additionally that the time it takes for the algorithms to converge depends on the environment shape. BeePCo and HybaCo have a similar converging time for the same environments (Figure 6.3), but the time differs between the environments (Figure 6.2). All three algorithms experience the highest increase in

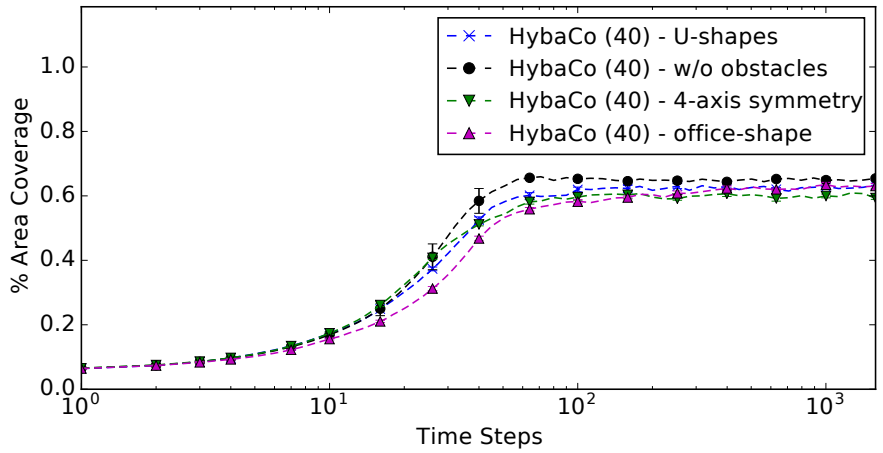
converging time, in the floor-shape environment (Figure 6.1d) and show a similar reduction, of the coverage gain, in the two symmetric environments (Figures 6.1b and 6.1c). The cause of the increase in time is further discussed in Section 6.3.



(A) StiCo

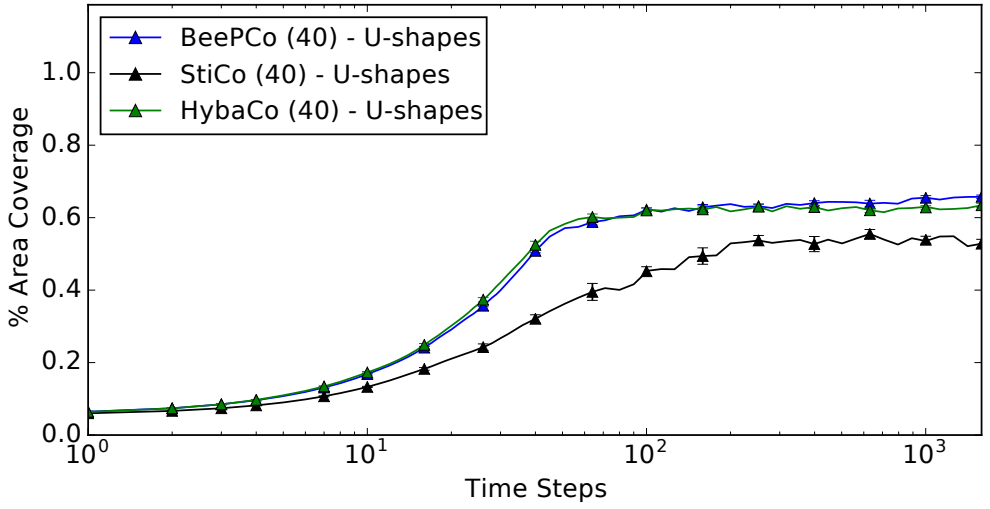


(B) BeePCo

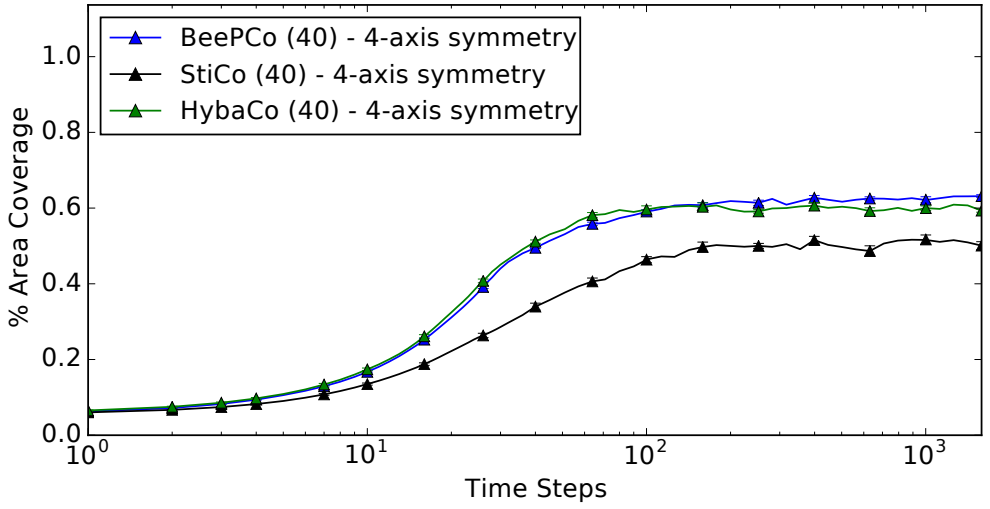


(C) HybaCo

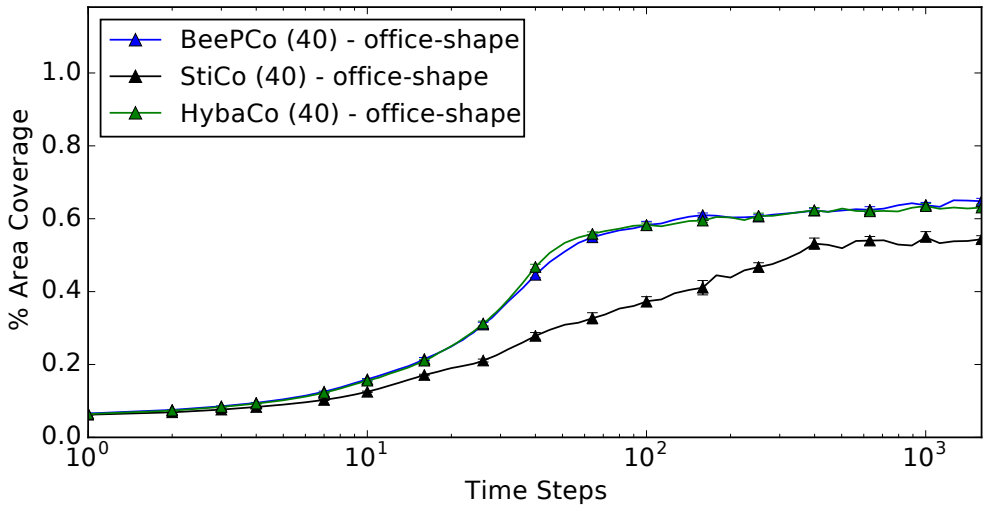
FIGURE 6.2: Compares the achieved area coverage between the environments, using 40 robots.



(A) U-shapes



(B) 4-axis symmetric



(C) floor-plan

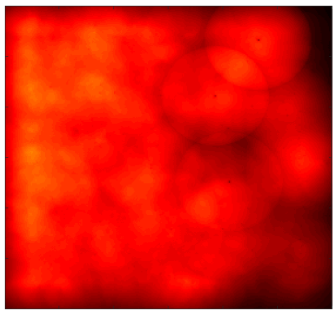
FIGURE 6.3: Compares the achieved area coverage between BeePCo, HybaCo and StiCo, using 40 robots.

6.2.3 Distribution Over Time

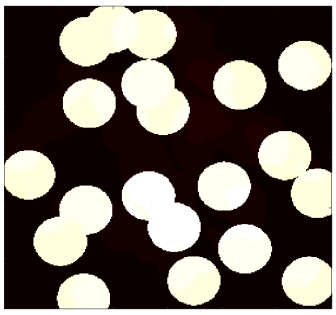
The distribution over time describes how uniform the environment is covered, over the duration of a run. We employ the technique explained in Section 4.4.1 to calculate the distribution, it divides the environment in a fine grid and measures the time-steps a certain cell is covered by a sensor. Figures 6.4 and 6.5 visualise these distributions as heat-maps. The cell-coverage is presented in a gradient color-scheme, which ranges from black (0% coverage) to white (100% coverage). Additionally, Figure 6.6 and 6.7 illustrates standard deviation of the coverage count over all the cells.

The experiments show that BeePCo's distribution is marginally affected by obstacles in the environment and matches the performance discovered in the empty environment. For a robot count of $10 - 30$ robots it converges to a state where the robots eventually are not influenced by the virtual pheromone and the regions the robots surveil are disjointed. This effect is shown by the white circles in the heat-map Figures 6.4b, 6.4e, 6.4h and 6.4k. This maximises the sensor coverage, but doesn't allow for an evenly covered environment, shown by the high standard deviation in Figures 6.6 and 6.7b. In the higher robots density setting of 40 robots BeePCo is not able to find a configuration where all robots are disjointed (Figures 6.5b, 6.5e, 6.5h and 6.5k). This results in a continues movement propagating through the swarm. The behaviour occurs in all four environments, illustrated in the heat-map Figures 6.5b, 6.5e, 6.5h and 6.5k. Through this movement is BeePCo able to cover the environment more evenly, reducing the standard deviation between the cell counts (Figure 6.7b). The deviation between the environments is marginally higher in environments with obstacles, compared to deviation archived in the empty environment (Figure 6.7b). This is based on the fact that the robots are not able to communicate or move through obstacles, which consequently inhibits the pheromone propagation.

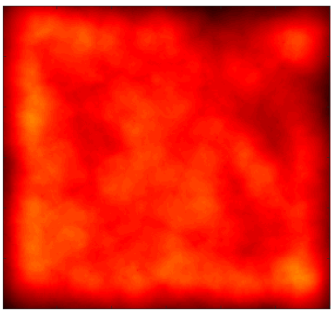
StiCo's and HybaCo's distributions behave similarly with respect to the environments. In the symmetric environments (Figures 6.1b and 6.1c) StiCo and HybaCo are able to match the standard deviation achieved in the empty environment, see Figures 6.7a and 6.7c. In the floor-plan environment the deviation is for both algorithms higher and certain parts of the environment are more frequently visited than others. This is caused by the asymmetric shape of the environment, where certain regions are easier accessible. This can be seen in the heat-map Figure 6.5l, where HybaCo has a higher visit count at the top part, compared to the bottom part of the environment. The floor environment (Figure 6.1d) has two entries at the top and only one at the bottom, StiCo and HybaCo are therefore more likely to enter the top region first. After this region is fully covered the pheromone, of both technique, prevents the remaining robots from entering and pushes them to uncovered space until the complete environment is covered. These effects are further discussed in Section 6.3.



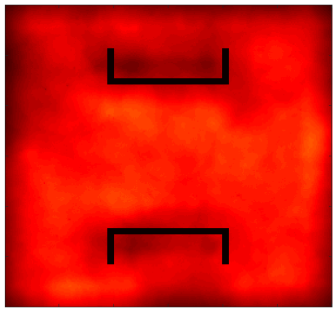
(A) StiCo - Environment without obstacles



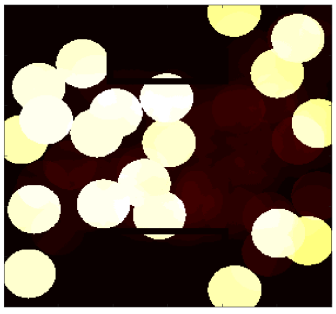
(B) BeePCo - Environment without obstacles



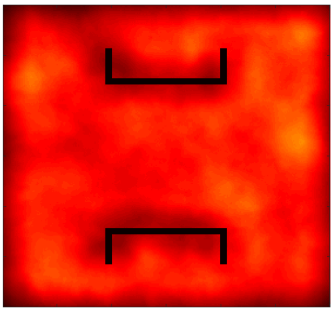
(C) HybaCo - Environment without obstacles



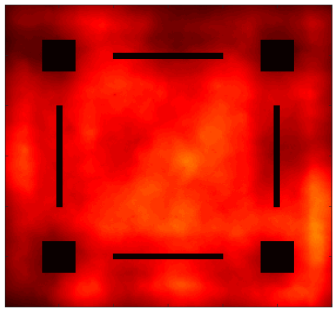
(D) StiCo - U-shape environment



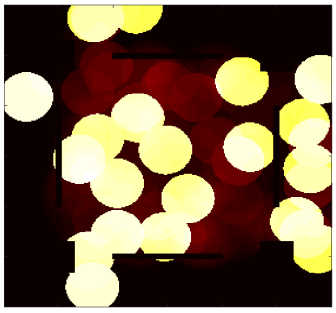
(E) BeePCo - U-shape environment



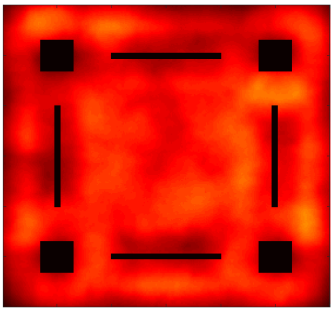
(F) HybaCo - U-shape environment



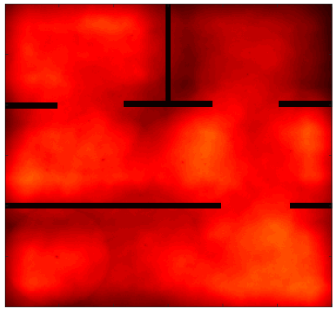
(G) StiCo - 4-axes symmetric environment



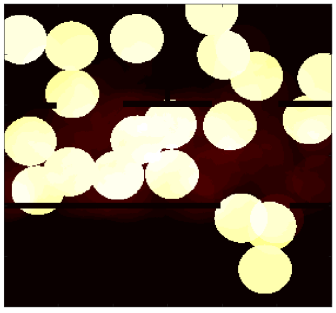
(H) BeePCo - 4-axes symmetric environment



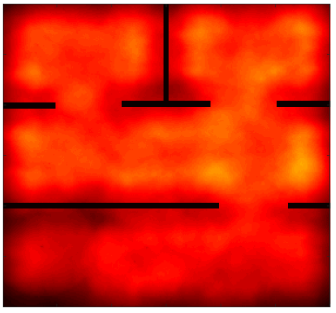
(I) HybaCo - 4-axes symmetric environment



(J) StiCo - floor-plan environment

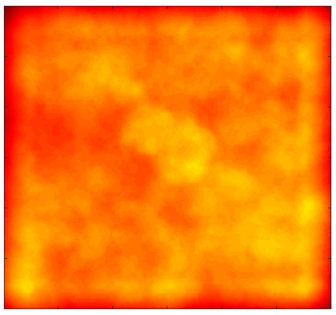


(K) BeePCo - floor-plan environment

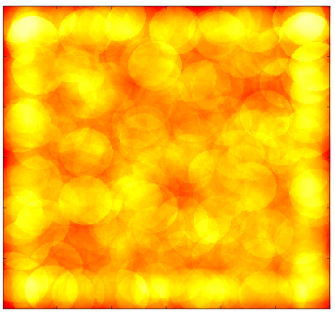


(L) HybaCo - floor-plan environment

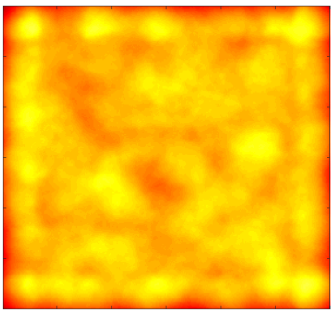
FIGURE 6.4: The distribution over time, in the different arenas, using 20 robots. The coverage is illustrated in a gradient color scheme. A cell is coloured white, if it is covered 100% of the run time and coloured black for a 0% coverage. The intermediate percentages are coloured accordingly to the scale at the top of the figure.



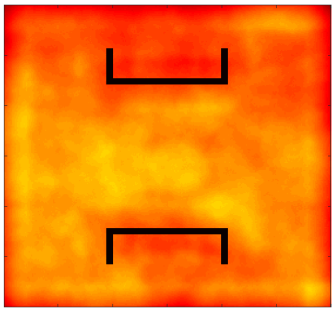
(A) StiCo - Environment without obstacles



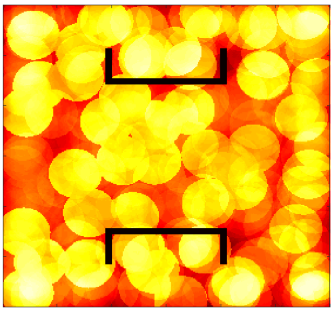
(B) BeePCo - Environment without obstacles



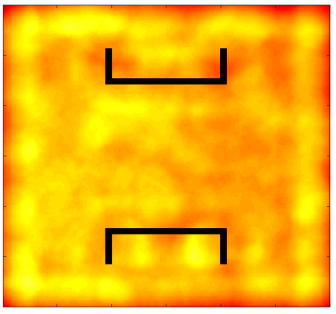
(C) HybaCo - Environment without obstacles



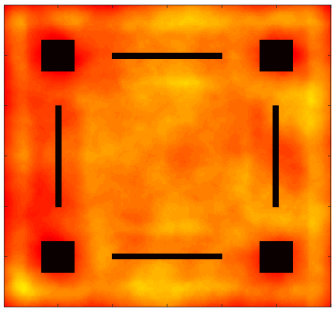
(D) StiCo - U-shape environment



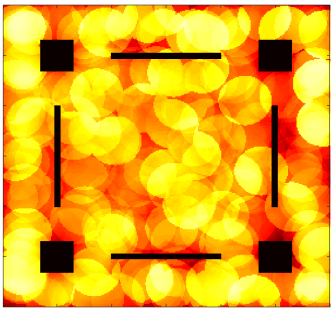
(E) BeePCo - U-shape environment



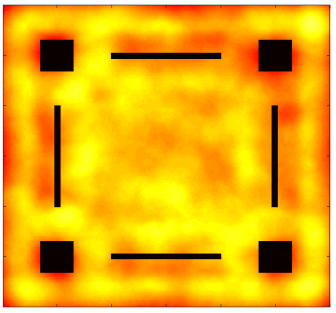
(F) HybaCo - U-shape environment



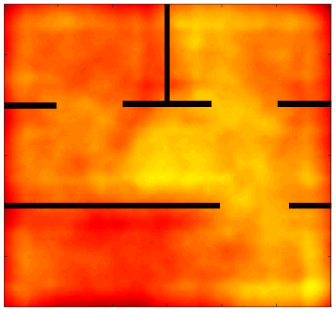
(G) StiCo - 4-axes symmetric environment



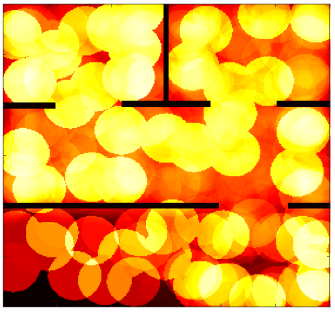
(H) BeePCo - 4-axes symmetric environment



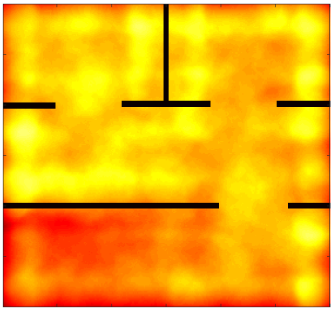
(I) HybaCo - 4-axes symmetric environment



(J) StiCo - floor-plan environment

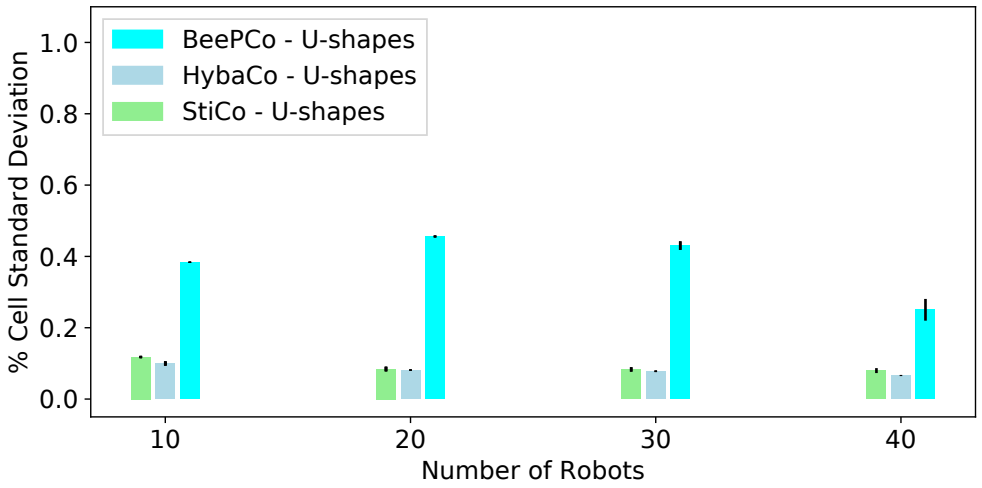


(K) BeePCo - floor-plan environment

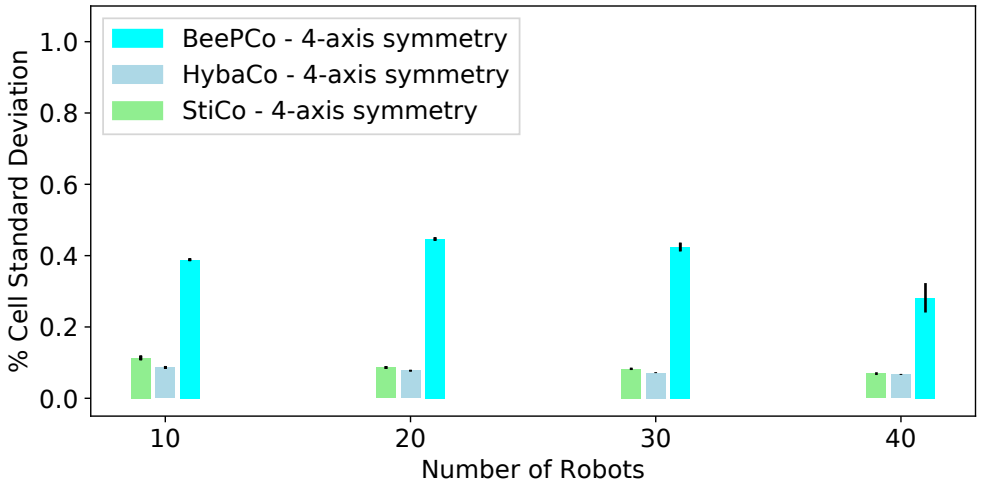


(L) HybaCo - floor-plan environment

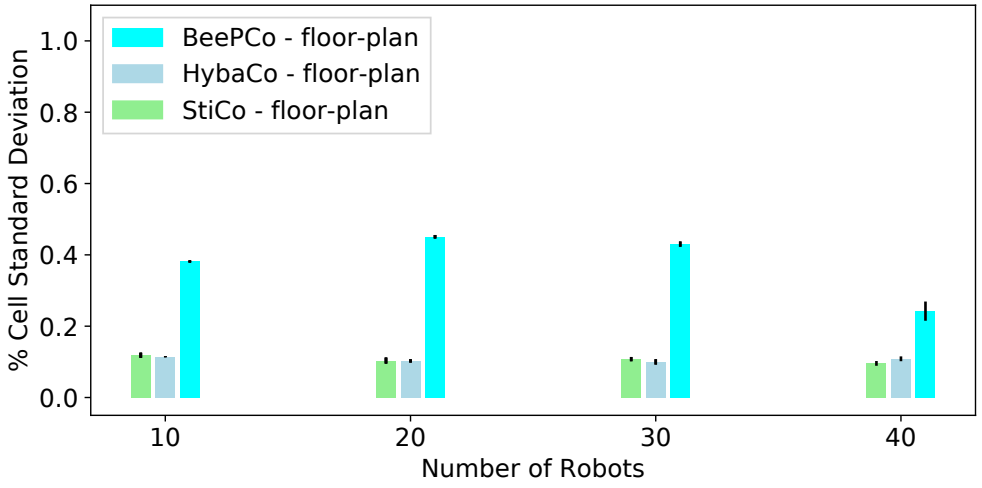
FIGURE 6.5: The distribution over time, in the different arenas, using 40 robots. The coverage is illustrated in a gradient color scheme. A cell is coloured white, if it is covered 100% of the run time and coloured black for a 0% coverage. The intermediate percentages are coloured accordingly to the scale at the top of the figure.



(A) U-shapes Environment

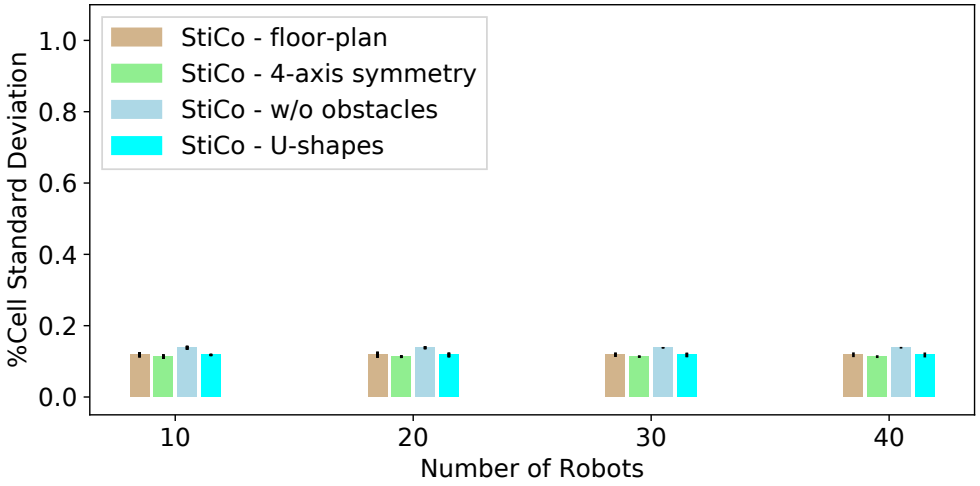


(B) 4-axes symmetry Environment

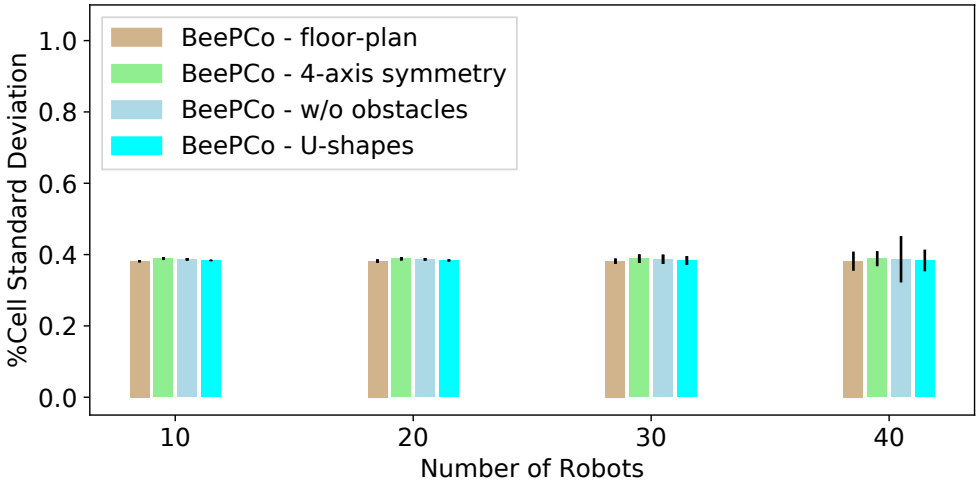


(C) Floor-plan Environment

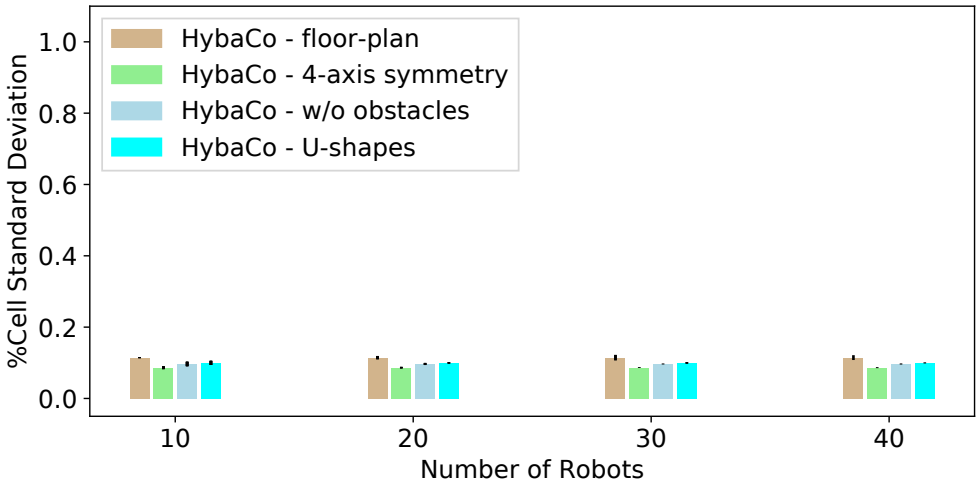
FIGURE 6.6: Compares the standard deviation over all cells, in order to show how evenly the area is covered by BeePCo, StiCo and HybaCo with respect to a specific environment.



(A) StiCo



(B) BeePCo



(C) HybaCo

FIGURE 6.7: Compares standard deviation over all cells and shows how evenly each environment is covered, with respect to a specific pheromone coverage approach.

		Algorithms											
		BeePCo				StiCo				HybaCo			
	#Robots	10	20	30	40	10	20	30	40	10	20	30	40
Environment	w/o obstacles	19.7%	37.9%	54.3%	67.9%	20.2%	35.8%	48.2%	59.2%	20.4%	38.2%	53.6%	66.8%
	4-axis sym.	18.9%	35.5%	50.6%	63.3%	18.1%	32.1%	44.2%	52.6%	18.9%	35.4%	49.5%	62.7%
	U-shapes	18.7%	36.6%	52.6%	66.0%	19.6%	34.4%	46.3%	56.4%	19.8%	37.0%	52.2%	64.2%
	Floor-shape	18.8%	36.4%	51.3%	65.7%	19.0%	34.9%	47.0%	55.8%	19.8%	37.3%	51.9%	63.9%

TABLE 6.3: Maximal Sensor Coverage

6.3 Discussion

As the experiments in Section 6.2 show, the maximal coverage of all three algorithms marginally is affected by the environments. Table 6.3 illustrates the maximal achieved sensor coverage with respect to the number of robots and the environment. It can be seen that all algorithms have the highest drop in the 4-axes symmetric environment. This is due to its high obstacle area, symmetry and its narrow areas at the outer realm of the environment (Figure 6.1b). Table 6.1 illustrates the different environments, with their total area, the combined area of the obstacles and the area coverable by the sensors. Through the high obstacle area and symmetry the likelihood for encountering an obstacle is in every part of the environment very likely. Obstacles can block or restrict the sensors, which leads to a reduction of the sensible area.

Additionally, the narrow pathways are at the outer region of the environment restricting the movement of the robots, which can result in a collision with other robots. These effects are proportionally increased by the number of robots, see Table 6.3. Each additional robot increases the chance of colliding with the environment or an overlap of sensed areas. For instance HybaCo's coverage is reduces by $0.5\% - 1.5\%$ by 10 robots, $0.9\% - 2.8\%$ by 20 robots, $1.4\% - 4.1\%$ by 30 robots and $2.9\% - 4.8\%$ by 40 robots.

This shows that StiCo, BeePCo and HybaCo coverage performances are robust but are reduced by an increasing robot density and obstacle area.

Figure 6.7 and the Table 6.4 show how the convergence time of StiCo, HybaCo and BeePCo are influenced by the environment. The table represents time as a percentage of the total runtime (6 minutes) and shows the duration, required by the algorithms, to converge to a stable coverage state. BeePCo and HybaCo converging times are similarly influenced by the environment and rise with the number of robots. BeePCo's and HybaCo's time increases by $1\% - 3\%$ for the symmetric environments (Figures 6.1b and 6.1c) and by around $1\% - 5\%$ in the floor-plan environment. The slow down can be explained by the fact that obstacles restrict the action space of near by robots and block their initially intended path. In the case of the floor-shape environment walls are positioned close to the initial position of the robots, this means the robots are not able to spread far apart before they encounter the obstacle. In such a dense group formation the movement is additionally blocked by other robots, which are responsible for the further increase in the convergence time.

		Algorithms											
		BeePCo				StiCo				HybaCo			
	#Robots	10	20	30	40	10	20	30	40	10	20	30	40
Environment	w/o obstacles	4%	4%	6%	4%	8%	8%	10%	12%	3%	4%	4%	4%
	4-axis symmetry	3%	4%	6%	7%	8%	8%	13%	18%	3%	4%	5%	6%
	U-shapes	3%	5%	6%	7%	9%	9%	13%	18%	4%	5%	5%	6%
	Floor-shape	4%	6%	7%	9%	14%	19%	27%	33%	4%	6%	8%	8%

TABLE 6.4: Percentage of total run-time required to converge to a stable coverage state.

StiCo suffers from the same effects as BeePCo and HybaCo, but it experiences a higher increase in time, especially for the floor shaped environment (Table 6.4 and Figure 6.7). This can be explained by the problem described in Section 5.1, where robots in StiCo are continuously switching between rotation directions when they are surrounded by pheromone. Robots in this situation are not able to make a valuable movement decision until the pheromone has evaporated. The probability of these situations increases in confined spaces with a high robot density, which explains the results in the floor-shape environment (Table 6.4).

Overall it can be concluded that BeePCo's, HybaCo's and StiCo's convergence times are influenced by obstacles in the environment, and increase proportionally to the number of robots. By applying BeePCo's bee-pheromone technique HybaCo is able to improve the converging time of its underlying technique StiCo.
For StiCo and HybaCo the distribution over time depends on the structure of the environment. As shown in Section 6.2 and summarised in Table 6.5, both algorithms are able to provide a similar cell coverage for all symmetric environments (Figure 6.1b and 6.1c), including the empty environment (Figure 6.1a). In the remaining floor shaped environment the deviation for StiCo is between $2.0\% - 3.6\%$, and for HybaCo between $2.0\% - 3.6\%$, higher than the deviation achieved in the empty environment. This is because certain parts of this environment are easier accessible than others. Since HybaCo and StiCo spread their robots equally in all directions, they will fill the open-parts of the environment first. This continues until the required amount of robots, cover these regions. By avoiding pheromones from covered areas, HybaCo and StiCo will eventually fill in the less accessible regions in the environment. This can be seen in the heat-map of Figure 6.3c, where certain parts of the environment are presented in a lighter colour than others. This means, that these regions were discovered first and are therefore covered for a longer time period. The heat-map (Figure 6.5l) and the maximal coverage (Table 6.3) show that HybaCo and StiCo will eventually cover a similar amount of area in each environment type, which means the deviation will decrease when the run-time increases.

The experiments in Section 6.2 show that BeePCo has two different behaviours, depending on the number of robots (Figure 6.7b). It converges, in a setting of 10, 20 and 30 robots, to a disjointed state where the robots stop moving and are not influencing each

		Algorithms											
		BeePCo				StiCo				HybaCo			
	#Robots	10	20	30	40	10	20	30	40	10	20	30	40
Environment	w/o obstacles	38.7%	46.9%	44.5%	23.5%	12.9%	8.5%	7.3%	7.4%	9.8%	7.7%	7.2%	6.4%
	4-axis symmetry	38.9%	44.7%	42.5%	25.4%	11.4%	8.7%	8.3%	7.0%	8.7%	7.8%	7.2%	6.7%
	U-shapes	38.4%	45.6%	43.1%	25.8%	11.8%	8.5%	8.4%	8.1%	10.0%	8.1%	7.8%	6.6%
	Floor-shape	38.2%	45.1%	43.0%	24.8%	11.9%	10.5%	10.9%	9.7%	11.5%	10.3%	10.1%	10.9%

TABLE 6.5: Standard Deviation

other. The heat-map Figure 6.4k shows that BeePCo also covers the more accessible spaces first, since the majority of robots are positioned in the top part of the environment, instead of the more hidden part at the bottom of the environment (Figure 6.1d). Table 6.5 illustrates that BeePCo's deviation for the coverage between the cells, in environments with obstacle is slightly lower, compared to the empty environment. This is because the wall avoidance introduces additional movement, which spreads the robots further across the environment.

In the setting with 40 robots all environments are densely packed and BeePCo is not able to disjoint all robots, which introduces a continuous motion. In this setting the robots are able to uniformly cover the environment. The deviation between the cells, for the environments with obstacles is marginally higher than without. This has two reasons: 1) BeePCo fills open spaces first and 2) it is not able to communicate through walls which reduces the pheromone propagation and avoidance.

Overall, we can say that all algorithms are able to maintain their even coverage in open and symmetric environments. In asymmetric environments BeePCo, HybaCo and StiCo cover the easier accessible parts of the environment first. If the number of robots is sufficient for the size of the environment, all algorithms will eventually cover the full environment and the deviation for the coverage between the cells will decrease when the run-time increases.

6.4 Conclusions

This chapter extends the initial experiments for BeePCo and HybaCo in Chapter 4 and 5 and analyses their performance in more complex environments, in order to address research question **RQ3**:

Research Question RQ3: Given algorithms derived from RQ1 and RQ2, to what extent does the environment shape and size affect their performance?

Section 6.1 introduces a selected set of test environments with different shape and symmetry properties. Section 6.2 shows that the pheromone based approaches are able to maintain their sensor coverage, and how their distribution and convergence times depends on structure of the environment. Additionally, the experiments show that HybaCo is able to maintain the maximal coverage of BeePCo while matching the distribution over time of StiCo.

Chapter 7

Distance-based Multi-robot Coordination on Pocket Drones

The previous proposed algorithms BeePCo and HybaCo (Chapters 4 and 5) focused on ground-based robots and are done in simulation, since certain aspects for instance artificial pheromones (HybaCo) are currently difficult to implement in a real life scenario. Most pheromone based implantations use visual pheromones by employing fluorescent foil [66, 83] or LCD-screens (Liquid Crystal Display) [7], to overcome this issue. These implementations demonstrate the efficiency and simplicity of these algorithms, but are not applicable in a real-world environment. Implementations suitable for real-life scenarios apply gas, e.g. ethanol (C2H5OH) as a pheromones substitute [44], this limits the duration, of the algorithm execution, to amount of gas a robot can carry. Technology able to produce pheromones from resources in the environment are still in development and will extend the variety of use cases for algorithms like HybaCo.

Wireless communication based coordination algorithms e.g. BeePCo are already applicable with modern technology, but still are facing challenges in the context of swarm robotics. One of them being the relative position estimation of other robots in the group. In the context of BeePCo, robots require the relative location of their neighbouring partners to cover the environment efficiently. This localisation can be difficult, especially in an indoor environment, where position systems e.g. GPS, are not applicable. Additionally, in order to provide a bigger swarm size, most swarm robots platforms are kept simplistic and low in price. Most of these robots have a limited payload and computing power, which doesn't allow for heavy sensors or highly computational algorithms.

The following chapter proposes a system to tackle these issues on an aerial platform, more specifically pocket drones (quad-copter). Quad-copters have the advantage to scan the environment in a top-down fashion and not being affected by obstacle on the ground, but their payload and computations capabilities are limited. The proposed system uses radio-wave distance calculation in combination with neural networks and particle filters to provide relative positions estimation. Additionally, it employs a second neural network

for low-level navigation and collision avoidance. The aim of this system is to provide position estimation, navigation and collision avoidance to high-level coverage techniques, such as BeePCo, to transition them from a ground to an aerial coverage approach. The following Section 7.1 will detail the employed techniques and provides an overview over the structure of this chapter.

7.1 Introduction

Coordination is a challenging issue in multi-robot systems and the more global knowledge a robot has the easier the problem becomes [38]. Such knowledge includes the global position and velocity of a robot and its coordination partners. Most of the time this information is obtained by employing a global map in combination with vision or laser based localisation [39]. This information is then passed on to a centralised system, or to each robot individually, in order to execute the best movement action possible.

When working with smaller robots, for instance pocket drones, these techniques are usually infeasible since most localisation techniques like "Vision-based localisation and mapping" are computationally too expensive and the robots might not be able to carry the required equipment. The MAVs (Micro Aerial Vehicle) used in this research have a payload of around $20g$, therefore, we are aiming for a light weight hardware solution, which can provide us with enough information to perform multi robot coordination, but doesn't require extensive post-processing such as laser scan matching or image processing. Some work already going on in a similar direction, is for example the system proposed in [9] consists of speakers and an array of microphones. By employing the obtained distance and bearing information, and by applying particle filters, the UAVs (Unmanned Aerial Vehicle) are able to detect and track neighbouring robots. Another system is proposed in [84] where ground robots are equipped with infra-red emitters and receivers in a spherical layout; both systems only perform on short distances and need additional processing power to calculate distances and filter external audio/light noises. The system proposed in this chapter is based on ultra wide band (UWB) distance calculations (see Section 2.5). These sensors are sending radio messages back and forth between two devices. The distance is then calculated by multiplying the duration, by the speed of travel. UWB systems are mostly used for global localisation by placing multiple UWB-modules in fixed known locations, a mobile device can be tracked by triangulation. Due to the small footprint and payload of our MAVs, we are only able to carry one module, triangulation is therefore not an option and we have to rely on sequences of distance measurements to provide a sufficient movement strategy. Programming and designing movement policies, e.g. based on state machines, can be a difficult and time consuming job because all edge cases have to be covered and extensively tested. The difficulty is also intensified if the input data is only a partial representation of the environment. In order to tackle this problem we are applying a recurrent neural network (Section 2.2.2) in combination with a particle filter (Section 2.4), to extract features from a sequence of

states to make the world more observable and therefore the application of a movement policy possible.

In the last couple of years recurrent neural networks have shown great performance in tasks which rely on understanding state sequences, for example: natural language processing or playing 3D games like Doom ([62],[37],[90]). Most of those models are applied on a computer generated environment and do not suffer from real-world sensor noise. In our case we are trying to obtain a movement policy which can operate on a robotic system. The system has to be able to control the movement of the drone towards a desired target position and avoid all approaching drones on the way. Each drone operates individually and is only provided with a distance to a target position, distances to approaching drones and velocity measurements. Our model consists of a recurrent neural network for sequential feature extraction, a particle filter to ensure stability in state predictions and a Deep Q-Learning Network (DQN) (see preliminaries Section 2.3.2) providing the movement policy.

The remainder of this chapter is organised as follows. Sections 7.2 and 7.3 describe the employed hardware in form of the quad-copter and UWB-module. Section 7.4 describes the structure and techniques of the proposed model, providing position estimation and motion control. Section 7.5 describes the training of the neural networks the model contains. Section 7.6 and 7.8 illustrate the simulation and real-world experiments the model is evaluated on. The experiments results are further discussed in Section 7.9 and the chapter is concluded in Section 7.10.

7.2 Drone

This section shows the employed quadrotor and its integration in the system. Section 7.2.1 shows the hardware specification and Section 7.2.2 illustrates the drones additional velocity control implementation and its interface to the proposed model.

7.2.1 Specifications

The employed platform is called crazyflie, produced by the company BitCraze (Figure 7.1). It is a 27 g micro drone, based on a STM32 micro processor (192Mhz), with a maximal payload of 15 g. The drone provides with SPI (Serial Peripheral Interface) and I^2C (Inter-Integrated Circuit) two hardware interfaces to connect external devices to the drone. The flight controller firmware is an open source software written in C allowing complete customisation. The flight controller utilises a cascades PID-controller (see Section 2.6.5), for a stabilised flight. The controller is able to control angular velocity and attitude in roll, pitch and yaw-axes of the drone.

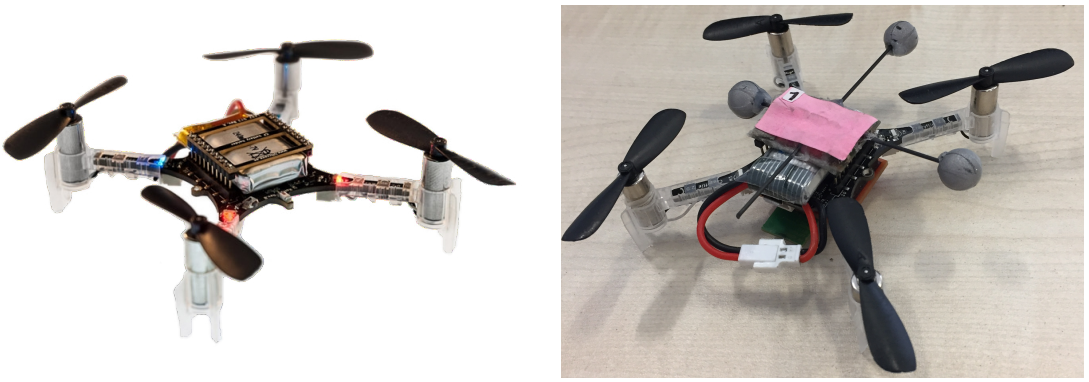


FIGURE 7.1: Employed drone named crazyflie. Right image shows a crazyfie with motion tracking markers, necessary to track position and orientation of the drone.

7.2.2 Velocity estimation and control

The model proposed in Section 7.4 requires linear velocity control along the x-, y- and z-axis, to navigate and hold positions. In order to allow the velocity control, we supply the drone with additional velocity information provided by a motion tracking system. The motion tracking system uses an array of infra-red (IR) cameras to estimate the position and velocity of the drones. In order to calculate the position, the tracked drone have to carry reflective markers (Figure 7.1). These markers bounce emitted IR-light to the cameras, enabling the detection and position estimation. By dividing the difference between two consecutive positions, by the time difference Δt we obtain the velocities for all axes. The motion tracking system limits the application of the proposed model its capture volume. The capture volume describes the volume the system is able to track the drone.

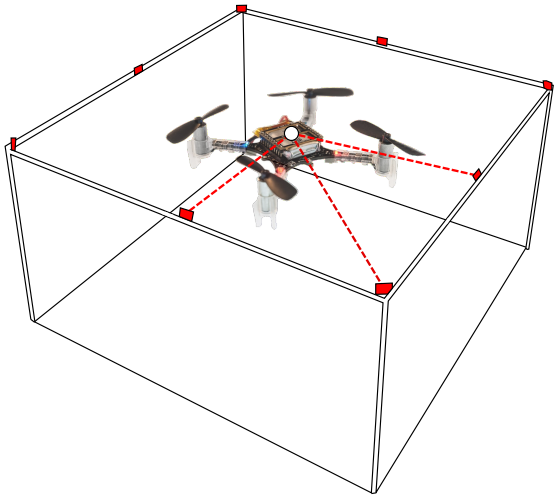


FIGURE 7.2: Simplified motion tracking rig. Cameras (red boxed) are placed around the capture volume. Each camera emits IR-light, which bounces from the reflective markers of the object into the image-sensors of the cameras. If the system find at least two images showing the same marker, it is able to triangulate the marker position in 3D. The system requires at least three markers to estimate the objects orientation.

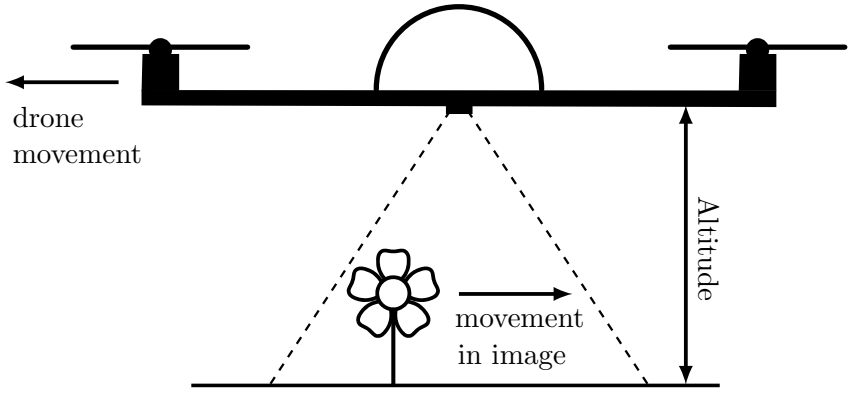


FIGURE 7.3: Optical flow sensor: The system is able to measure the linear velocity of the drone, by tracking the movement of features (flower) in two consecutive images. By knowing the field of field (dashed lines) of the camera and the altitude of the drone, the system is able to estimate the velocity.

In order to overcome this limitation in the future, the motion tracking system will be substituted by a optical flow system. An optical flow sensor uses a small camera at the bottom of the drone to measure the velocity in the x- and y-axis and rotational velocity around the yaw-axis. Additionally, it employs a distance sensor (for instance sonar) to detect the height. The optical flow sensor tracks features in two consecutive images and measures the moved distance (in pixel) between the images. By knowing the height, returned from the distance sensor and the intrinsic parameters of the camera, it is able to derive the velocity in x-, y- and yaw. The climb rate (velocity in z-axis), is measured by the distance sensor (Figure 7.3).

These measurements (from motion tracking or optical flow) are utilised by a PID-controller to control the velocities of the drones. The controller is placed on top of the native controllers and passes down set-points, based on the difference between target- and current velocity. The proposed model resides on top of the velocity controller, deciding on movement actions and passing down velocity commands, which are then maintained in a closed loop fashion, by the velocity controller (Figure 7.4).

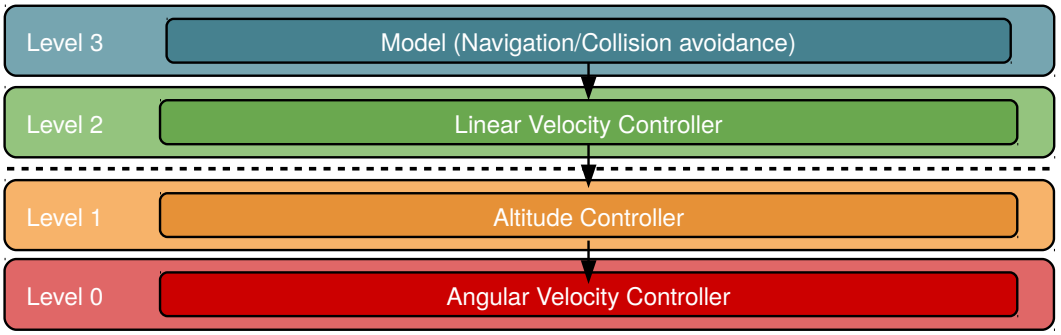


FIGURE 7.4: Hierarchy of drone controllers. The dashed line marks the border between the native drone controllers at the bottom and the newly added linear velocity controller and the proposed model at the top.

7.3 Ultra Wide-Band Distance Calculation

The system proposed in this chapter provides navigation and collision avoidance for high level swarm coordination principles on pocket drones. As described in Section 7.1 is the position estimation between the drones, an intermediate challenge we are tackling.

We are applying Ultra Wide-Band UWB distance calculation in combination with the model described in Section 7.4 to accomplish this task. UWB distance calculation uses radio-wave based communication to determine distances between two devices. By exchanging multiple messages back and forth, the system is able to determine the travel time of a specific message. The distance is obtained by multiplying the duration of a single message exchange, by the speed of travel (light speed).

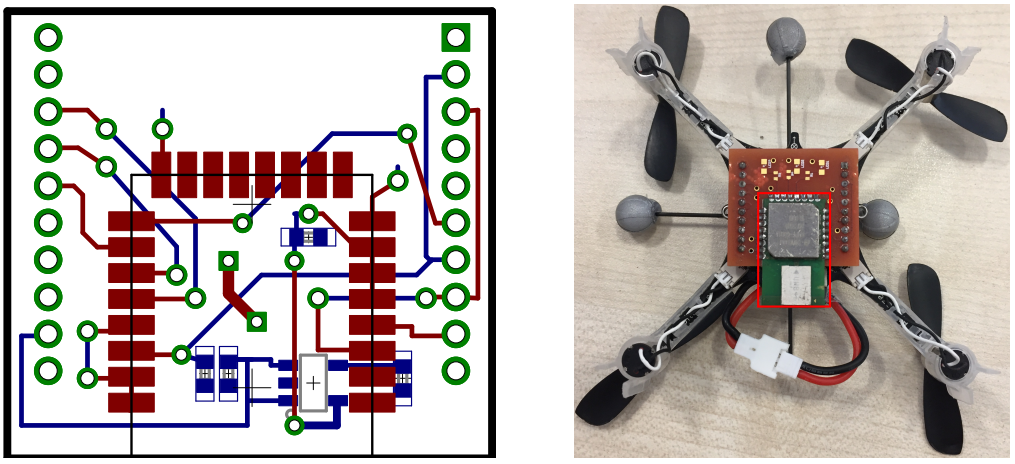
The employed message protocol requires four messages for the distance calculation and is usually known as two-way ranging. For more detail about UWB distance calculation and two-way ranging, see Section 2.5.

UWB distance calculation has different properties, which makes it attractive in the field of pocket drones. Its hardware is extremely light and the communication can penetrate through walls and obstacles. Since it uses messages for the distance calculation, it provides an option to exchange additional information between the drones. It is able to integrate this information in the normal message exchange. This makes it attractive for message based coverage approaches such as BeePCo (Section 4).

The remainder of this section describes the employed hardware in Section 7.3.1, its noise model in Section 7.3.2 and the handling of cross-talking in Section 7.3.3.

7.3.1 Hardware

The UWB distance calculation is accomplished by the DWM1000 module produced by the company DecaWave. The module bases on Decawave's DW1000 UWB transceiver and integrated additional antenna, power management and clock circuitry required for the distance calculation. The module stores the time stamps of send and received messages in memory, this functionally is necessary for an accurate estimate of a messages travel duration. The functionalities of sending, receiving and reading message time stamps are accessible via the hardware interface SPI (Serial Peripheral Interface). In order to use this interface, we designed a circuit board which connects the SPI-buses of drone and UWB-module and regulates battery voltage to the required 3.3V. A schematic of the board can be seen in Figure 7.5a.



(A) (B)

FIGURE 7.5: A: Shows the schematic designed to connect the UWB-module to the drone. B: Shows the module (DWM1000), surrounded by the red rectangle, connected to the crazyflie.

7.3.2 UWB Noise Model

In order to simulate the UWB-module's distance estimation and have an overview over its performance. We collected the module distance estimation at different fixed distances and did 40 flights with random movement and compared the estimation with the data gathered from the motion capture system. The error between actual and estimated distance data is modelled as a normal distribution with a mean of $\mu = 0.0085569$ and standard deviation of $\sigma^2 = 0.061295$.

7.3.3 Communication Protocol

The distance between two drones is calculated by applying a two-way ranging protocol, which requires four message exchanges, for more detail see Section 2.5. A drone has to initialise a two-way ranging communication with all neighbouring drones to receive their distances. Depending on the swarm size, this can result in a lot of parallel communications, causing interference in the message exchange. This effect is reduced by reporting the calculated distance back to the communication partner and utilisation of a token ring protocol.

The distance is usually calculated by the device initialising the conversation, by reporting the distance back to the communication partner, this partner doesn't require an extra two-way communication. The token-ring is a common principle in wireless communication to inhibit parallel communications, by allowing only the devices holding a token to initialise conversations. After the device gathered the required information it passes the token to the next device etc. The employed token-ring protocol can be simplified in the following steps: All drones have a unique id and the drone with the lowest id is initialised with a token. This drone initialises the two-way ranging with the next higher id (See Algorithm 11).

Algorithm 11: initialise ranging for id i

```
1: set the current communication parter to i + 1
2: initial two-way ranging with communication partner
```

All continued steps are trigger when a new message is received. If the received message is the last message of the ranging protocol, it calculates the distance, sends it to the last communication partner and initialises a ranging communication with the next possible drone. If it received the distance to all drones it passes the token on and the process continues. After the token reaches the second to last highest id, the distance between all drones are updated, and the token is passed on to the first drone etc. The process, triggered by a received message, is illustrated in the pseudo-code of Algorithm 12.

Algorithm 12: message received by id i

```
1: if message contains token then
2:    save token
3:    initialise ranging (Algorithm 11)
4: else
5:    if message is last message of two-way ranging communication then
6:       calculate distance
7:       report distance to last communication partner
8:       if last communication partner has highest possible id then
9:          forward token to next possible partner
10:      else
11:         initial two-way ranging with next communication partner
12:      end if
13:   else
14:      continue two-way ranging
15:   end if
16: end if
```

In order to increase the update rate, the implemented system allows handling of multiple tokens. but in the scenarios described in Section 7.8 the use of one token is sufficient and inhibits the possibility of interference.

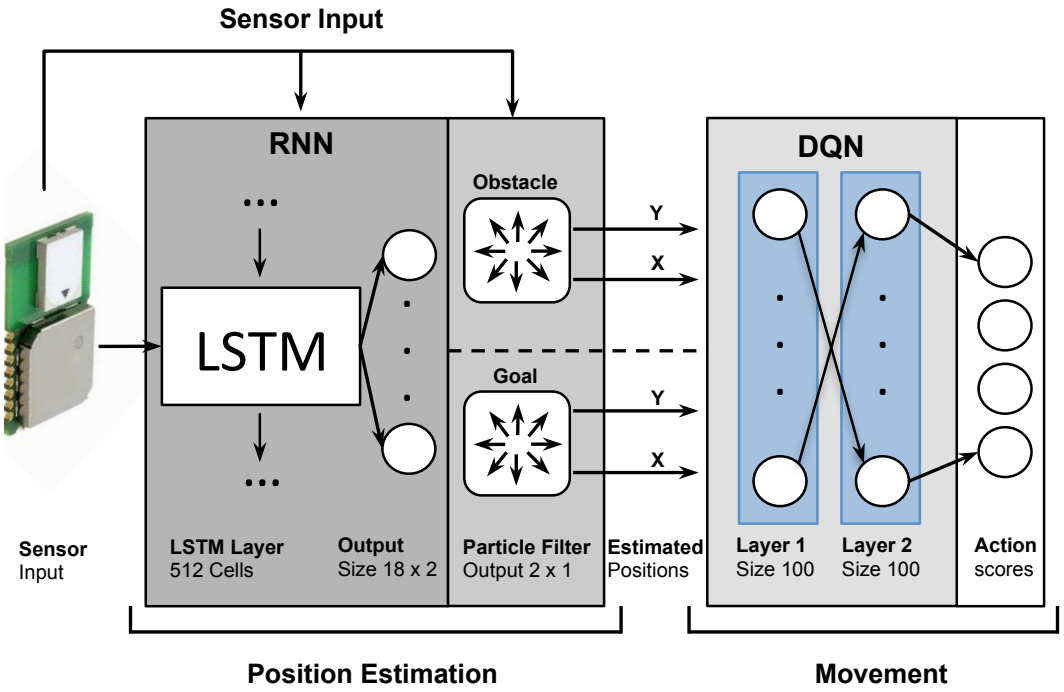


FIGURE 7.6: Architecture and data flow of the employed model.

7.4 Model

The model is designed to provide navigation and collision avoidance on pocket drones for autonomous flight, high level coverage or coordination approaches. As mentioned in Sections 7.2 and 7.3, the applied hardware is able provide distances between drones and their velocities, but is not able to provide the relative position between the drones, natively.

Therefore, the extraction of the relative position based on the provided data, is one of the proposed models aims. Since it is mathematically not possible to determine the position based on a single state (distance and velocity at time t). It has to be able to analyse a sequence of states, to extract an accurate estimation.

A continues position estimation and tracking over the complete group of drones is computational expensive, especially with an increasing group size. This means, the system should only track drones, important for the collision avoidance and allow a quick estimation of new drones if they become important.

A valuable technique to estimate the position based on movement and distance measurements is a particle filter, described in Section 2.4. It initialises particles based on a random distribution. Each particle represents a possible position and orientation of a drone. The filter moves the particle according to the odometry (measured velocity) and weighs their probability based on the distance measurements. By resampling more likely particles, in every time step, the distribution converges towards the actual position (Section 2.4). The quality of the estimation and the speed of the convergence, depends partially on the initialisation of the samples. The closer they are spawned to the actual solution, the more likely they converge to a correct solution. Additionally increases the

probability of finding a valuable solution, proportional to the sample count. A higher sample size causes consequently, computational overhead and the increases the likelihood of a convergences to multiple possible solution.

In order to compensate those issues, we employ a Recurrent Neural Network (RNN) to estimate the required positions and use a particle filter to stabilise the estimations (Figure 7.6). This reduces the necessary particles to a minimum and increases the speed of convergence, shown in Sections 7.6 and 7.8. The combined system is used to estimate the position of drones and navigation goals. It requires a separate particle filter for each position it has to track, but can utilised the same RNN for the estimation. During this chapter we refer the navigation destination of the system as goal and the drone the system is trying to avoid as obstacle (see Figure 7.6).

The second part of the proposed system uses the estimated goal and drone positions, to navigate and avoid collisions with other drones. It employs a Deep-Q Network (DQN) to calculated the optimal action based on its current state (Figure 7.6). For more information about DQNs see Section 2.3.2.

The remainder of this section is structured as follows: Section 7.4.1 describes terminology and assumptions employed for the remainder of this section. Section 7.4.2 describes the structure, input and output states of the employed RNN, Section 7.4.3 shows the specifics of the particle filter and its combination with the RNN. The section is concluded with a description of structure, input and output states of the employed DQN in Section 7.4.4.

7.4.1 Terminology and Assumptions

This section describes the terminology and assumptions, employed in the following sections detailing the proposed model. One of the main assumption is that all drones operate at the same altitude. This is caused by the fact that drones in an indoor environment with an average sealing height are not able to fly over one another. Since this causes a so called prop-wash effect, where the drone on the top, blows air on the drone at bottom. This additional air flow complicates the dynamics and therefore the stabilisation of the drone, causing it to oscillate and eventually crash.

The second reason for using the same altitude, is that it reduces the navigation and position estimation to 2D-problems, which also simplifies the transfer of ground robot coordination principles to this drone system.

All following terminologies are summarised in Table 7.1. A subtask of our system, is to navigate a controlled drone, to its destination and avoid other drones on the way. During this chapter, we will refer the destinations as goals and since the system has to avoid the others drones, we refer them as obstacles.

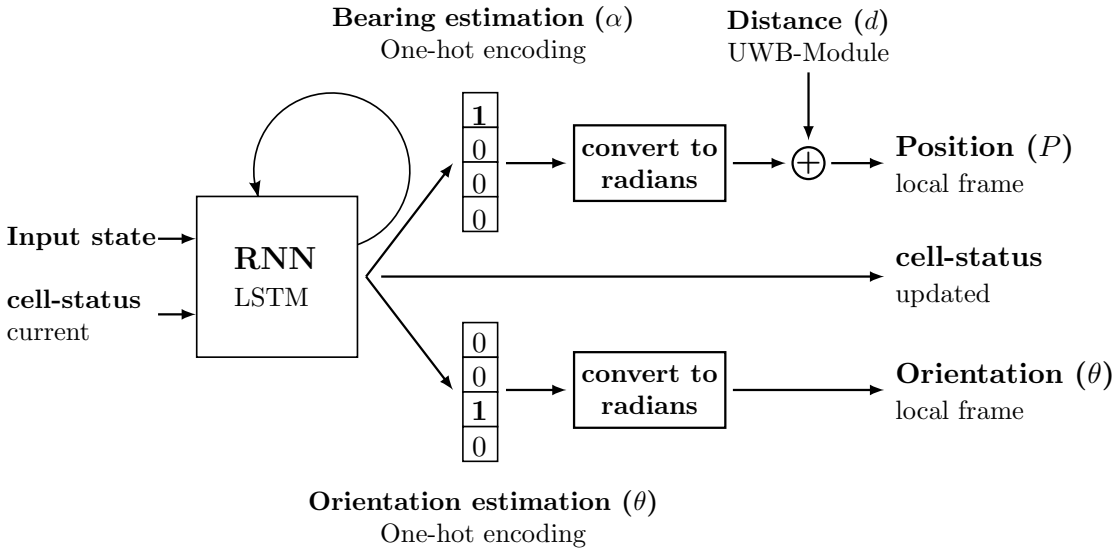
The second subtask of the system is the positions estimation of these obstacles, by using a sequence of sensor inputs. Additionally, we use the same system and input state structure (see Section 7.4.2) to estimate the goal position. This has the advantage, that can we use other drones or robots (with a UWB-module) as navigation destination,

Term	**Description**	**Default Parameter**
Goal	Destination (position) a drone is trying to reach.	–
Obstacles	Drones the observed agent is trying to avoid.	–
Object Position	Describes the position of either a goal or an obstacle.	P
Orientation	Describes the yaw-rotation of an obstacle in the frame of the observed drone	θ
Distance	Distance to an object estimated by the UWB-module	d
Bearing	Angle describing the direction pointing to an obstacle or goal.	α

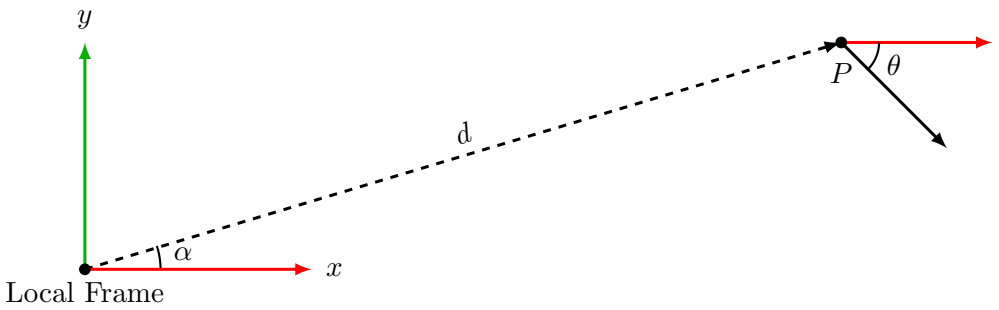
TABLE 7.1: Terminology Chapter 7

which is an interesting feature in exploration or search and rescue scenarios. Therefore, when we talk about the position estimation of an object, the object can either be a goal or an obstacle. The estimated position is formalised with by the variable P.

As mentioned before, the position estimation can be reduced a 2D-problem, and the occurring geometry can be described as follows. When we describe the estimation of a goal or obstacle position, we always take the view of the drone performing the estimation. Therefore, all positions and orientations are in local frame of this particular drone. Besides estimating the positions of goal and obstacle, the particle-filter requires additionally the orientation of the obstacle (see Section 7.4.3). Since the drones operate on a 2D-plane, the particle filter requires exclusively the yaw-rotation (rotation around the up pointing z-axis) of the tracked object. As a result the mentioning of the orientation (θ), during this chapter, will always refer to the yaw-rotation of an obstacle, in the local frame of the drone we observe (Figure 7.7b). Since the system acquires a distance from the UWB-module, the 2D-position estimation can be reduced to a one dimensional problem, when using a polar coordinate system. Polar coordinates describe a 2D-point by its distance (d) from the origin and the angle (α) describing the direction to the point. This is shown in Figure 7.7b. During this chapter we will refer the direction angle as bearing. The mentioned terminology is summarised in Table 7.1.



(A) **RNNs data flow:** The figure shows the single steps required, to obtain estimated position and orientation of an object, based on a given input state.



(B) **Geometric interpretation:** $\boldsymbol{\alpha}$: Bearing estimation returned from the RNN. $\boldsymbol{d}$: Distance measure by the UWB-module. $\boldsymbol{P}$: Estimated position of the object, calculated by combing bearing α with the distance $\boldsymbol{d}$. $\boldsymbol{\theta}$: RNN's orientation estimation.

FIGURE 7.7: Estimation of the objects position and orientation, provided by the RNN.

7.4.2 Recurrent Network Model

Recurrent neural networks (RNN) have the ability to "remember" previous input states, allowing them to extract informations from a sequence of states. Instead of simply propagating the data from the input- to the output-layer of the network, RNNs contain recurrent links propagation the data back to nodes of the same or previous layers. These recurrent connections can therefore be seen as memory. For more detail about RNNs, see Section 2.2.2. The employed RNN uses Long-Short-Term-Memory (LSTM) cells with the ability to "remember" states over a longer period of time. The cells are able to make decisions, when to store, read or forget these informations, by utilising gates know as input gate, output gate and forget gate. The behaviour of these gates depends on weights, which are learned during the training process. For more detail about LSTMs, see Section 2.2.4.

The aim of the RNN, is to estimate the bearing to a certain object (goal or drone) and the objects relative orientation (yaw-rotation θ). Using the estimated bearing α (in radians) and the distance d received from the UWB-module, we can calculate the relative x and y position of the object, assuming the drones operate at the same altitude (see assumptions, in Section 7.4.1):

$$x = d \cdot \cos(\alpha) \tag{7.1}$$

$$y = d \cdot \sin(\alpha) \tag{7.2}$$

The estimated bearing α can either be pointing towards a goal or to another drone. The ability to use the same RNN to estimate the goal and obstacle positions is enabled by the way its implemented. Instead of storing the cell status in the network, it is stored externally and given as an additional input every time, an estimation is required (see Figure 7.7a). The network returns the estimation together with the updated cell-status. This allows to track a unlimited amount of objects (drones/goals) as long as the system stores a different cell-status for each object.

The estimated orientation θ, represents the translation between the own frame and the frame of the object. This information is required by the particle filter, in order to translate the objects velocity into the local frame of the current drone. The RNN requires, as input, the delta time dt since the last update, the distance d to an object, its own velocity (v_x, v_y) and the velocity of the object (v_{ox}, v_{oy}). Therefore, is the input state defined as $(dt, d, v_x, v_y, v_{ox}, v_{oy})$. where:

$\boldsymbol{dt}$:	Time passed since last update	$\boldsymbol{d}$:	Distance to object
$\boldsymbol{v_x}$:	Local velocity in x-direction	$\boldsymbol{v_y}$:	Local velocity in y-direction
$\boldsymbol{v_{ox}}$:	Objects velocity in x-direction, in local frame of obstacle	$\boldsymbol{v_{oy}}$:	Objects velocity in y-direction, in local frame of object

Preliminary tests showed that the inclusion of the objects velocity information, result in a high gain, in the networks performance. Since the ranging requires a message transfer, we are able to include this information into the ranging message.

The RNN illustrated in Figure 7.6 consists of one layer of 512 LSTM-cells and discretises the output angles (orientation and bearing) in steps of 20 degrees, resulting in two vectors with the size of 1×18. Each output is an one hot encoding vector, marking the predicted angle-class with one and all others with zero (see Figure 7.7a).

The loss-function used to evaluate the network performance during training is the combined cross-entropy of the predicted vectors and an one hot encoded vector of the actual bearing/orientation angle.

7.4.3 Particle Filter

In the particle filter, each particle represents a possible position and orientation of an object (goal or drone). By moving each particle according to the movement measured (u), provided by the odometry (Section 7.2.2) and by calculating their probability in respect to the sensor data (z), certain particle become more likely than others and the estimation will converge towards a probable solution. For more details about particle filters see 2.4. All parameters required to describe the employed particle filter are summarised in the following Table 7.2.

$\boldsymbol{v_x}$:	Local velocity in x-direction	$\boldsymbol{v_y}$:	Local velocity in y-direction
$\boldsymbol{v_{ox}}$:	Objects velocity in x-direction, in local frame of objects (goal/obstacle)	$\boldsymbol{v_{oy}}$:	Objects velocity in y-direction, in local frame of object (goal/obstacle)
$\boldsymbol{P}$ or $\boldsymbol{x}$:	Representation of particle pose with x-position, y-position and yaw-rotation: $x = P = (P_x\ P_y\ P_\theta)$	$\boldsymbol{u}$:	Movement update, combination of local velocity and velocity of object $u = (v_x,\ v_y,\ v_{ox},\ v_{oy})$
$\boldsymbol{dt}$:	Time passed since last update	$\boldsymbol{z}$:	Distance measurement (UWB-module)
$\boldsymbol{m}$:	Number of samples	$\boldsymbol{s}$:	$s \in [0,\ 1]$ determines the percentage of particles spawned based on the RNN estimation.
$\boldsymbol{l_r}$:	Linear x- and y-range samples are spawned around the RNN estimation.	$\boldsymbol{a_r}$:	Angular range spawned samples are orientated around the RNN estimation.

Table 7.2: Particle filter Parameters

In the proposed model, the main purpose of the particle filter is to stabilise and disregard unrealistic estimations, made by the RNN. The RNN is able to provide a sufficient estimation of the object it tracks, but has no validation if two consecutive estimations are physically possible. Since the particle filter takes the motion into account, it is able to filter outliers and maintain previous probable estimations.

As mentioned in Section 2.4, the filter updates the particles based on the control u_t for every time step t. In the case of our model, is u_t the combination of the x and y velocities $(v_x,\ v_y)$ of the observed drone and the velocities $(v_{ox},\ v_{oy})$ of the drone/goal represented by the particles. Since the particles are in the local frame of the observed drone, we apply the inverse of the own velocity and translating the velocity of the object into the own local frame, to derive the new particle pose P'. The movement update applied to the particle pose P is summarised in the following Algorithm 13:

Algorithm 13: Motion update

apply_motion(u_t, P, dt)
  initialise $P' = P$
  extract $v_x,\ v_y,\ v_{ox},\ v_{oy}$ from u_t
  // apply inverted velocity
  $P'_x = P_x - v_x \cdot dt$
  $P'_y = P_y - v_y \cdot dt$
  // rotate object velocity into local frame and apply to particle position
  $P'_x = P'_x + (v_{ox} \cdot cos(P_\theta) - v_{oy} \cdot sin(P_\theta)) \cdot dt$
  $P'_y = P'_y + (v_{ox} \cdot sin(P_\theta) + v_{oy} \cdot cos(P_\theta)) \cdot dt$
  return P'

The weighting is done based on the noise model of the DecaWave UWB-module, shown in Section 7.3. The filter itself operates similar to a standard particle filter, but besides resampling probable particles, it additional draws a small percentage (s), based on the estimation made by the RNN. In order to avoid redundancy in the recently spawned samples, we introduce a slight randomness to the estimated poses. The generating process of these kind of particles is described in Algorithm 14.

Algorithm 14: generate RNN sample

generate_RNN_sample()
  generate P based on RNN estimation
  $P_x = P_x + \text{random}\left(-\frac{l_r}{2},\ \frac{l_r}{2}\right)$
  $P_y = P_y + \text{random}\left(-\frac{l_r}{2},\ \frac{l_r}{2}\right)$
  $P_\theta = P_\theta + \text{random}\left(-\frac{a_r}{2},\ \frac{a_r}{2}\right)$
  return P

The complete pseudo code, of the particle filter implementation, is illustrated in Algorithm 15, the major alteration can be seen in lines 9 - 13. By employing the additional knowledge from the RNN, we are able to reduce the particle count to a minimum, while providing a stable estimation and fast converging speed (see Section 7.6 and 7.8).

7.4.4 Deep Q-Network

Reinforcement learning is designed to learn a policy for choosing the "best" action based on the current state. This is realised by incentivising a learning agent with rewards, by allowing the agent to interact with an environment and administer a positive reward for reaching desired states and giving a negative reward for reaching undesired states. The reward is determined by a designed reward function. The Deep-Q-network DQN, is a neural network to approximate a function determining the long term reward, of taken a specific action in a specific state. This function is usually known as Q-function, where the quality of a state action pair, is determined by a so called Q-value, see Section 2.3.2.

Algorithm 15: Alternated Particle Filter (lines 9 - 13)

```
1:  particle_filter(X_{t-1}, u_t, z_t)
2:     X̄_t = X_t = ∅
3:     for m = 1 to M do
4:        x_t^[m] = apply_motion(u_t, x_{t-1}^[m])
5:        w_t^[m] = weight_sample(z_t, x_{t-1}^[m], m)
6:        X̄_t = X̄_t + ⟨x_t^[m], w_t^[m]⟩
7:     end for
8:     // generate new samples based on RNN estimation
9:     N = s · M
10:    for i = 1 to N do
11:       add x_t = generate_RNN_sample()
12:       add x_t to X_t
13:    end for
14:
15:    for i = 1 to M − N do
16:       draw i with probability w_t^[i]
17:       add x_t^[i] to X_t
18:    end for
19:    return X_t
```

Based on the networks estimation, the desired policy is achieved by choosing the action proving highest Q-value score.

In respect to the proposed model, the reward function's reward is increased when the distance to the goal decreases and reduced when the agent approximates other drones or crashes. For more detail about the used reward function, see Section 7.5.4. The employed DQN has two fully connected hidden layers with size 100 nodes each, see Figure 7.6. The input is a vector of size 1 x 7 and the output is a Q-action value vector with a size of 9. The possible actions are fixed accelerations in eight different directions with a 45° step size, the remaining action decelerates the drone's velocity. The DQN requires relative 2D coordinates to the goal (d_x,d_y) and the obstacle (o_x, o_y), provided by the particle filter, its own velocity (v_x, v_y) and the delta time dt since the last update. The input state is therefore defined as ($v_x, v_y, d_x, d_y, o_x, o_y, dt$). Where:

- $\boldsymbol{v_x}$: Velocity on x-axis
- $\boldsymbol{v_y}$: Velocity on y-axis
- $\boldsymbol{d_x}$: Distance to goal on x-axis
- $\boldsymbol{d_y}$: Distance to goal on y-axis
- $\boldsymbol{o_x}$: Distance to obstacle on x-axis
- $\boldsymbol{o_y}$: Distance to obstacle on y-axis
- $\boldsymbol{dt}$: Time passed since last update (delta time)

7.5 Training

The following section describes the training of the two employed neural networks. Section 7.5.1 describes the employed simulator, Section 7.5.2 illustrates the frame skipping technique to reduce the required training time. Section 7.5.3 details the training used for the recurrent neural network (RNN) and Section 7.5.4 concludes the section with a description of the Deep-Q Network (DQN) training.

7.5.1 Simulation

The training is conducted in a 2D python simulator with a simple inertia and noise models based on the employed drone and UWB-module (Section 7.3.2). The system is evaluated, in real life and simulation, on maximal four drones. In order to provide a challenging setting the test-area is kept relatively small, with a area size of $3\,\mathrm{m} \times 3\,\mathrm{m}$.

7.5.2 Frame Skipping

Frame skipping [11] is a common technique, used for playing 2D or 3D games with neural networks. This technique handles the problem that consecutive states or, in their case, images look similar. This makes it difficult to find features or patterns in a sequence of input states. The solution to the problem is to skip steps in between, however it is important that the action, which was applied before the skipping, is held until the next used frame. In the hardware setting, we are working in, this could also be referred to as update rate. In order to accomplish this, both our networks have to be synchronised. At each update step, the DQN is choosing an action based on the input state, provided by the RNN's predictions, the action is held until the next step etc. During training and execution the update-rate is kept at $8\,\mathrm{Hz}$.

7.5.3 Recurrent Neural Network

All parameters and setting used for the training of the RNN are summarised in Table 7.3. The RNN is trained in a supervised fashion, by gathering data over a longer time period and training the network offline (see Section 2.2.3). In our case, the data is a collection of 3000 episodes. One episode is a 10 minute scenario of two drones flying in a $3\,\mathrm{m} \times 3\,\mathrm{m}$ area. We are collecting data of both drones, this adds up to around 41 days worth of data. The simulated drones use a pre trained DQN policy on perfect position information and to a certain extent random moves. We are recording the direction to the other drone, the bearing to a fixed target point and the required input states. Since there is a relation between the goal position and the direction the movement policy is travelling, we are not recording the actual goal position, instead we are choosing a fixed point at random. The data is recorded without any sensor noise, the sensor noise is added to the data every time a new batch is drawn from the data set; this prevents the network from over fitting. The network draws every training-step a batch of b data sets, compares its prediction with the actual state and back-propagates the error through the network to adjust the

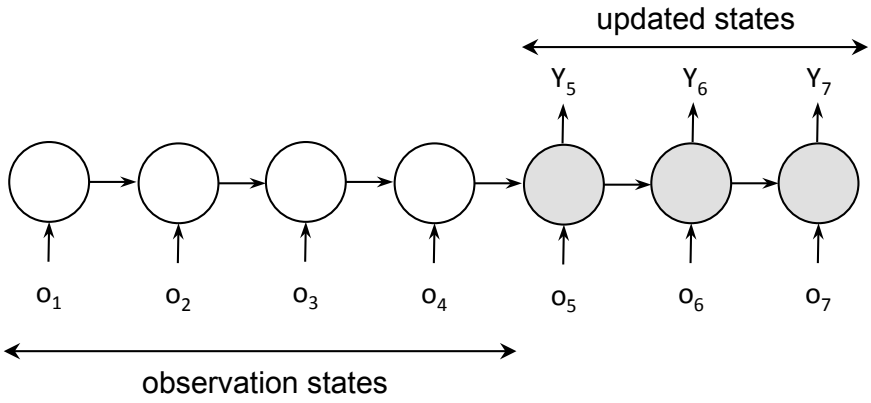


FIGURE 7.8: In this example only errors of $k > 4$ are back propagated through the network.

weights using gradient descent. During our training we keep the batch size b to a size of 30. As described in Section 7.4.3 is the bearing, the network is trying to estimate, divided into discrete angles with a step size of 20 degrees. The accuracy of the network can therefore be defined as the amount of the correctly predicted angle classes divided by batch size it is evaluated on.

The main purpose of the designed network, is the estimation of the current bearing to an object, based on the sequence of input states. Similar to the approach presented in [62] we are drawing sequences of observations of length n from a replay buffer. Since the states in the beginning of the sequence don't have a lot of information to rely on, we are only considering states, which have at least k predecessor states. Therefore, we only back propagate errors through the network for states k and higher as illustrated in Figure 7.8. The converging speed depends on number of states considered for the update. Figure 7.9 shows that it takes the optimiser, which considered the last ten states, significantly less iterations to reach a 90% accuracy, than the optimiser which only considered one state. Based on these results we set the minimal history count k to 15 and we update the 10 consecutive states, which results in the sequence length n of 25.

For stability, we divide the recorded data into a training, validation and test set. The ratio is 64% for training, 16% for validation and 20% for test data. The sets are randomly sampled. The networks training and hyper parameter tuning is done on the training set and its performance is validated on the validation set, during training. The test-set is used as a final confirmation of the training performance after the training, to avoid overfitting.

As a metric, we apply the mean accuracy of all predicted angles and their target value. We save the weights, when the accuracy improves over a fixed number of validations. The training takes approximately eight hours, on our setup and converges towards a accuracy of 95%. During training, the weights are updated using the RMSProp algorithm with a learning rate of 0.001. The learning rate determines how aggressive the optimiser

Parameter	**Setting**	**Description**
Training size	64%	Percentage of total set used for training.
Validation size	16%	Percentage of total set used for validation.
Test size	20%	Percentage of total set used for final testing.
b	30	Batch size during training.
n	25	Length of state sequence used, to train the network.
k	15	Number of states, in the beginning of the sequence, not considered for the update of the network, see Figure 7.8.
Optimiser	RMSProp	Algorithm used to optimise the weights, based on current prediction error.
Learning rate	0.001	Determines the rate the optimiser adapts the weights.

TABLE 7.3: Parameters employed to train the recurrent neural network.

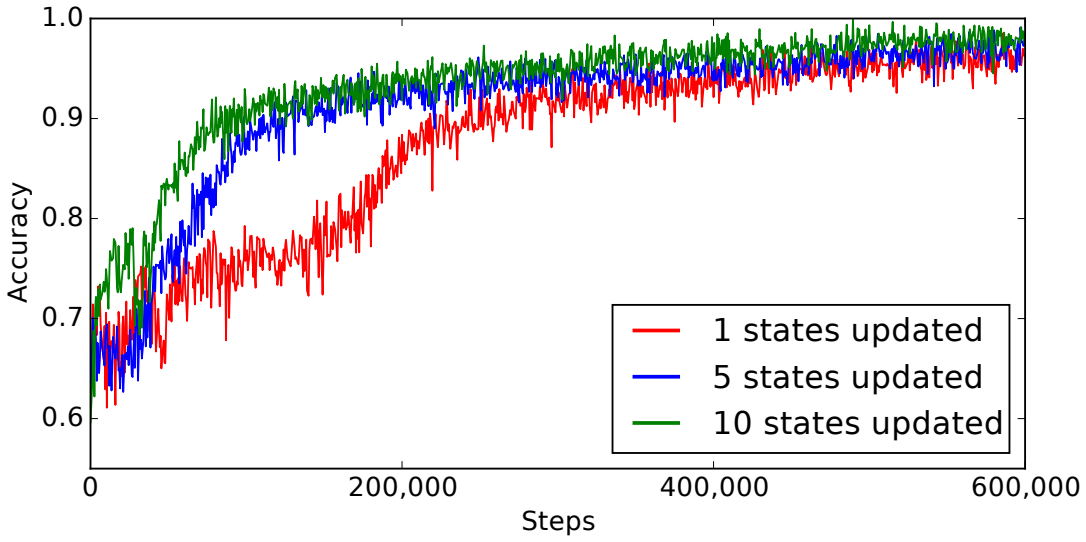


FIGURE 7.9: Converging behaviour based on the number of states considered for the update.

updates the weights, to match the outputs from the current batch. All previous described parameters are summarised in Table 7.3.

7.5.4 Deep Q-Network

The main goal of the Deep Q-Network (DQN) is to learn a policy to guide the drones towards a target position and avoid other drones on the way. During the training process the agent is continuously interacting with the environment and recording the current state s_t, the taken action a_t, a reward r_t returned by the environment and the state s_{t+1} it transitioned to. These recording are usually known as experiences:

$$e_t = (s_t,\ a_t,\ r_t,\ s_{t+1})$$

These experiences are stored in an experience buffer of size N, where the oldest experience is dropped, when the storage-size exceeds N. The training is done in an online fashion, where the network weights are continuously updated, while the agent is interacting with the environment. Every i time steps, a batch of b experiences is drawn from the buffer to train the network. The aim of the training is to approximate a function, representing the long-term reward of taking an action a in a certain state s. The learning process is categorised as reinforcement learning and the function it approximates, is called Q-function. For more detail about reinforcement learning and DQNs see Section 2.3.

The employed reward function is designed to incentivise actions bringing the drone closer to the goal, while keeping a certain safety distance to the surrounding drones. Therefore, we define the reward as the negative distance to the goal. If the distance to another agent is under a defined threshold c, we add a second negative reward as the inverse of this distance. Both rewards are normalised and can be defined as follows:

$$r = \begin{cases} -d_1, & \text{if } d_2 \geq c \\ -d_1 + p \cdot \dfrac{d_2 - c}{c}, & \text{otherwise} \end{cases} \tag{7.3}$$

With d_1 as the goal distance, d_2 the distance to the closest drone and c being the safety distance, the agents are trying to maintain. Additionally, we apply the fixed factor p to increase the negative impact of going below the safety distances. The network is trained with the RMSProp algorithm and a learning rate of 0.0003. All previously mentioned parameters and their settings are described in the following Table 7.4.

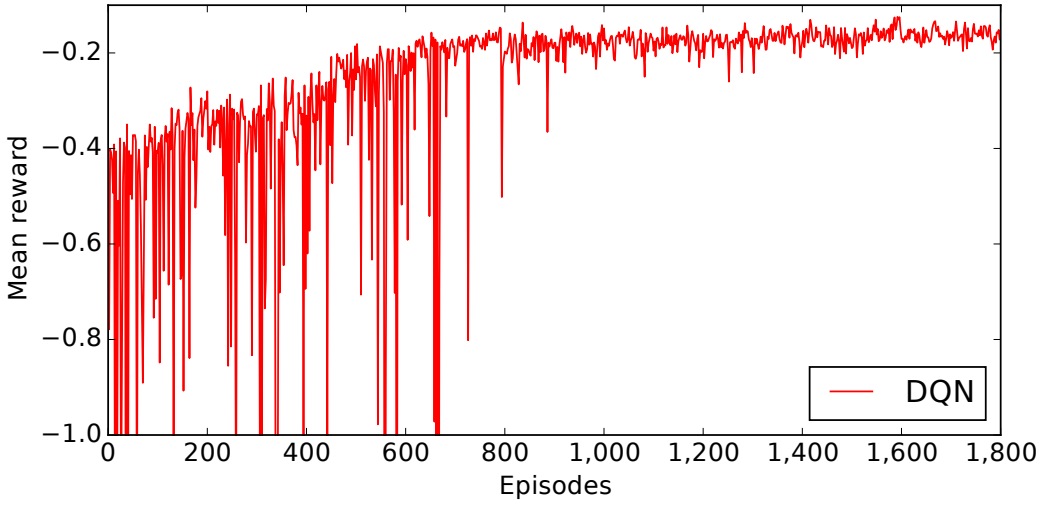


FIGURE 7.10: DQN's average estimated reward over 100 randomly selected states.

In reinforcement learning we can't evaluate the model on a pre recorded validation set. In order to tackle this problem, we apply the evaluation metric suggested by [40], where the evaluation selects m random states in the beginning of the training. After

each iteration, we calculate the maximal reward possible for each state, based on the estimation of the current DQN. The calculated rewards are averaged over the number of states:

$$x = \sum_{i=1}^{m} \frac{Q_{max_a}(s_i, a)}{m} \tag{7.4}$$

Every time a DQN weight configuration is able to surpass the previous achieved average, its weights are stored. A converging of the networks prediction, is indicated if the average remains constant for a longer period of time. Depending on the machine, the training takes around three hours to converge to a stable state. This can be seen in Figure 7.10, where the network reaches a constant estimation after around 1000 episodes. For the action selection, during training we apply ε-greedy. In this selection process has the learner a probably of ε, of choosing a random action and a $1 - \varepsilon$ probability of selecting an action based on the networks Q-Value. To explore the reward-function, ε is initialised with $\varepsilon_{start} = 80\%$ and decayed linear over time to its final value of $\varepsilon_{end} = 10\%$. We set the duration ε is reduced in, to $\varepsilon_d = 90\,\text{min}$. All previously discussed parameters and their settings are summarised in Table 7.4.

Parameter	**Setting**	**Description**
s_t	–	State at time step t.
a_t	–	Action taken at time step t.
r_t	–	Reward gathered at time step t.
b	30	Batch size used during training.
N	5000	Maximum size of replay buffer.
d_1	–	Distance to goal.
d_2	–	Distance to closest drone.
c	30 cm	Minimal safety distance.
i	2	Determines step frequency the network is trained in.
m	100	Number of randomly selected states to evaluate the DQN's performance.
ε	–	Probability of choosing a random action during training.
ε_{start}	80%	Value ε is initialised with.
ε_{end}	10%	Final value ε is reduced to, over time.
ε_d	90 min	Duration in which ε is reduced from ε_{start} to ε_{end}.
p	2.0	Factor to amplify the negative reward, caused by approaching drones under the safety distance.
Optimiser	RMSProp	Algorithm used to optimise the weights, based on current prediction error.
Learning rate	0.003	Determines the rate the optimiser adapts the weights.

TABLE 7.4: Parameters employed to train the Deep Q-Network.

System	Description
RNN	The pose-estimation is exclusively done by the RNN.
Hybrid (30)	The pose-estimation is done by combining the estimation of the RNN together is a particle filter using 30 particles.
Particle (30)	The pose-estimation is exclusively done with a particle filter using 30 particles.
Particle (100)	The pose-estimation is exclusively done with a particle filter using 100 particles.
Particle (300)	The pose-estimation is exclusively done with a particle filter using 300 particles.
Particle (900)	The pose-estimation is exclusively done with a particle filter using 900 particles.

TABLE 7.5: Settings employed in evaluation.

7.6 Simulation Experiments

The following Section describes the initial evaluation done in simulation. Since the proposed system employs a recurrent neural network RNN and a particle filter, we compare its performance with the components it consists of. The system is divided in two parts, the first part is responsible for estimating 2D-poses of drones and goals, based on a sequence of input states and the second part handles the navigation and collision avoidance. Section 7.6.1 illustrates the setting each algorithm is tested in, Section 7.6.2 analysis the different configurations in respect to the pose-estimation performance and Section 7.6.3 investigates the navigation performance of the combined system.

7.6.1 Evaluated Configurations

Since the navigation part of the system consist of a neural network, which doesn't doesn't require any extra parameters to operate, we mostly compare the pose-estimation sub-system in different configurations. A system exclusively relying on the estimation provided by the recurrent neural network (RNN), will be referred to as RNN-system, a system exclusively relying on a particle-filter will be referred to as Particle-system and the system combining those previously mentioned systems, will be referred to as Hybrid-system.

The particle-system is evaluated in different number of particle sizes: 30, 100, 300 and 900. In the hybrid-system, we aiming to keep the number of particle required low and evaluate its performance with 30 particles. All evaluated configurations are summarised in Table 7.5. The simulated distance calculation employs the noise-model of the UWB-module, described in Section 7.3.2. The environment, the algorithms are evaluated in has a size of $3\,\mathrm{m} \times 3\,\mathrm{m}$.

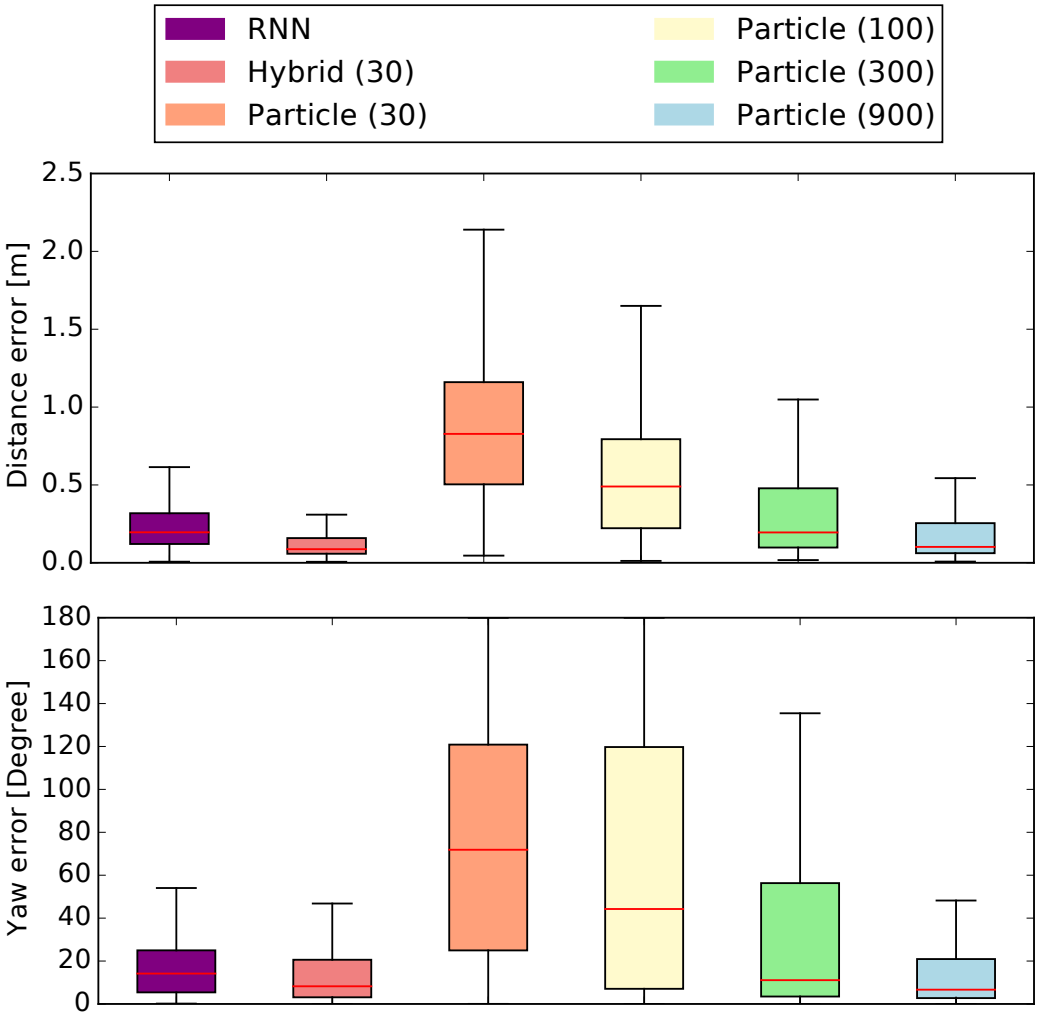


FIGURE 7.11: Error distribution achieved by different estimation configurations, in respect to distance and orientation estimation.

7.6.2 Pose Estimation

The pose estimation is evaluated in a setting with two drones. The estimation is isolated from the movement, where the simulated drones are controlled by DQN base on perfect position information. The DQN is navigating the drones towards goal-positions and avoids the other drones (obstacles), if necessary. During this section, we evaluate the accuracy of the different system configurations, presented in the previous Section 7.6.1. The systems are continuously estimating the positions and orientations of the goal and obstacles, in the frame of the drone they are executed on. The goal position, the system has to estimate, is independent from the goal the drone is navigating to and randomly chosen in the beginning of a run. Selecting an independent goal for the estimation, allows us to see how stable the estimation performs over a longer period of time. The navigation goal is altered as soon as it is reached. This enables a continues movement of the drones.

Algorithm	**Mean distance error**	**Standard deviation**
RNN	0.26m	0.25m
Hybrid (30)	**0.17m**	**0.19m**
Particle (30)	0.94m	0.61m
Particle (100)	0.54m	0.38m
Particle (300)	0.33m	0.31m
Particle (900)	0.20m	0.23m

TABLE 7.6: Mean-error and standard deviation (position estimation).

Algorithm	**Mean yaw error**	**Standard deviation**
RNN	23.41°	31.39°
Hybrid (30)	**20.66°**	**30.95°**
Particle (30)	75.64°	53.95°
Particle (100)	63.74°	60.20°
Particle (300)	35.61°	46.49°
Particle (900)	21.50°	34.99°

TABLE 7.7: Mean-error and standard deviation (orientation estimation).

The metric to determine the position accuracy is the euclidean distance between the actual position and its estimation. The orientation error is defined as the shortest angular distance between the true angle and estimated angle. Each configuration is executed over 30 runs with a runtime of 90 s.

Figure 7.11 illustrates the error distributions over all runs. Tables 7.6 and 7.7 show the mean-error and standard deviation for the position and orientation estimation and Figure 7.12 presents the estimation progression over a representative single run.

When we consider the particle-filter exclusive configuration, we can see that the performance increases proportional to the number of particles applied. This behaviour can equally be observed in the position estimation, as in the relative orientation estimation. Table 7.6 shows a steady improvement starting with and mean distance-error of 94 cm for 30 particles and ending with mean distance-error of 20 cm for the 900 sample configuration. Table 7.7 show the same effect in the estimations of the yaw-orientation. Ranging from mean-error of 21.50° degrees for 900 particle setting to the maximal mean error of 75.64° degrees in the 30 particle-system. Additionally, we can see the same effect for the standard deviation of both metrics, shown in Table 7.6, Table 7.7 and Figure 7.11. In terms of stability, we can see in Figure 7.12, that a high particle count, especially in the beginning, increases the chance of converging to the accurate solution, but this causes an additional computational overhead discussed in Section 7.9.

The result in Table 7.6, Table 7.7 and Figure 7.11 show that the RNN-system is able to surpass the performances of most particle-based configurations, by performing slightly under the performance of the highest particle configuration. Its mean-error is around 26 cm for the position estimation and 23° for the orientation. An advantages of the RNN can be seen in Figure 7.12, where it is able to converge the fastest to a relative

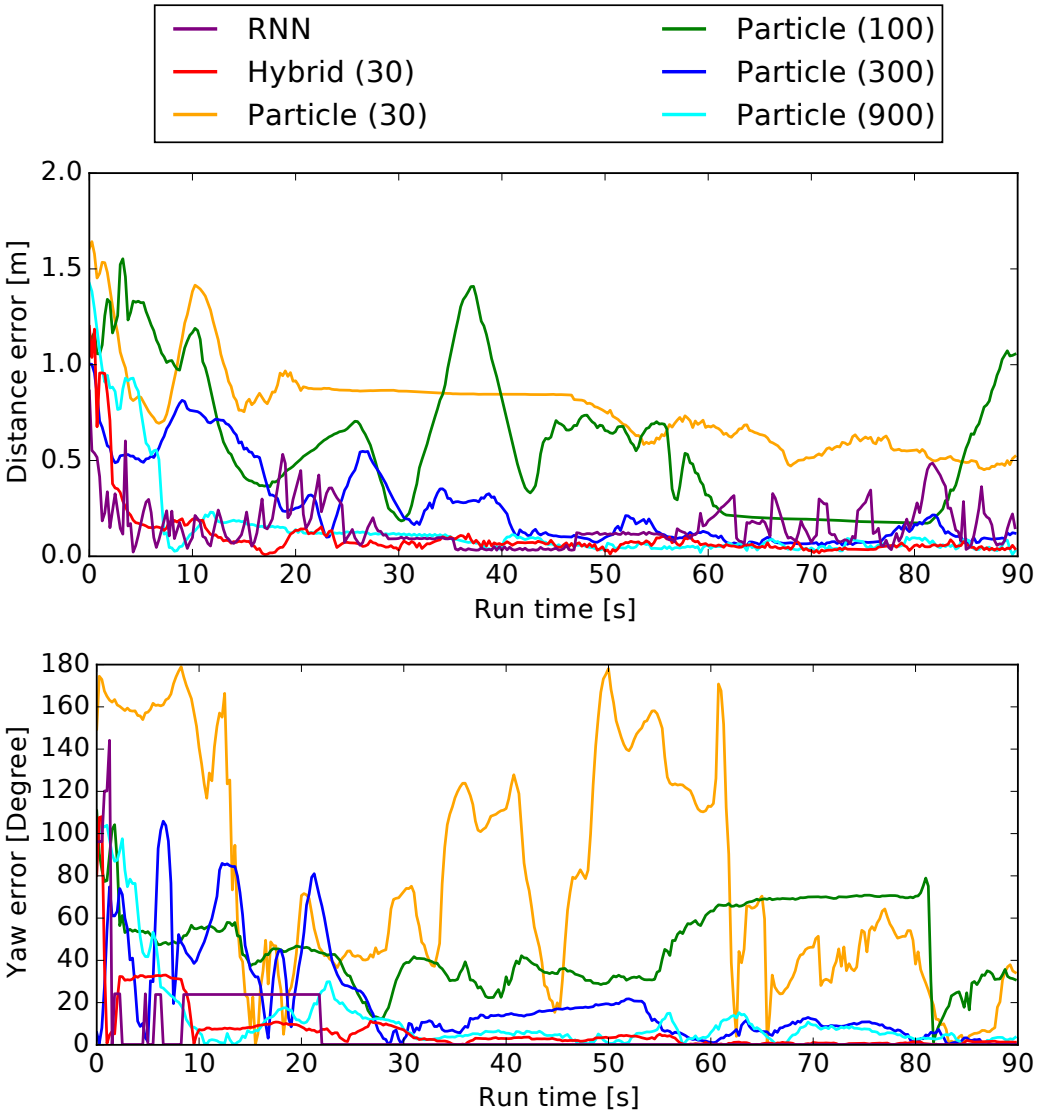


FIGURE 7.12: Performance over a single run

accurate position and angle estimation, but the figure shows also occasional spikes or outliers in the estimation.

Lastly, the Table 7.6 and Table 7.7 show that the proposed hybrid system is able to the achieve the lowest standard deviation and mean error of all tested evaluation metrics. With a mean-error of 17 cm for the position estimation and 20.66° for the orientation estimation. It is able to utilise the fast initial estimation made by the RNN, while providing a stable estimation through the particle filter (see Figure 7.12). Figure 7.12 shows that the approach requires around to 10s converge to a stable estimation. When we consider the estimation error after the initial phase the hybrid approach achieves a mean-error of around 0.07 cm with a standard deviation of 0.07 cm. All results are further discussed in Section 7.9.

Algorithm	**1 Robot**	**2 Robots**	**3 Robots**	**4 Robots**
RNN	8.65	8.73	7.74	5.36
Hybrid (30)	**9.95**	**9.00**	**7.92**	**6.07**
Particle (30)	0.70	0.63	0.40	0.54
Particle (100)	1.95	1.93	1.72	1.87
Particle (300)	5.65	5.49	2.99	2.58
Particle (900)	7.95	7.78	3.83	2.20

TABLE 7.8: Average goals per minute achieved in a run.

7.6.3 Navigation Performance

Where the previous Section 7.6.2 evaluated the pose estimation isolated from the motion controller. This section investigates to what extent the position estimation and the number of drones is influencing the navigation and collision avoidance of the system. The system is evaluated on $1-4$ robots in a $3\,\mathrm{m} \times 3\,\mathrm{m}$ area, employing the different estimation configuration introduced in Section 7.6.2. Each configuration is executed 50 times, with a run time of 90 s. The drones are initialised at random a position in the environment. In order to test the navigation, each drone is given a goal point it has to reach. As soon as drone is in proximity of a goal it is given the next one etc. A run is prematurely terminated when two drones collide.

As described in Section 7.4, the system is only given the distance to the goal and the position estimation provided by the internal estimation model. By only employing the goal distance, we are able use other UWB-modules as goals, this functionally is beneficial for future applications e.g. foraging and search and rescue scenarios.

In order to evaluate the performance of the navigation, we measure the average goals "collected" per minute and the percentage of total runs completed. The results are illustrated in Tables 7.8 and 7.9.

In terms of navigation, we general can see, that the rate of goals reached per minute decreases with an increasing number of drones (Table 7.8). This can be explained by the fact that a higher number of drones forces the control network (DQN) to chose longer routes, in order to avoid collisions. Therefore, we can consider the setting with a single drone as benchmark for the other configurations of the same algorithm. This shows that the navigation performance improves, when the pose estimation has a quick converging time. Since the RNN- and Hybrid-system converge the fastest, they are able to reach the most goals per minutes (Table 7.8). The reason for this is, that the faster the estimation converges towards the true position, the faster is the control network able to reach the target position.

Table 7.8 and Table 7.9 show that the drone number increase, has a higher negative impact on the configurations, exclusively relying on particle filter estimation. Especially on the collision avoidance shown in Table 7.9. The proposed hybrid model is able to finish 98% of the runs with 3 drones and 88% the runs with 4 drones. The best performing particle-system reaches 72% of the runs with 3 drones and 36% the runs with 4 drones. This is due to the converging speed and the design of the estimation system. Since we

Algorithm	**1 Robot**	**2 Robots**	**3 Robots**	**4 Robots**
RNN	100%	92%	76%	56%
Hybrid (30)	100%	**100**%	**98**%	**88**%
Particle (30)	100%	86%	42%	28%
Particle (100)	100%	82%	56%	16%
Particle (300)	100%	92%	72%	36%
Particle (900)	100%	100%	72%	36%

TABLE 7.9: Percentage of completed runs, in respect to the different estimation configurations.

are trying to avoid computational overhead, because of the execution on pocket drones, we only consider the closest drone for the estimation and consequently the collision avoidance. This means the estimation system has to provide a fast estimation of new drones position, if the drone of interest changes.

The hybrid system provides the overall best performance, but still has some collisions in the setting with a higher robot count. One of the reasons for this is, that the simulated environment is surrounded by walls and it can happen that the DQN triggers an avoidance action, which results in a collision with a wall and eventually with the drone it is trying to avoid. This is because the current drone hardware is not able to provide a distance measurement to the walls and is therefore not considered in the current motion decision of the DQN. Additional sensors like these, will be included in the next hardware interaction as part of the future work.

7.7 Computational Overhead

The major benefits of the introduced hybrid system is that it is able to match the accuracy and converging time of a system with high particle count, while minimising the computational overhead. Table 7.10 shows the time each system requires to estimate the position of the other drones. In order to investigate how the system scales, we evaluate the estimation times of up to 3 drones. The calculation time is averaged over 30 runs, for each configuration. As shown in Table 7.10, RNNs estimation requires, on our setup,

Algorithm	**1 Robot**	**2 Robot**	**3 Robot**
Particle (30)	11.8528 ms	21.0967 ms	33.4906 ms
Particle (100)	20.5191 ms	43.5611 ms	66.3835 ms
Particle (300)	47.1051 ms	94.0268 ms	137.2816 ms
Particle (900)	125.0071 ms	243.4804 ms	377.6352 ms
Hybrid (30)	15.6430 ms	30.5485 ms	43.4412 ms
RNN	4.3397 ms	9.3460 ms	13.4269 ms

TABLE 7.10: Computation time each configuration requires to estimate the positions of 1 - 3 drones.

only around 4.33 ms per drone (see Table 7.10 row 6). By adding the small particle-filter in the hybrid configuration, we add an additional 11.85 ms per additional drone,

FIGURE 7.13: Employed motion capture system to provide the drones with velocity information and for recording position informations for post-performance evaluation. The tracking cameras can be seen in the top-part of the image.

while improving the accuracy as discussed in Section 7.6. All systems scale linear, since each additional drone increases the computation time around the same amount. When we compare the proposed hybrid system with the particle filter with a similar accuracy (900 particles), we are able to reduce the computation time by a factor of 8, which is important for update-rate dependent applications e.g. collisions avoidance.

7.8 Real-world Experiments

A practical use-case for pocket drones is mostly an indoor environment, since they are light and sensitive to strong winds and gust. Their advantage against ground robots is their ability to fly and therefore the option to avoid obstacles in the third dimension. Furthermore, they are able to observe the environment from a top-down perspective, which can be an advantage in certain use cases, e.g. surveillance and exploration. The tests are therefore applied exclusively indoors, in the robotic-lab of the University of Liverpool. The test area measures a size of around 4m $\times$ 4m. To execute the developed approach on real-drones, we employ the open-hardware platform called "crazyflie", described in Section 7.2. Additionally, we employ an optical tracking system to provide the drones with valuable velocity information and track their position, in order to evaluate the proposed systems position estimation, see Figure 7.13. Similar to the tests in simulation, the evaluation described in this section is twofold. Section 7.8.1 describes the performance in respect to the position estimation sub-system and Section 7.8.2 shows complete prosed model performing navigation and collision avoidance.

7.8.1 Position Accuracy

As described in Section 7.4, the proposed model is divided into two sub-systems. The first system estimates the position and orientation of other drones and goals, based on

System	**Description**
RNN	The pose-estimation is exclusively done by the RNN.
Hybrid (30)	The pose-estimation is done by combining the estimation of the RNN, together with a particle filter using 30 particles.
Particle (30)	The pose-estimation is exclusively done with a particle filter using 30 particles.
Particle (100)	The pose-estimation is exclusively done with a particle filter using 100 particles.
Particle (300)	The pose-estimation is exclusively done with a particle filter using 300 particles.
Particle (900)	The pose-estimation is exclusively done with a particle filter using 900 particles.

TABLE 7.11: Settings employed in evaluation.

the input states described in Section 7.4.2. The second sub-system is responsible for the control of the drone.

During this section, we are exclusively focusing on the pose-estimation performance of the system. The pose-estimation combines a particle-filter with recurrent neural neural network to provide a fast and stable estimation (see Section 7.6). Similarly to the evaluation in simulation, we compare the hybrid system against the components it consist of.

A system exclusively relying on the estimation of the recurrent neural network (RNN), will be referred to as RNN-system, a system exclusively relying on a particle-filter will be referred to as particle-system and the system combining those two techniques will be referred to as hybrid-system. The particle-system is evaluated in different number of particle sizes: 30, 100, 300 and 900. In the proposed hybrid-system, we aiming to keep the number of particle required low and evaluate its performance with 30 particles. All evaluated configurations are summarised in Table 7.11.

In order to estimate the system performance on the real UWB-module's distance data, we employ three drones. Two drones performing a random walk and the third drone poses as a fixed "goal" pose, in a save distance. The random walk is performed by the movement network (DQN), on perfect position information provided by the motion capture system.

All drone positions, velocities and distance informations (UWB-module) are recorded over 20 runs. The position of the "fixed" drone is altered in every run. The estimation and its evaluation is executed offline, on the recorded data. This allows us to test the system in every possible setting and reduces the number of flights required.

By considering the view, of one of the moving drones, the system has to estimate the positions and orientations of the two remaining drones. As metrics, we apply the euclidean distance error between estimated and actual positions as well as the difference between estimated and actual orientation (yaw-rotation). Both results are illustrated in Figures 7.14 and 7.15 and Tables 7.12 and 7.13. The number in the brackets behind

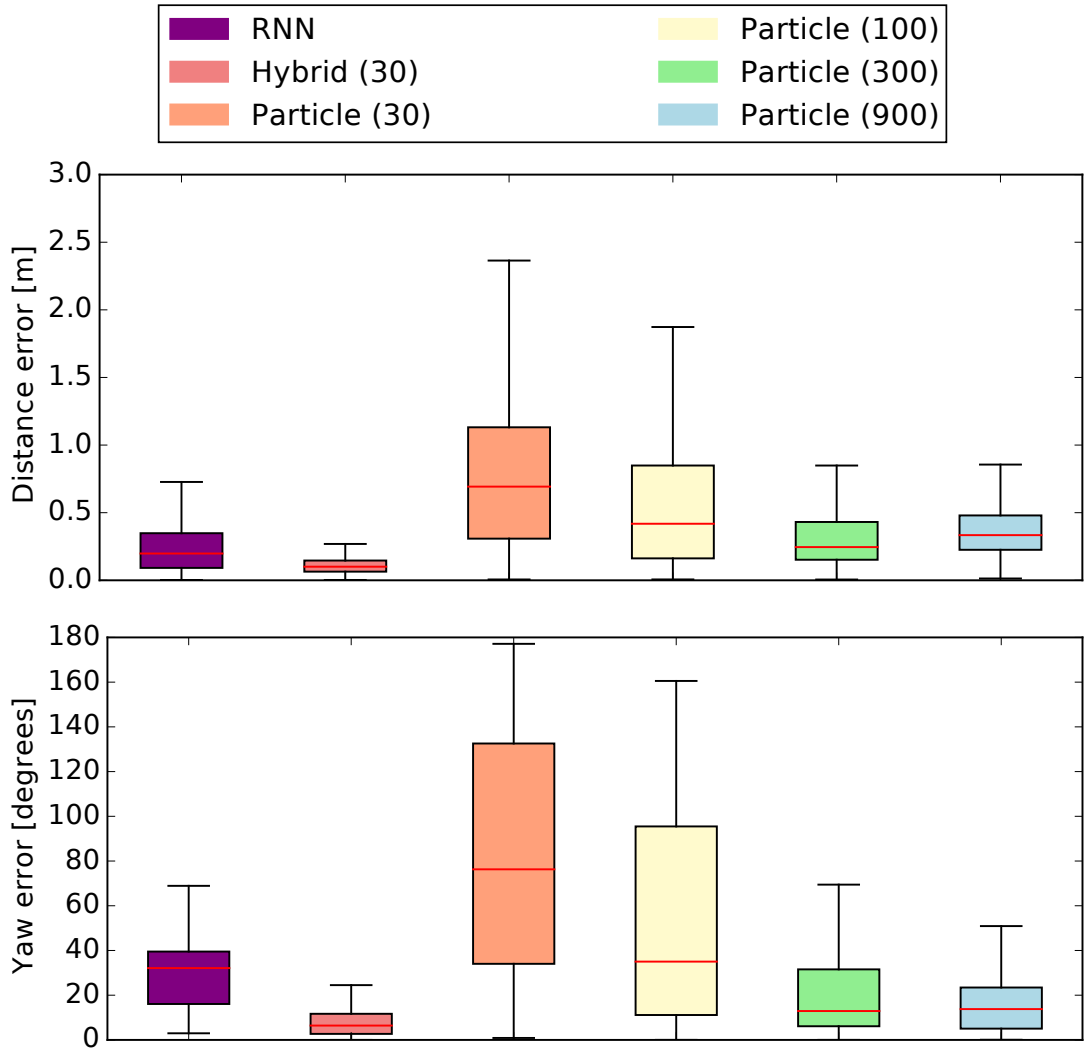


FIGURE 7.14: Error distribution achieved by different estimation configurations, in respect to distance and orientation estimation.

the name of the approach, indicates the amount of particles used. The figures and tables show a similar behaviour seen in the simulated tests. Figure 7.14 shows the average error over all runs and reveals that the RNN has a similar error profile as the particle filter configurations with higher particle counts. The distance error between estimated obstacle/goal position is around 34cm and the average error of the estimated obstacle orientation is around 31°. The performance of the particle filter exclusive configurations is improved by an increasing the particle count, but the tests show that configuration with higher particle count have the tendency to converge to multi possible solutions/particle clouds, which explains the slight drop in performance between the 300 and 900 particle configuration. A lower particle count configuration is faster in terms of computation, but it less likely to converge to a correct solution and if the particles are drifting away from the correct solution, it is less likely to recover. The evaluation shows that combination of RNN and particle filter is a promising compromise, it doesn't require a high quantity of particles to keep the error low (see Figure 7.14) and it converges

Algorithm	**Mean yaw error**	**Standard deviation**
RNN	31.00°	34.82°
Hybrid (30)	13.56°	18.82°
Particle (30)	89.76°	49.11°
Particle (100)	58.93°	48.28°
Particle (300)	28.88°	34.37°
Particle (900)	17.80°	13.58°

TABLE 7.12: Mean-error and standard deviation (orientation estimation).

Algorithm	**Mean yaw error**	**Standard deviation**
RNN	0.34m	0.44m
Hybrid (30)	0.17m	0.27m
Particle (30)	0.79m	0.56m
Particle (100)	0.57m	0.50m
Particle (300)	0.37m	0.35m
Particle (900)	0.38m	0.22m

TABLE 7.13: Mean-error and standard deviation (orientation estimation).

quicker to a accurate estimate than the best performing particle filter configurations (see Figure 7.15).

When comparing the results between real-world experiments an the simulation, we can see that most configuration achieve a similar performances in both environments. The only setting, suffering from a significant drop in performance is the configuration using 900 particles. This can happen when addition noise of movement and distance measurements is introduced and multiple particle clouds have a high probability of representing the correct solution. The higher the particle count the more likely is the converging to multiple solutions.

This illustrates additionally the robustness of the hybrid-system. When the particle filter is stuck in a local minima, the network can introduce new plausible positions, pushing the estimation towards the actual solution. Figure 7.15 shows the performance of each configuration in an average run. It shows that the pure RNN and the hybrid-configuration are converging quicker, than the pure particle based configurations, to a accurate position estimation. After an initial phase, the hybrid approach is able to maintain a low estimation error, where the RNN shows some outliers.

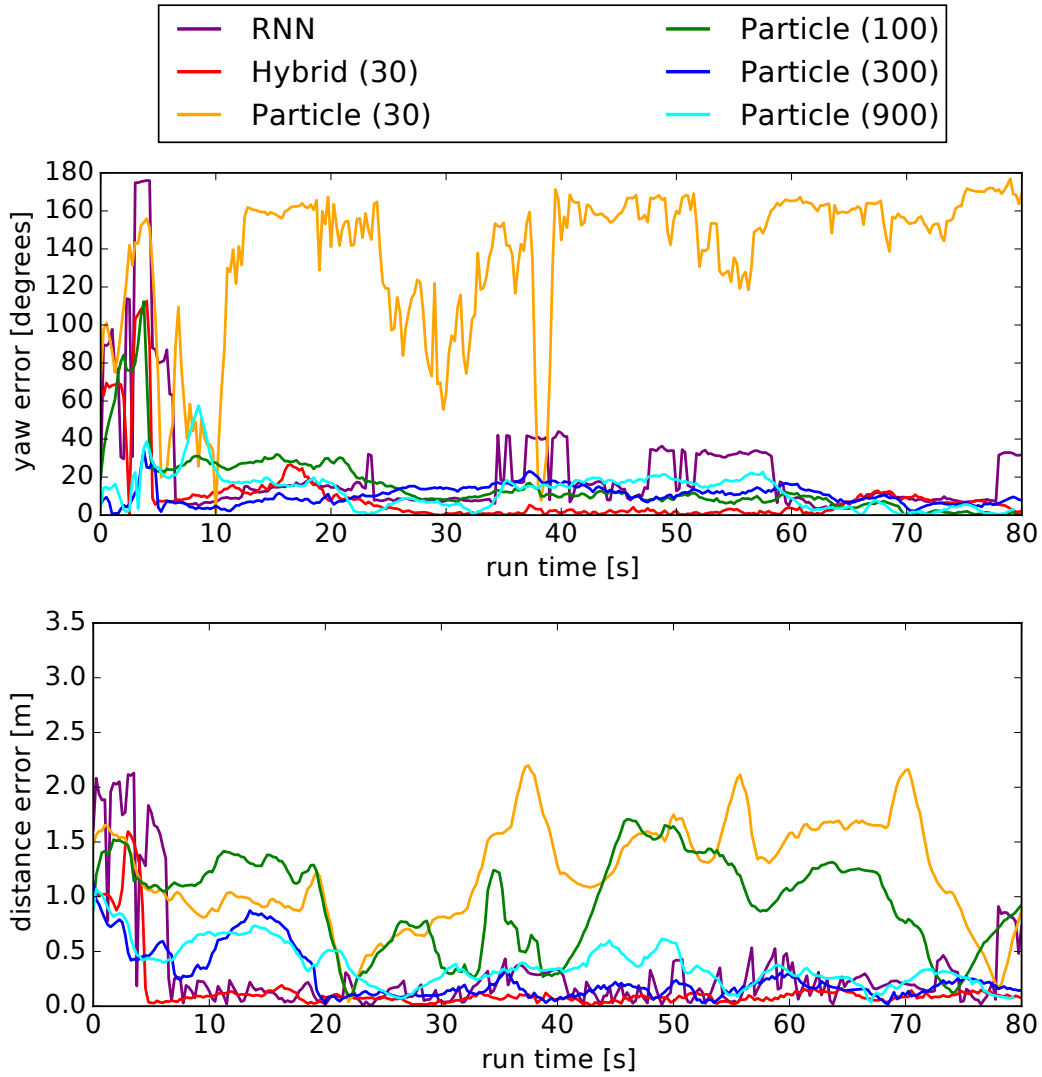
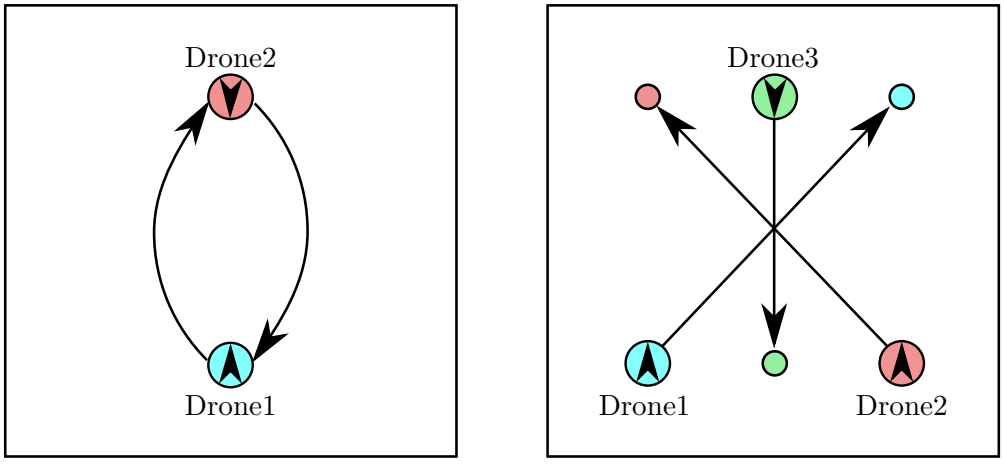


FIGURE 7.15: Performance over a single run

7.8.2 Navigation

The previous Sections 7.6 and 7.8.1 showed that the proposed hybrid approach is able to provide a stable and accurate position estimation with a fast converging time, in simulation and with real sensor data. The final tests, described in this sections, are designed to examine the control network (DQN), in combination with its position estimation subsystem. The position estimation is exclusivity executed by the hybrid-model, introduced in Section 7.4.

The complete system, is tested in two types of scenarios. A random goal-scenario and a pre-defined goal scenario. In the former scenario the drones are give random destinations, as soon as a drone is in proximity of its goal, it is assigned to a new goal. This scenario is designed to check, if the drones are able to operate in a relative confined space for a longer period of time without crashing. It is executed for $2-4$ drones, with a runtime of two minutes.



(A) (B)

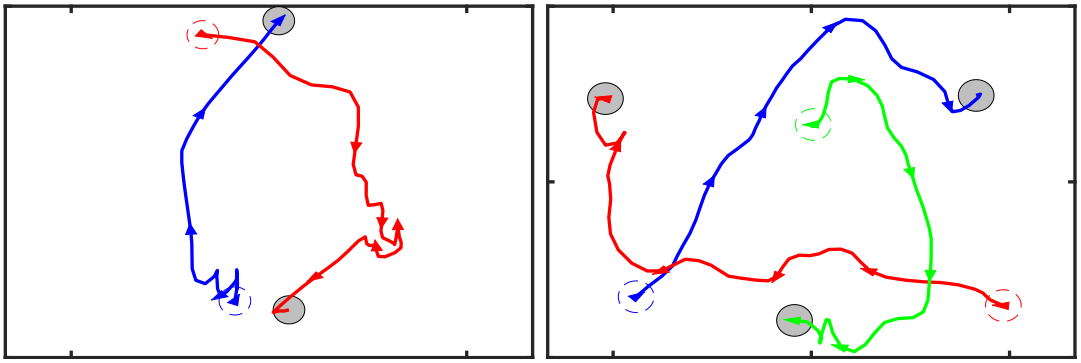
FIGURE 7.16: Fixed goal-point scenarios.

The pre-defined goal setting, is designed to compel the drones to avoid collisions. Figure 7.16a illustrates the first scenario, where two drones have to switch places. This means, if they chose the shortest route possible they are on a direct collision path. The second scenario, positions three drones in a equilateral triangle shape, their destinations are chosen in a way, that each drone has to cross the path of the two other drones, see Figure 7.16a. Each scenario, including the random flights, are executed 10 times.

The simulation experiments in Section 7.6 showed that system is able to provide a sufficient collision avoidance even in drone denser scenarios. One issue, mentioned in Section 7.6, is that the network responsible for the movement, does not take the walls into account, which will be tackled in future work. In order to take this problem out of the equation, all selected goals have at least a 1 m buffer to the outside of the test environment.

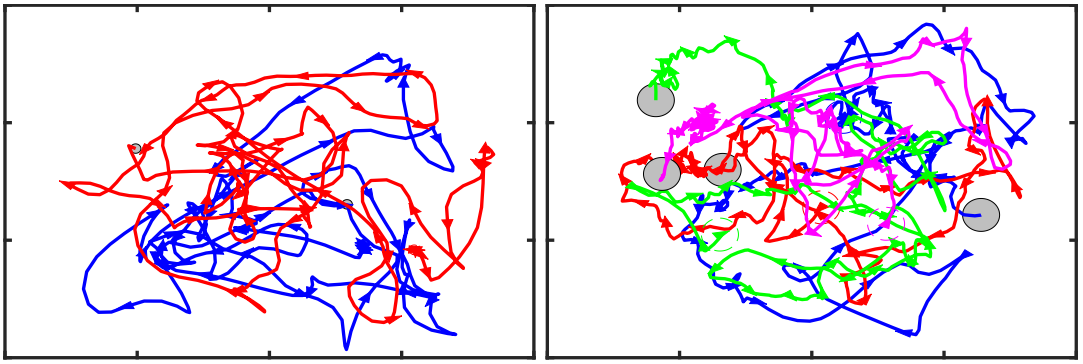
One additional precautionary measure, is to restrict the maximal velocity of drones in close proximity to one another. The network responsible for the movement is trained to maintain a safety distance from at least 30 cm, see Section 7.5.4. Therefore, we limit the velocity of two drones to 10 cm/s, if there distance between each other is smaller than 30 cm (trained safety distance).

The proposed control and estimation model are executed externally and are transferred to the software stack of the drone as part of the in future work. By applying the previous mentioned precautionary measures, all runs were finished without any collisions. Figure 7.17d illustrates a representative run of 4 drones with its traversed trajectories and the continues movement of the drones. Table 7.14 shows to what extent the movement policy benefits from the "slow-down"-strategy and its total activation time during a run. The total duration the strategy is active, increases with the number of robots. Starting with a 0% activation in the 0 drone setting and increasing to a 10% activation, over the full run-time, in the 4 drone setup.



(A) 2 Drones: Fixed goal points.

(B) 3 Drones: Fixed goal points.



(C) 2 Drones: Random goal points.

(D) 4 Drones: Random goal points.

FIGURE 7.17: Path-recordings of the real-world tests.

	2 Robot	**3 Robot**	**4 Robot**
Slow-down active(%)	0%	4%	10%

TABLE 7.14: Illustrates the percentage of total run-time (120 s), the drones using the reduced velocity-strategy, in order to avoid collisions.

Finally, Figures 7.17a and 7.17a presents the trajectories in the fixed goal setting and show that the learned policy is able to reach their final destinations, but prefers wider paths to avoid collisions. The figures show additionally, that the system is executing curves in the beginning of the path, before it converges towards a direction to the goal. The behaviour is caused by the fact, that the estimation, requires a sequence of velocity and distance inputs to approximate the goals position. Since the goal is not moving, the estimation is exclusivity relying on the velocities of the drone executing the estimation, the first "random" movement is required to converge the estimation towards the actual goal position.

The experiments show that the trained networks are transferable to the real-life system and are able to provide the required navigation and collision avoidance ability. All results are further discussed in Section 7.9 and a video of the previous mentioned experiments can be found at: `https://youtu.be/yj6Qqh0zpok`

7.9 Discussion

The previous experiments in Section 7.6 and 7.8, showed that the proposed system is able to provide the required pose estimation, navigation and collision avoidance on pocket drones. In terms of pose estimation, achieves the model consisting of a recurrent neural network (RNN) and a particle-filter, a fast and accurate estimation and is able to outperform standard particle-filter implementations (Figures 7.11, 7.12, 7.14, 7.16 and Tables 7.6, 7.7, 7.12, 7.13). The results show additionally, that the trained RNN is able to provide a sufficient estimation on its own, which reduces the number of particles, required to stabilise the estimation to a minimum. The experiments show that particle filter, especially with a small particle count have a tendency to converge to a local minima, the "educated guess" from the RNN allows to overcome this issue. The fact that the network of the hybrid system uses only one layer of LSTM-cells (Section 7.4) and the particle filter requires only a small quantity of particles (30), reduces the computational power needed, which makes the system attractive for ultra-light platforms e.g. pocket drones. Lastly, the comparison between the performances in the simulation- and real-world environment, showed that the system is able to provide a similar performance in both scenarios. This displays the robustness of the system, since the simulation and the underlying models are only a simplified approximation of the real-world. Since we use the same parameters in simulation and real-world, we can conclude that the system performance doesn't rely significantly on parameter tuning to derive the required accuracy.

In regards to proposed DQN network, we showed that is able provide the required navigation and collision avoidance with multiple drones, in the simulated and real environment. As seen in Tables 7.8 and 7.9 is the performance depending on the quality of the pose estimation. The experiments showed that the estimation by the proposed model is sufficient, if certain precautionary measures are given. These measures include a safety distance to the wall and a limitation of velocities, for drones in close proximity. Since the employed reinforcement learning approach is able to learn on its own and the employed network-structure is relatively compact, we are able train the network with additional input information, e.g. distance to wall, in a relative short time period (3h). Additional sensors for wall detection and internal velocity estimation are part of the future work, discussed in Section 8.

7.10 Conclusion

During this chapter we address research questions RQ4 and RQ5:
Research Question RQ4: How can modern sensors and technology be applied to provide a bio-inspired coordination on pocket drones?
Research Question RQ5: To what extent can deep learning be beneficial for the performance of the hardware discovered in RQ4?

We show how modern communication technology in form of UWB-communication provides a ultra light weight solution to calculate distances between robots and show how deep-learning can be applied to utilise this data to offer low computational navigation and robot-robot localisation to high level coordination principles.

The aim of the system, described in this chapter, is to provide position estimation, navigation and collision avoidance to high-level coverage techniques e.g. BeePCo (Section 4), to transition them from a ground to a aerial coverage approach.

In the scope of this chapter, we introduced a novel end-to-end system, which is capable of providing these capabilities on pocket drones. Pocket drones' limited payload and computational power, complicates the required position estimation. The proposed system uses distance information, gathered by a light weight UWB-module, in combination with velocity measurements to tackle this issue. The position estimation is handled by a particle filter in combinational with small recurrent neural network. Experiments showed that the combination outperforms the traditional particle filter approaches in terms of converging speed, robustness and computational overhead.

The navigation and collision avoidance is accomplished by a Deep-Q Network DQN, applied to estimate the underlying Q-function in a reinforcement setting. These networks are commonalty used in computer games and other simulated environment. During the course of this chapter, we show that we are able to transfer the learned policy from a simple simulation environment, to the real robot platform and provide the required navigation and collision avoidance strategy. The developed hardware and software is publicly available under the repositories described in the Appendix B.

Chapter 8

Conclusions and Future Work

To conclude, we can divide the thesis in two separate parts, which ultimately can be combined to solve coordination and coverage issues in light weight multi-robot systems.

In the first part of the thesis, we draw inspiration from behaviours and strategies seen in the social insect colonies, specifically bees and ants. Classical centralised multi-robot approaches require expensive sensors to provide the required global state representation and complex computations to handle the massive state space, occurring in large groups of robots. An individual social insect employs only local information and simple strategies, which emerge over the large number of cooperating agents to complex task solvers. The two proposed algorithms, i.e. of BeePCo (Bee Pheromone Coverage) and HybaCo (Hybrid Bee and Ant-pheromone Coverage), are based on the pheromone communication of insects and combine them with a simple rule based system, to provide efficient and robust coverage approaches. Where BeePCo relies exclusively on pheromone propagation observed in bee-hives, HybaCo combines this approach with the passive commutation of ants, to improve the overall performance of the system. Over the course of this thesis we examined BeePCo's and HybaCo's strengths and weaknesses w.r.t. sensor coverage.

Both previous mentioned approaches show a high scalability and robustness in all evaluated scenarios, but their application is restricted to controlled environments e.g. simulation or laboratories. Certain aspects, like for example the artificial pheromone generation and sensing required in HybaCo, are currently researched and will extend the range of application in future work.

BeePCo relies on local communication and sensing between the cooperating agents. Communication based coordination principles for instance BeePCo are feasible with modern day technology, but are still facing challenges on the light weight robot platforms, which most swarm robot techniques rely on. In the context of BeePCo, this is the relative localisation of their neighbouring partners.

The second part of the thesis proposes a complete end-to-end system, tackling these challenges and applies the developed technology on the aerial light weight platform, known as pocket drones. The proposed system uses radio-wave distance calculation in combination with neural networks and particle filters to provide the necessary relative positions estimation. Since aerial platforms are vulnerable to collisions, we employ the

capabilities of an additional neural network (DQN) to provide low-level navigation and collision avoidance. The simple and deterministic structure of the networks, allow for a fast calculation with low computational overhead, necessary for the application on light-weight platforms. These combined abilities then allow for a straightforward transition of ground-base coordination principles (e.g. BeePCo) to the proposed aerial system.

8.1 Contributions

The main contribution of this thesis can be summarised as follows:

Chapter 4 presents the novel coverage approach BeePCo (Bee Pheromone Coverage). Inspired by the pheromone propagation, it is able to distribute and weight important position information, to allow for a simple and effective coverage of the environment. Conducted experiments highlight BeePCo's ability to achieve a high sensor coverage and converging rate, compared with other pheromone coverage principles.

Chapter 5 introduces HybaCo (Hybrid Bee and Ant Coverage) a novel bee- and ant-based pheromone coverage approach. HybaCo combines the virtual message strategy, proposed in BeePCo, with the passive ant pheromone communication observed in ant-colonies and introduced in previous work [83]. By utilising the information, obtain from both communication principles, with a simple rule-base movement behaviour, HybaCo is able to provide an effective continuous coverage (sweeping coverage). Experiments show that HybaCo combines the strengths and exceeds the performances, of both underlying approaches.

Chapter 6 presents a case study investigating the influence of the environment shape, on the performance of the proposed algorithms BeePCo and HybaCo. It shows that the maximal area covered is only marginally affected by the environment, where the time required to reach the maximal coverage increases in environments with confined and hidden spaces. These findings are valuable to determine BeePCo's and HybaCo's deployment in the future.

Chapter 7 presents a complete end-to-end system, allowing the transition of ground based coordination approaches, e.g. BeePCo, to aerial applications on pocket drones. The system utilised Ultra-Wide-Band (UWB) distance calculation, particle-filter and neural networks (RNN and DQN) to provide relative position estimation, navigation and collision avoidance capabilities. The simple and deterministic structure of the neural networks, allow a fast calculation with low computational overhead, necessary for the application on light-weight platforms.

8.2 Future Work

This section discusses limitations and use-cases, which are beneficial to explore in the future. As mentioned before the system proposed in Chapter 7 is designed to provide relative pose-estimation, navigation and collision avoidance behaviour to extend the use-case of ground based coordination principles to aerial applications. Therefore, the implementation of BeePCo on this system is one of the next steps, for BeePCo's application in real-world scenarios. BeePCo relies on direct communication, to propagate virtual pheromone messages through the multi-robot system. Since the UWB-distance calculation is realised by exchanging messages between neighbouring devices, we can implement BeePCo's propagation by integrating the required information into the range-messages.

Most of the restrictions of the pocket drone system (Chapter 7) are hardware based. The required velocity information is currently provided by a motion capture system. In order to allow an application outside the tracking system, we need to supply the drones with a sensor capable of measuring linear movement e.g. visual flow (see Section 7.2.2). The second sensor related limitation, is the fact that the UWB-module is able to provide distances information to drones, but not to walls or other solid obstacle. This makes the drones "blind" in this regard, and can cause collisions with the environment. One valuable option to provide this data is the employment of light distance sensors e.g. sonar. In order to utilise additional sensor information, in the movement policy, provided by the Deep-Q Network (DQN), we have to include this information in the simulator, input state and reward function and simply retrain the network.

8.3 Publications

The material from Chapter 4 is presented in publications 1 and 2. Material from Chapter 5 is featured in publication 3. The Material from Chapter 6 is illustrated in publication 4 and the material shown in Chapter 7 is presented in publication 5.

[1] **B. Broecker**, I. Caliskanelli, K. Tuyls, E. Sklar. A Bee Pheromone Signalling Approach. In: Artificial Life and Intelligent Agents (ALIA), 2014 [17]

[2] B. Ranjbar-Sahraei and K. Tuyls and I. Caliskanelli and **B. Broecker** and D. Claes and S. Alers and G. Weiss. Bio-inspired multi-robot systems, pp. 273-299, Woodhead Publishing, 2015 [80]

[3] **B. Broecker**, I. Caliskanelli, K. Tuyls, E. Sklar, D. Hennes. Social Insect-Inspired Multi-Robot Coverage In: Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2015 [15]

[4] **B. Broecker**, I. Caliskanelli, K. Tuyls, E. Sklar, D. Hennes. Hybrid Insect-Inspired Multi-Robot Coverage in Complex Environments. In: Towards Autonomous Robotic Systems (TAROS), 2015 [16]

[5] **B. Broecker**, K. Tuyls, J. Butterworth Distance-based multi-robot coordination on pocket drones, Submitted.

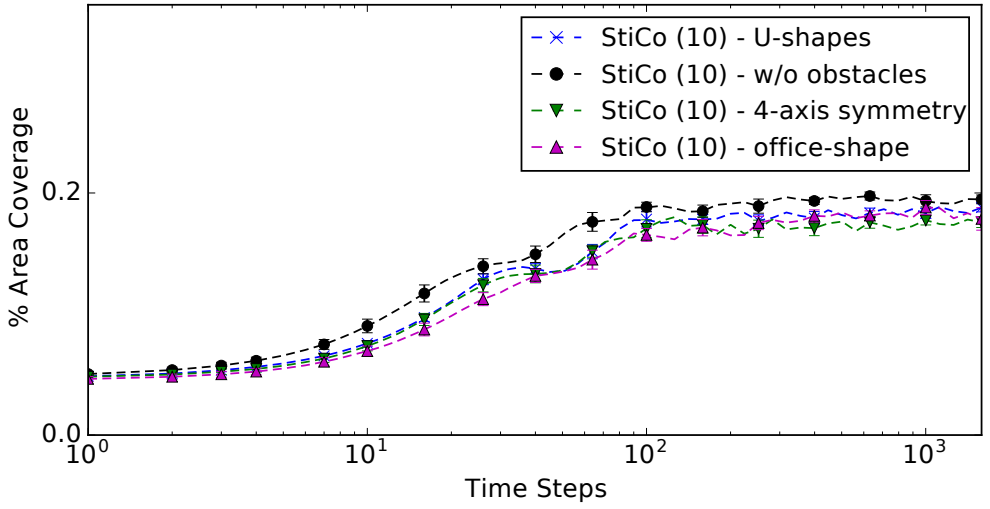
Appendix A

Extended Experiments Chapter 6

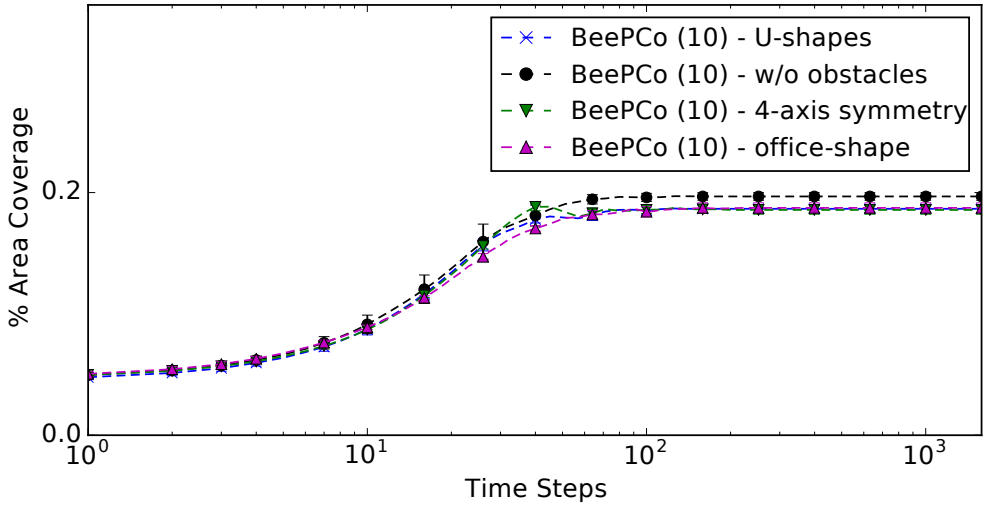
The following appendix contains the full set of plots, generated during the experiments described in Chapter 6. Section A.1 includes all coverage related figures and Section A.2 incorporates all plots related to the algorithms distribution over time.

A.1 Coverage

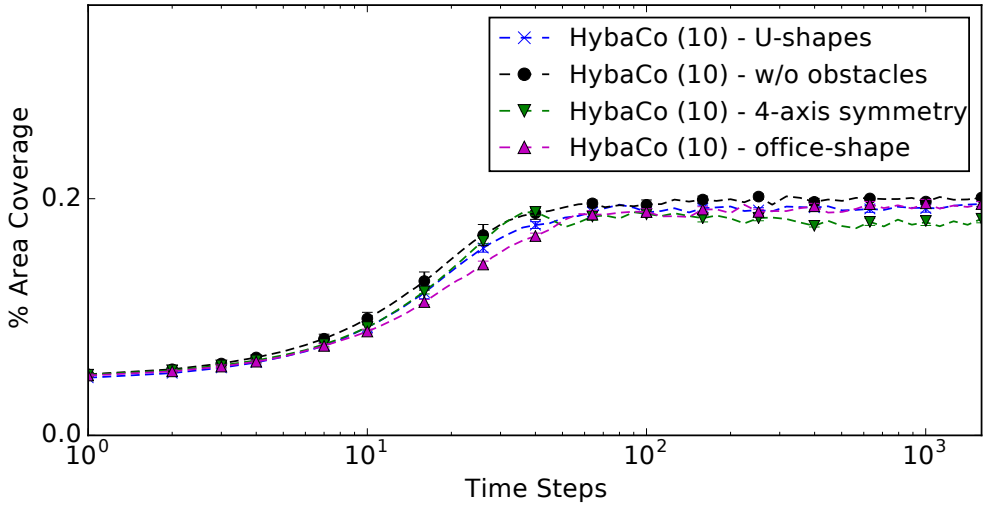
The follow pages show all figures related to the coverage of BeePCo, HybaCo and StiCo conducted during the experiments described in Chapter 6. Figures A.1- A.4 compares each algorithm against its own performance in the different environments (Section 6.1) with respect to the number of robots. Figures A.5- A.8 compare the performances between the algorithms in respect to the environment and the robot count.



(A) StiCo

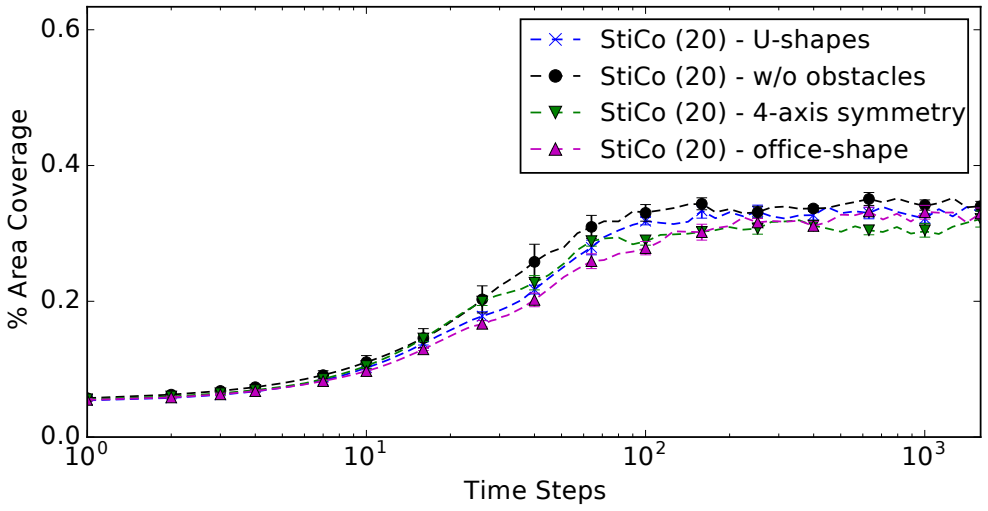


(B) BeePCo

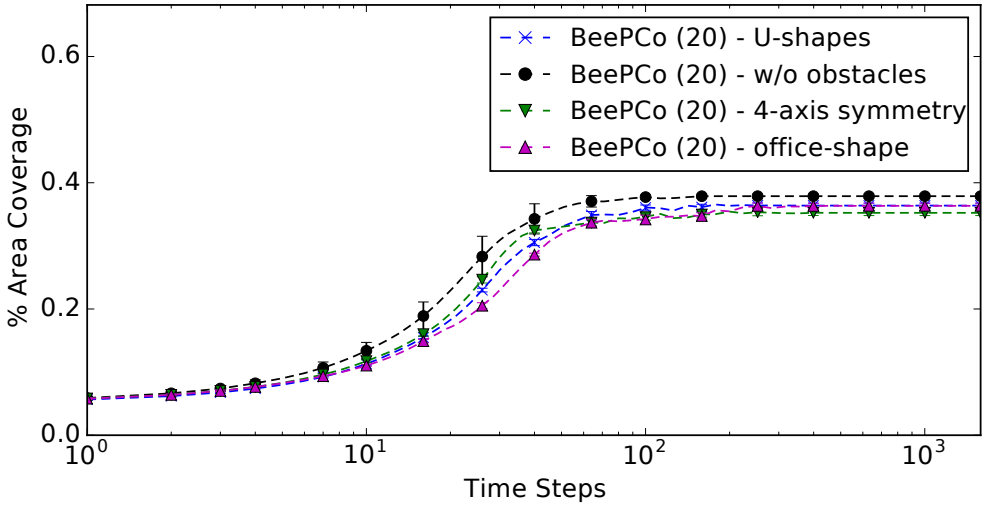


(C) HybaCo

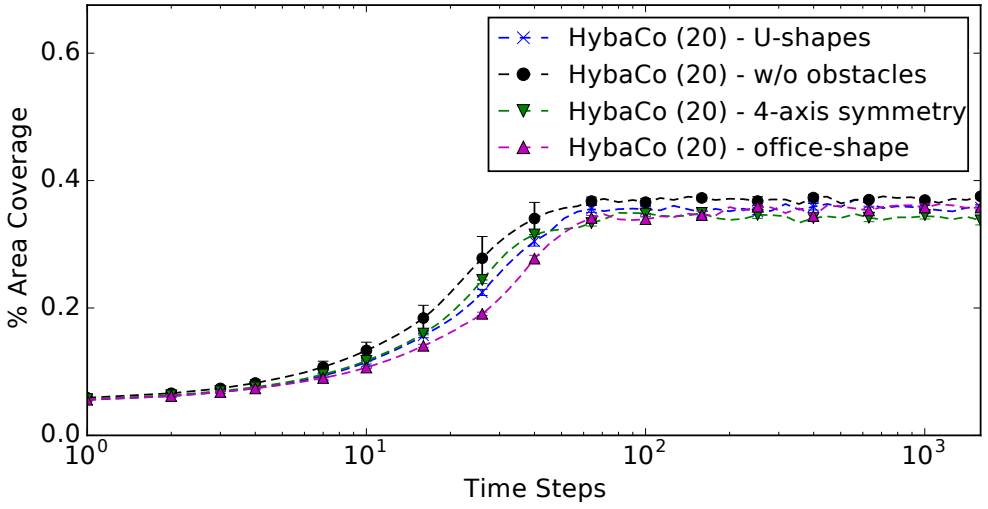
FIGURE A.1: Compares the achieved area coverage between the environment, using 10 robots.



(A) StiCo

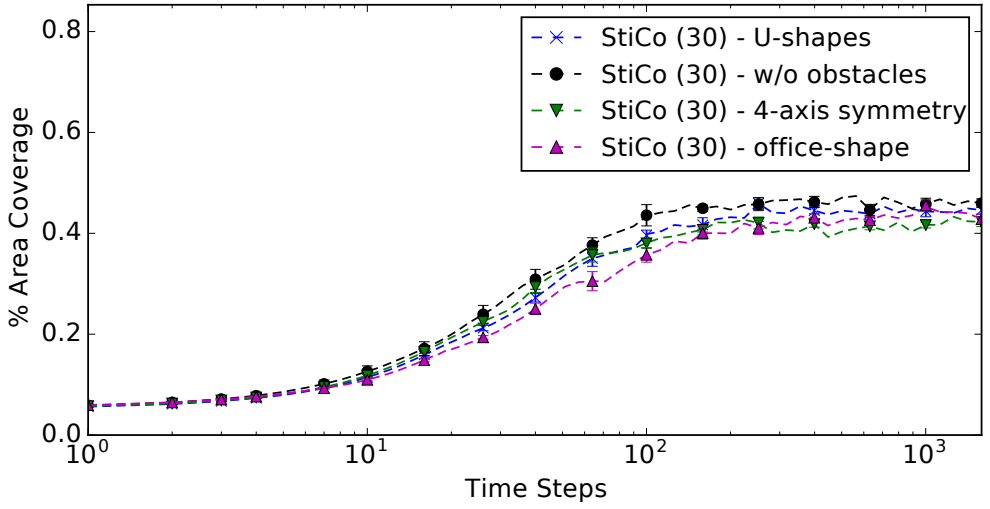


(B) BeePCo

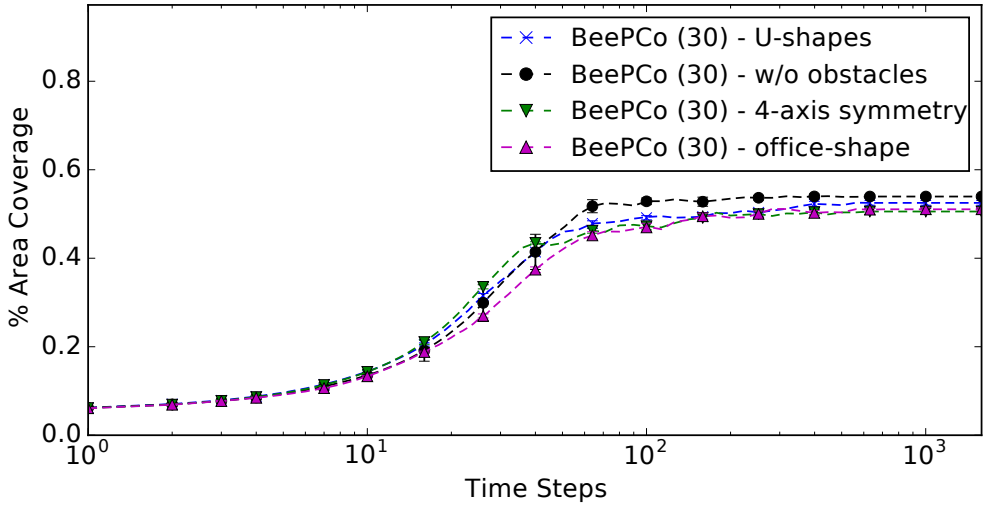


(C) HybaCo

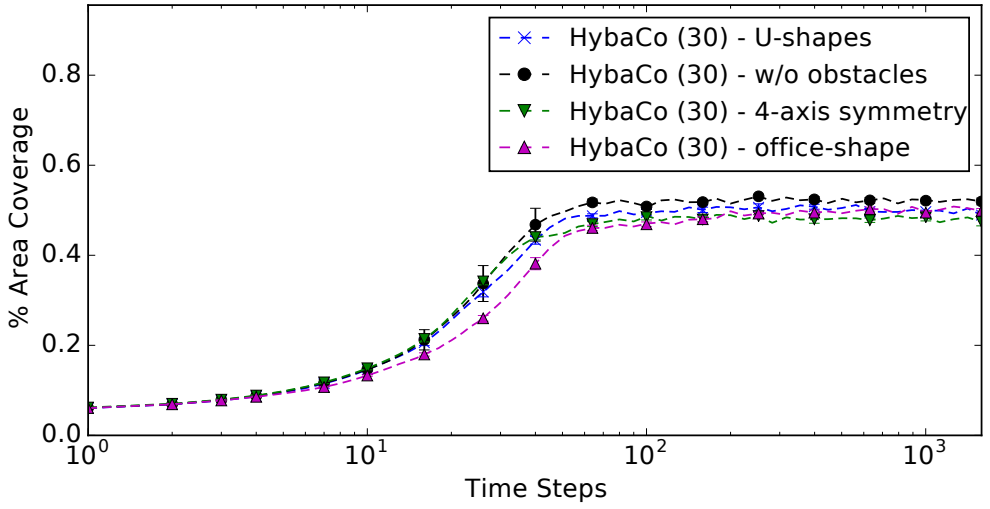
FIGURE A.2: Compares the achieved area coverage between the environment, using 20 robots.



(A) StiCo

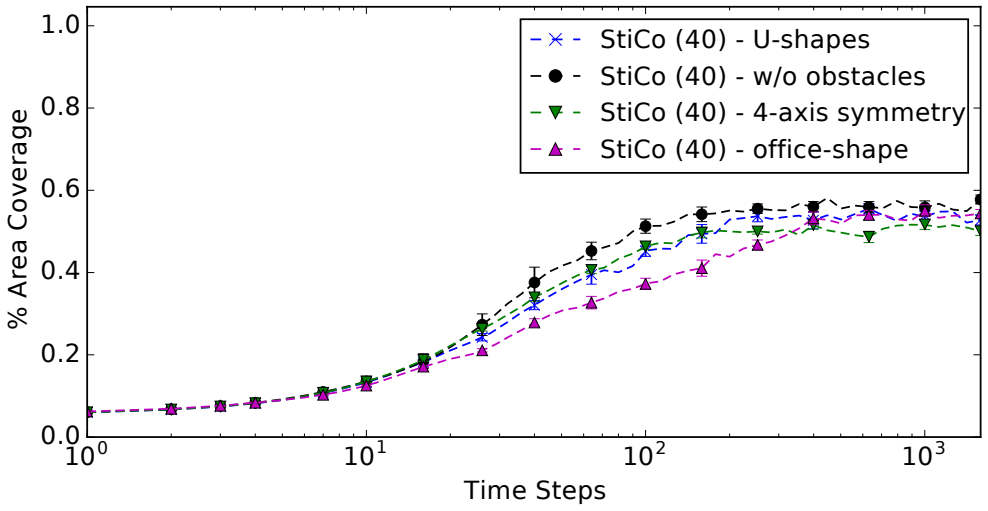


(B) BeePCo

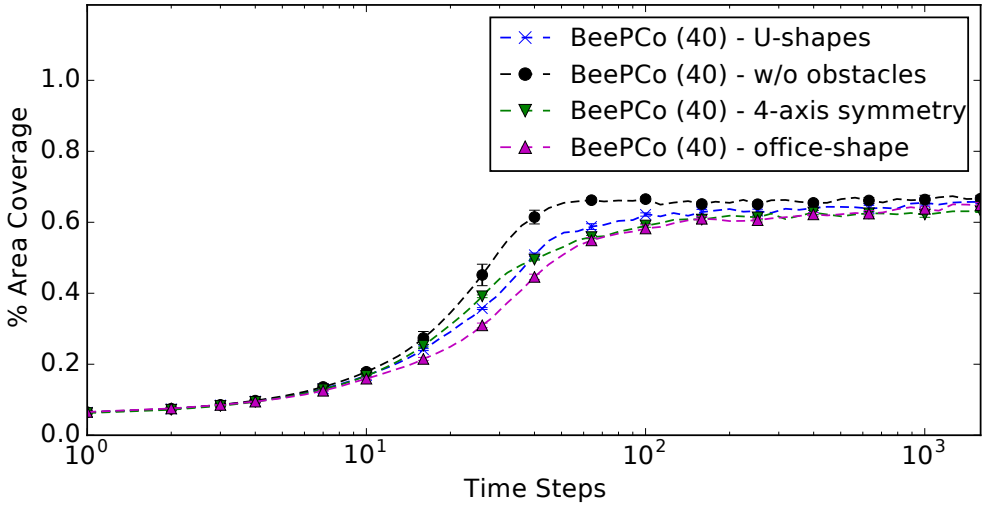


(C) HybaCo

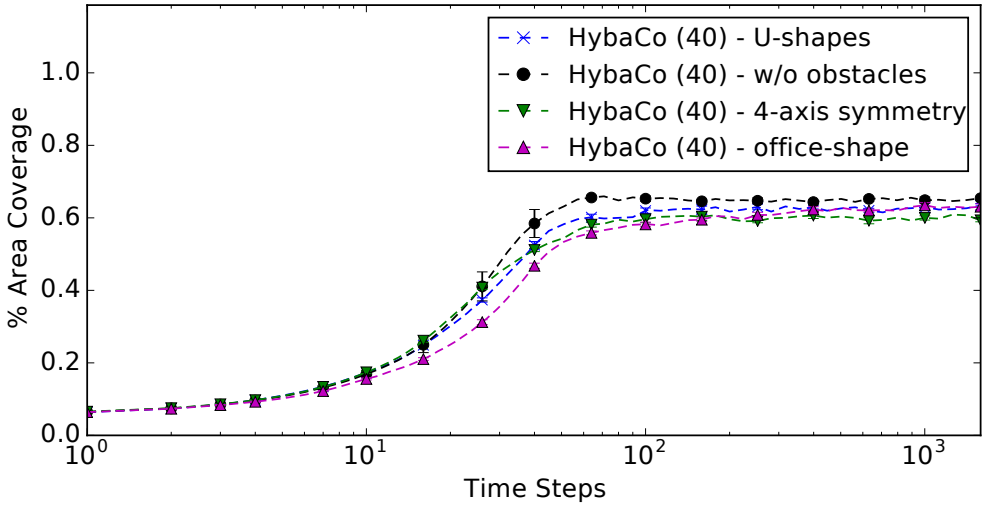
FIGURE A.3: Compares the achieved area coverage between the environment, using 30 robots.



(A) StiCo

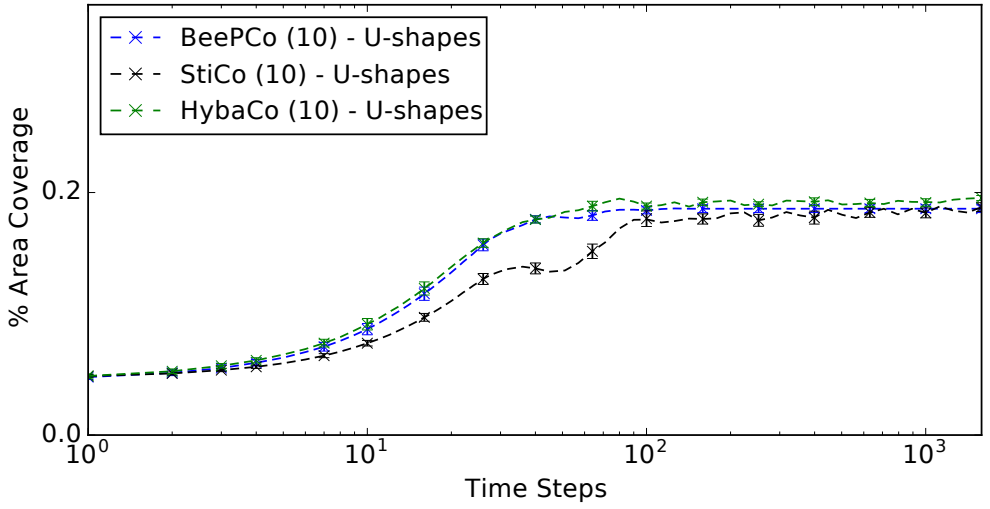


(B) BeePCo

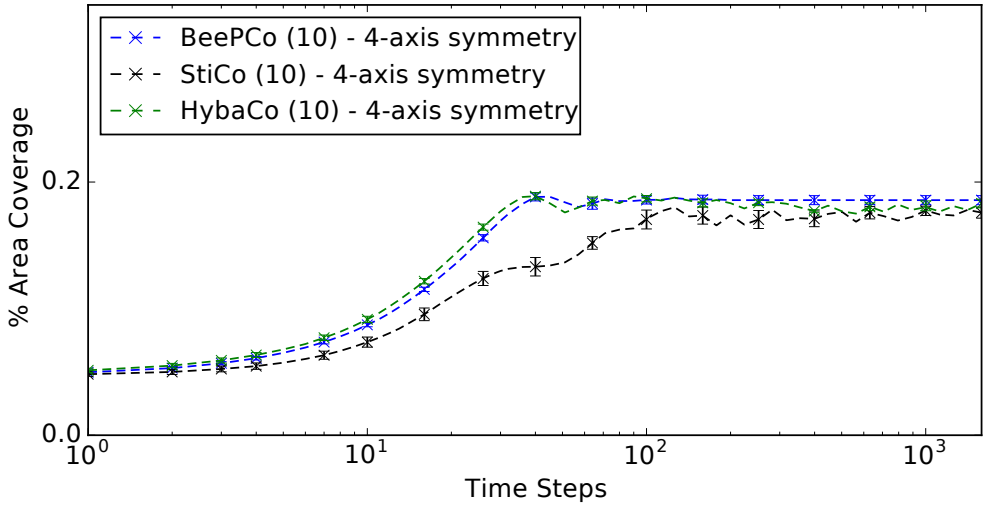


(C) HybaCo

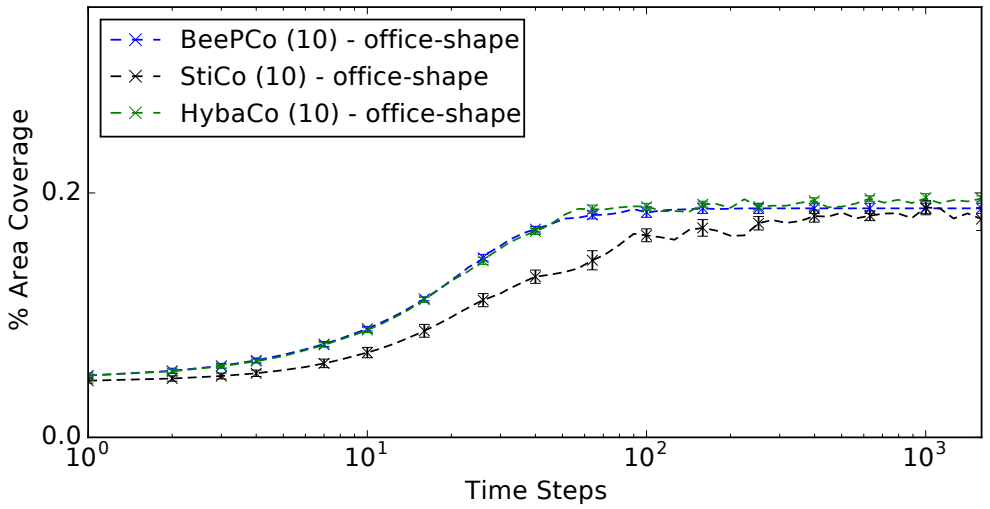
FIGURE A.4: Compares the achieved area coverage between the environment, using 40 robots.



(A) U-shapes

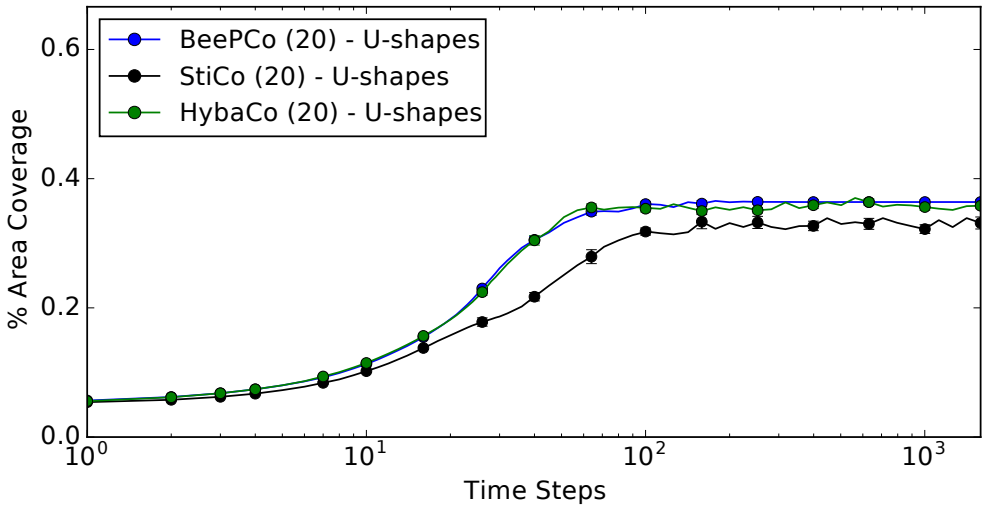


(B) 4-axis symmetric

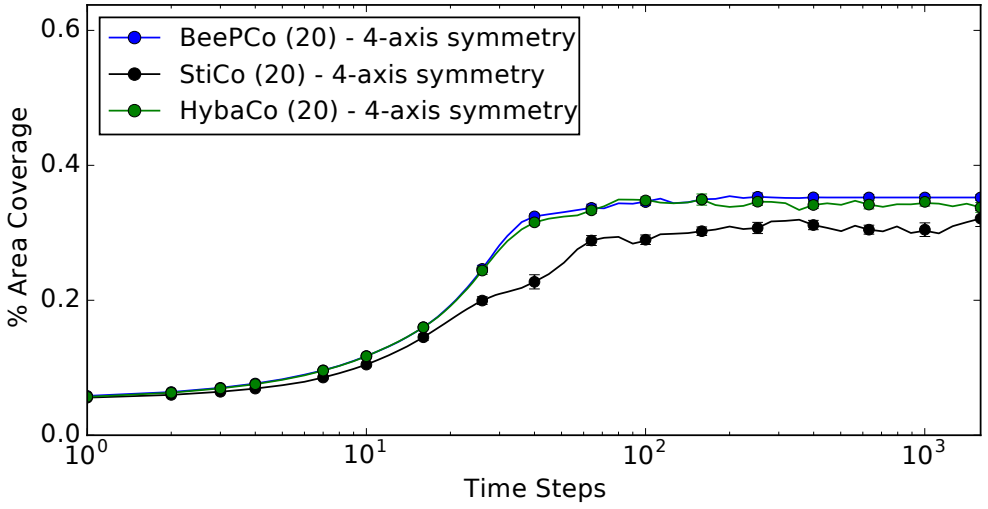


(C) floor-plan

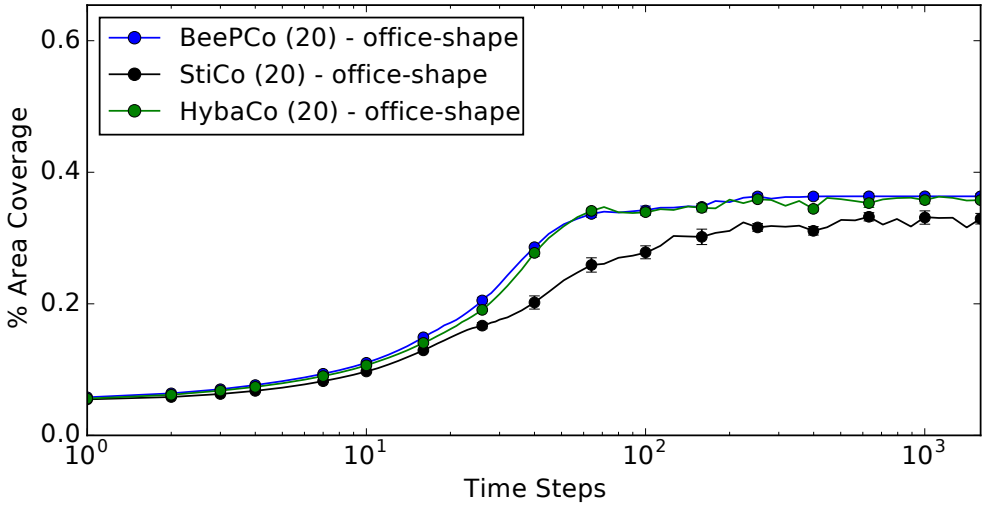
FIGURE A.5: Compares the achieved area coverage between BeePCo, HybaCo and StiCo, using 10 robots.



(A) U-shapes

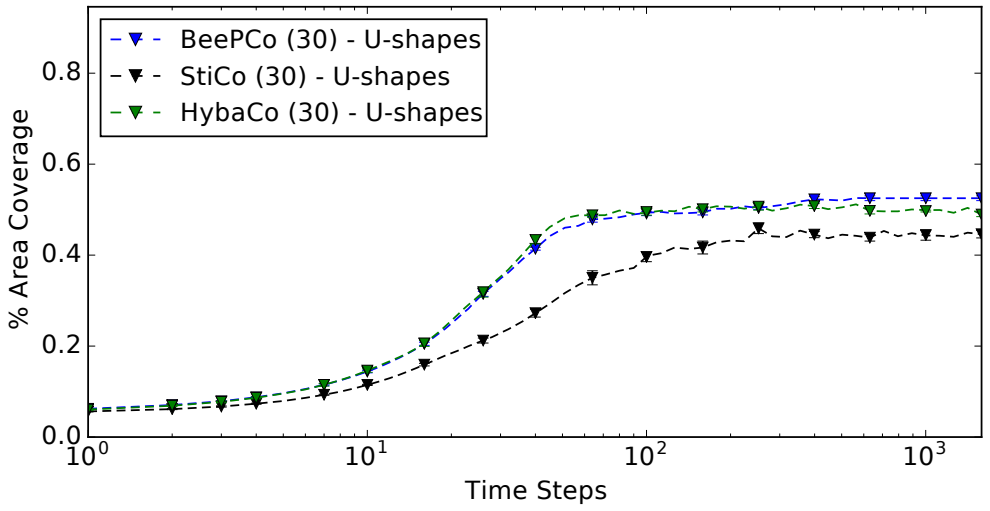


(B) 4-axis symmetric

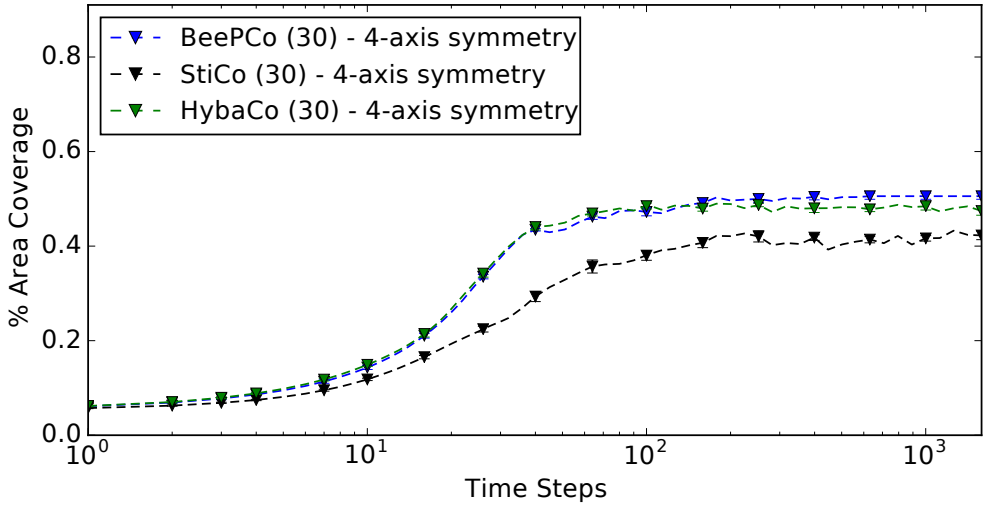


(C) floor-plan

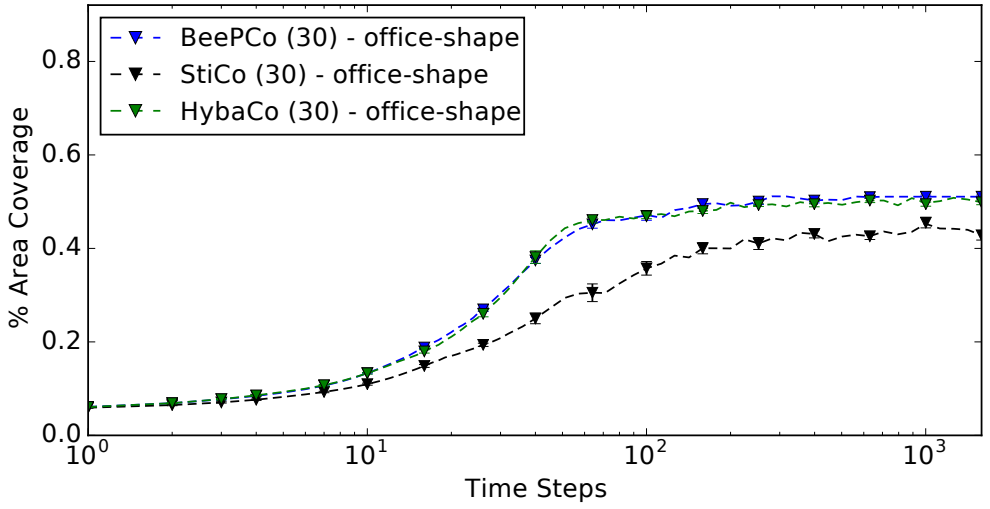
FIGURE A.6: Compares the achieved area coverage between BeePCo, HybaCo and StiCo, using 20 robots.



(A) U-shapes

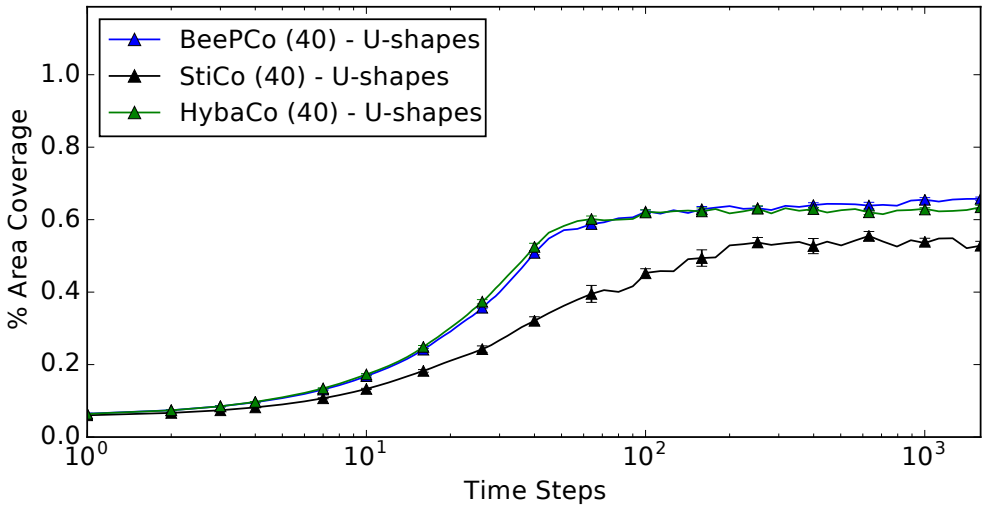


(B) 4-axis symmetric

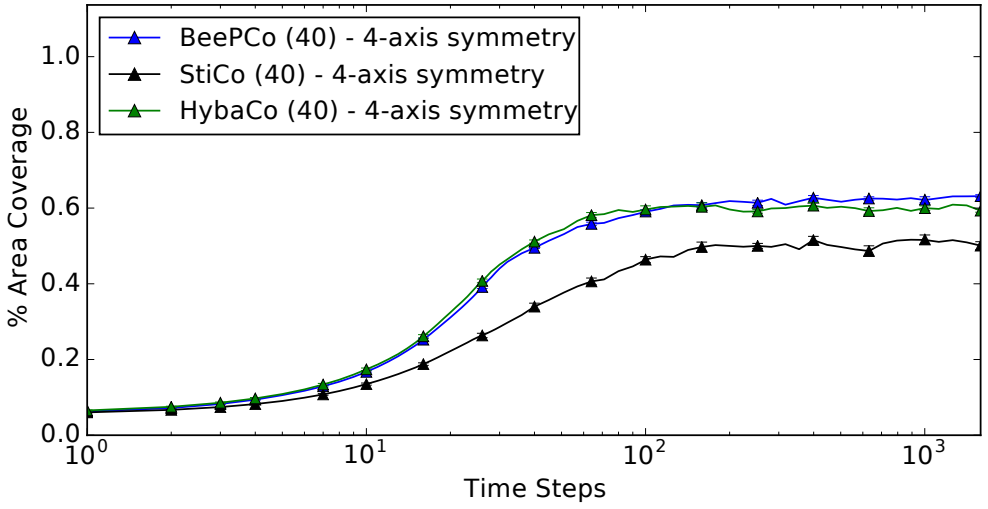


(C) floor-plan

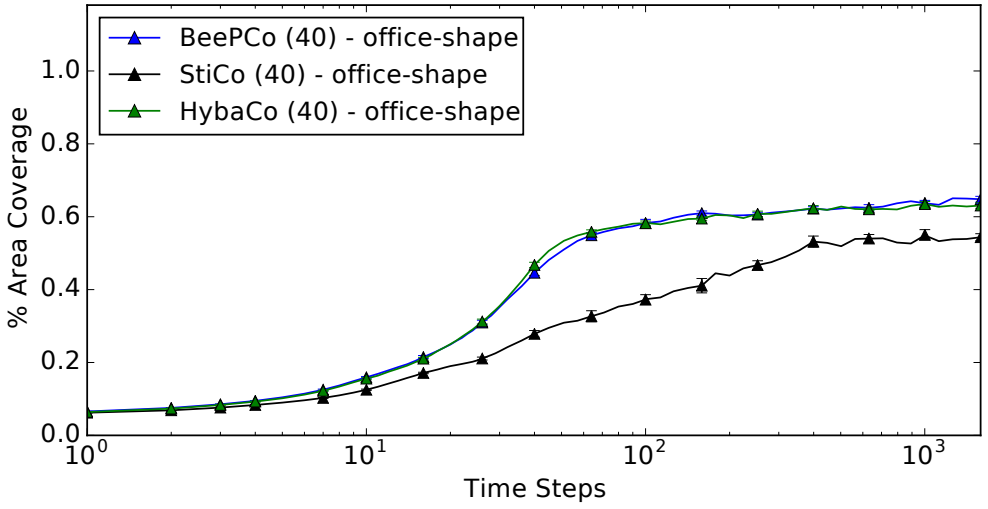
FIGURE A.7: Compares the achieved area coverage between BeePCo, HybaCo and StiCo, using 30 robots.



(A) U-shapes



(B) 4-axis symmetric

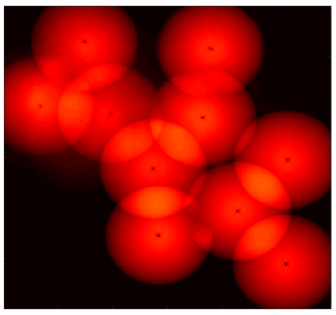


(C) floor-plan

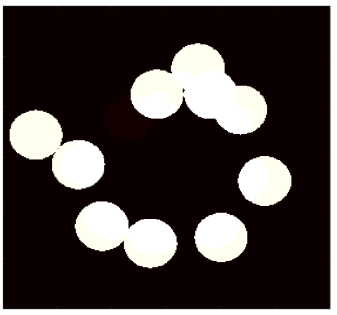
FIGURE A.8: Compares the achieved area coverage between BeePCo, HybaCo and StiCo, using 40 robots.

A.2 Distribution over time

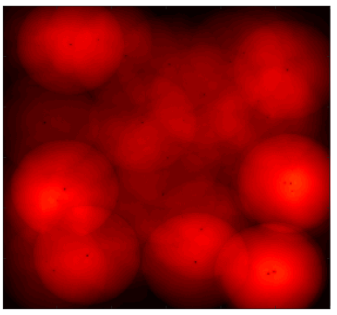
The follow pages show all figures related to the distribution over time conducted during the experiments described in Chapter 6. Figure A.9 - A.12 illustrate the distributions in the heap-map visualisation (introduced in Section 4.4) for BeePCo, HybaCo and StiCo in all employed environments and robot quantities.



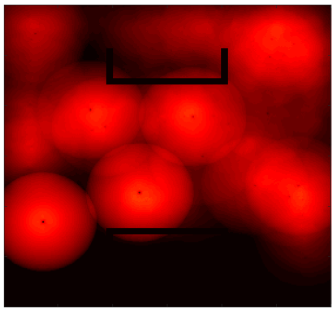
(A) StiCo - Environment without obstacles



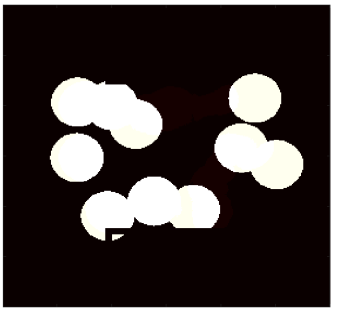
(B) BeePCo - Environment without obstacles



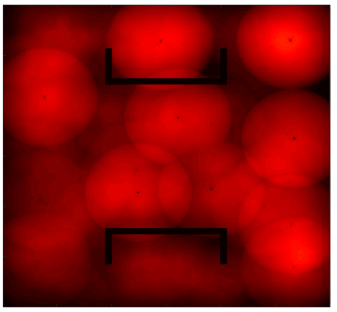
(C) HybaCo - Environment without obstacles



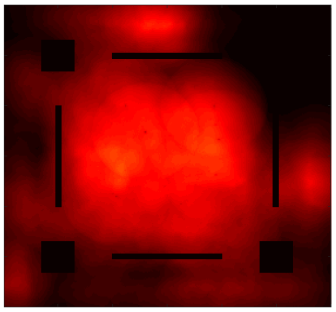
(D) StiCo - U-shape environment



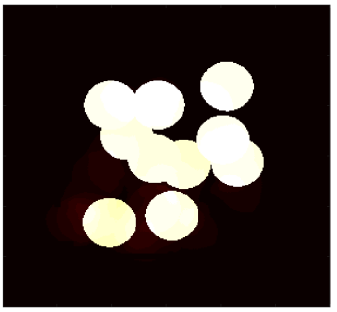
(E) BeePCo - U-shape environment



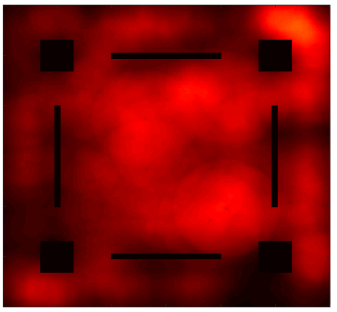
(F) HybaCo - U-shape environment



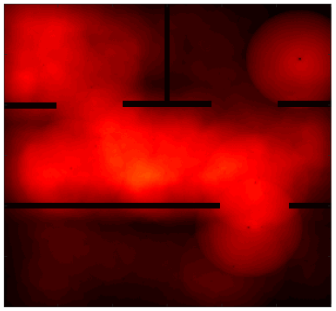
(G) StiCo - 4-axis symmetric environment



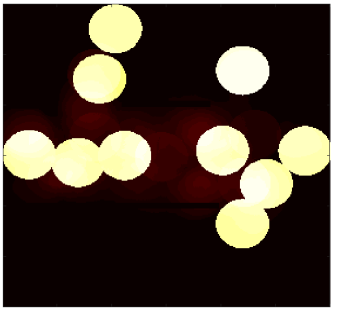
(H) BeePCo - 4-axis symmetric environment



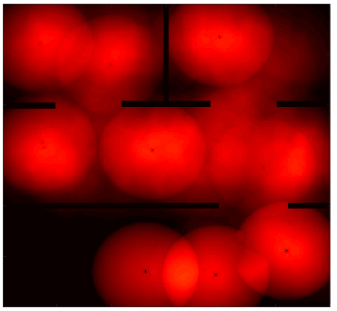
(I) HybaCo - 4-axis symmetric environment



(J) StiCo - office environment

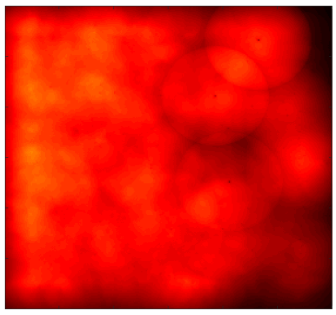


(K) BeePCo - office environment

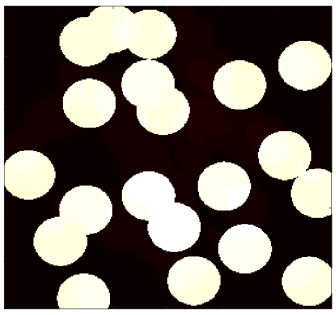


(L) HybaCo - office environment

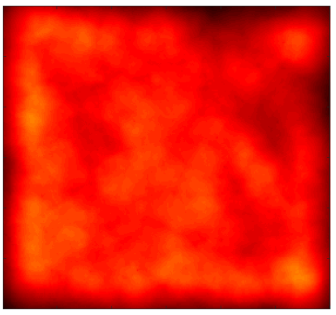
FIGURE A.9: The distribution over time, in the different arenas, using 10 robots. The coverage is illustrated in a gradient color scheme. A cell is coloured white, if it is covered 100% of the run time and coloured black for a 0% coverage. The intermediate percentages are colour accordingly to the scale at the top of the figure.



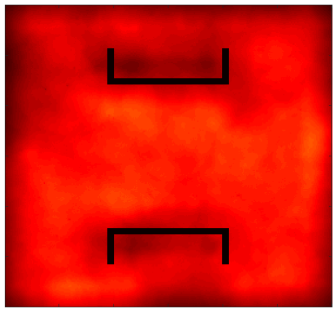
(A) StiCo - Environment without obstacles



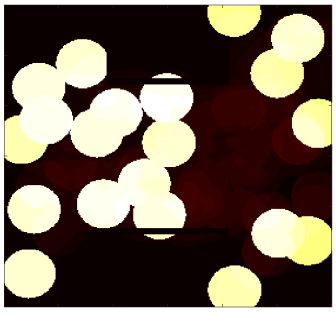
(B) BeePCo - Environment without obstacles



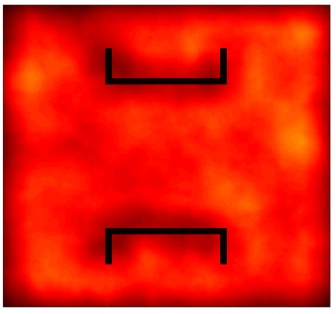
(C) HybaCo - Environment without obstacles



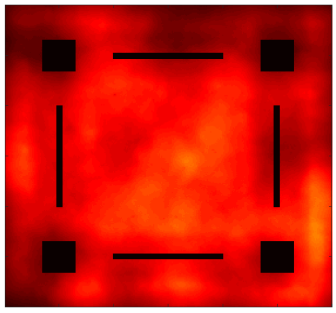
(D) StiCo - U-shape environment



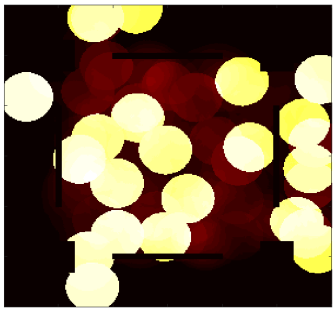
(E) BeePCo - U-shape environment



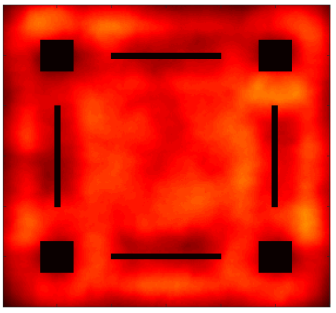
(F) HybaCo - U-shape environment



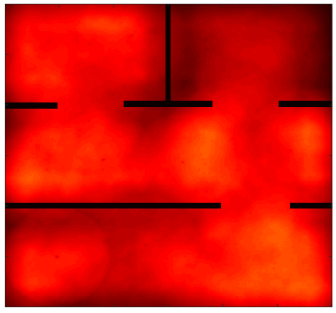
(G) StiCo - 4-axis symmetric environment



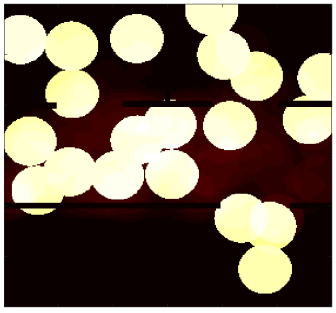
(H) BeePCo - 4-axis symmetric environment



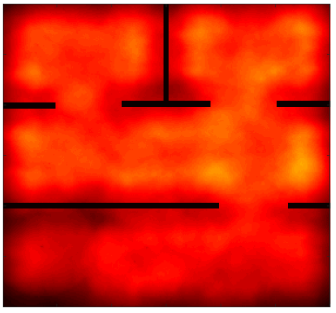
(I) HybaCo - 4-axis symmetric environment



(J) StiCo - office environment

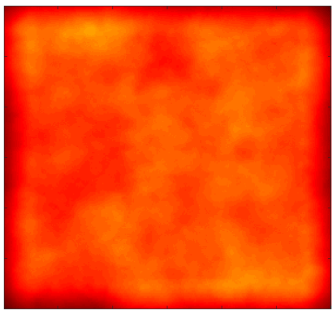


(K) BeePCo - office environment

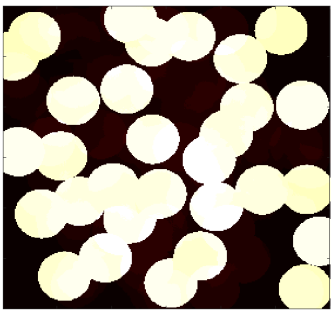


(L) HybaCo - office environment

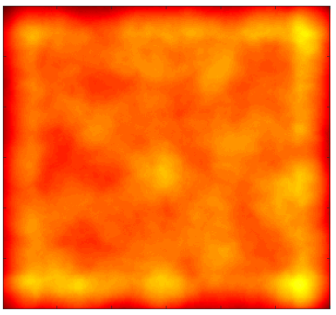
FIGURE A.10: The distribution over time, in the different arenas, using 20 robots. The coverage is illustrated in a gradient color scheme. A cell is coloured white, if it is covered 100% of the run time and coloured black for a 0% coverage. The intermediate percentages are coloured accordingly to the scale at the top of the figure.



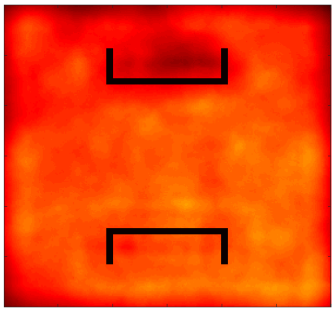
(A) StiCo - Environment without obstacles



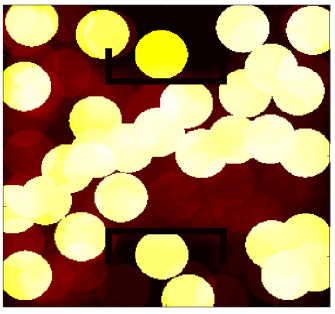
(B) BeePCo - Environment without obstacles



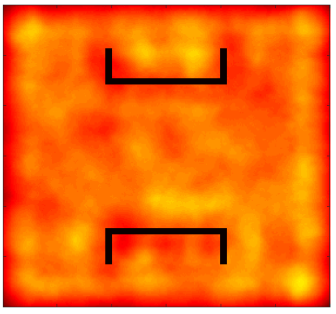
(C) HybaCo - Environment without obstacles



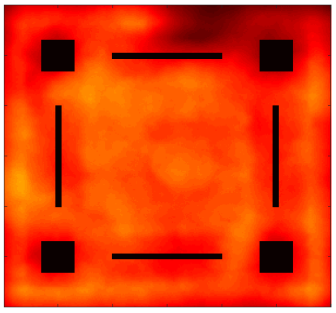
(D) StiCo - U-shape environment



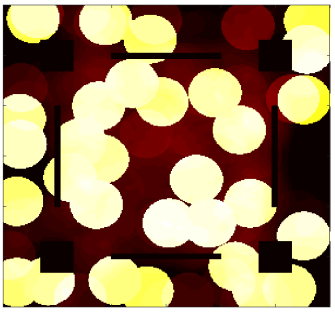
(E) BeePCo - U-shape environment



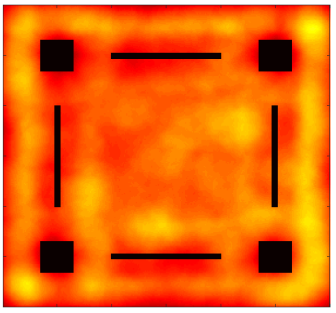
(F) HybaCo - U-shape environment



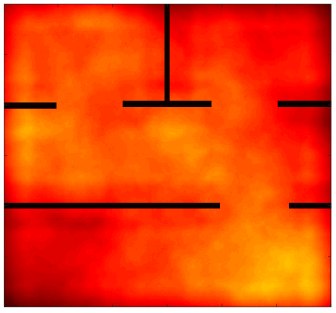
(G) StiCo - 4-axis symmetric environment



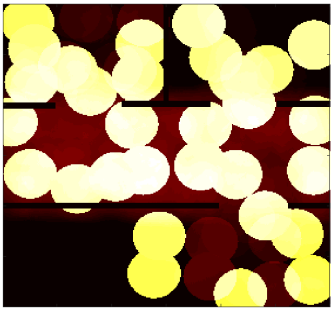
(H) BeePCo - 4-axis symmetric environment



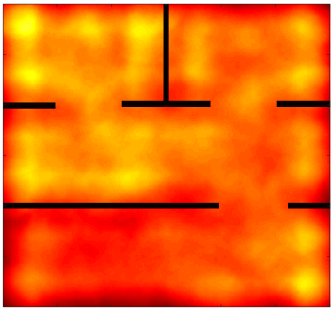
(I) HybaCo - 4-axis symmetric environment



(J) StiCo - office environment

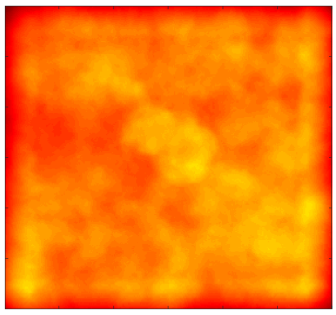


(K) BeePCo - office environment

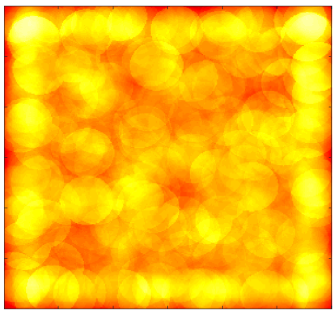


(L) HybaCo - office environment

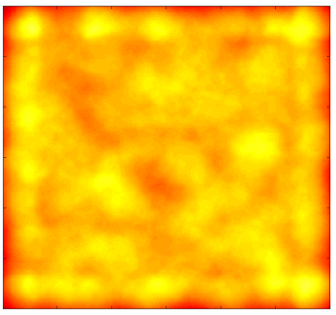
FIGURE A.11: The distribution over time, in the different arenas, using 30 robots. The coverage is illustrated in a gradient color scheme. A cell is coloured white, if it is covered 100% of the run time and coloured black for a 0% coverage. The intermediate percentages are coloured accordingly to the scale at the top of the figure.



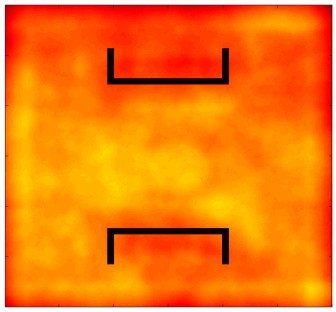
(A) StiCo - Environment without obstacles



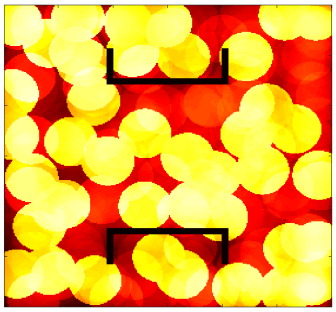
(B) BeePCo - Environment without obstacles



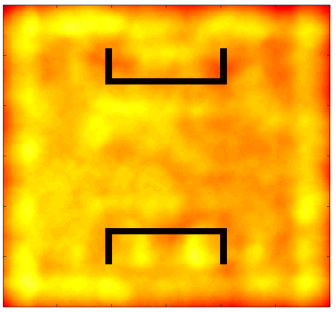
(C) HybaCo - Environment without obstacles



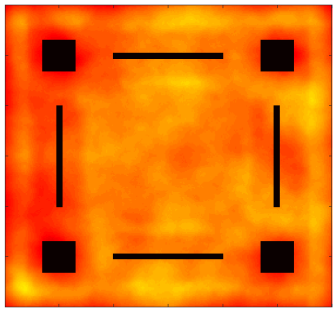
(D) StiCo - U-shape environment



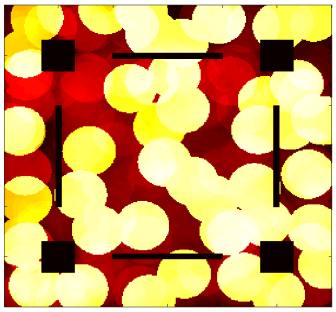
(E) BeePCo - U-shape environment



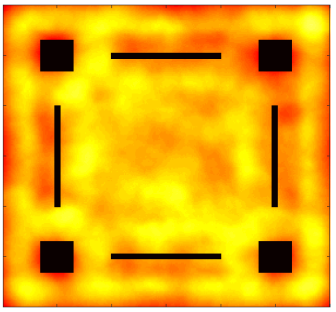
(F) HybaCo - U-shape environment



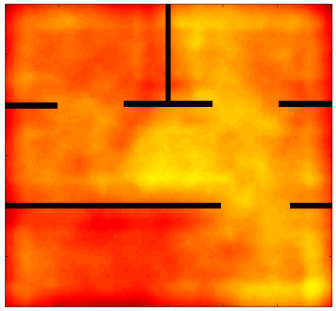
(G) StiCo - 4-axis symmetric environment



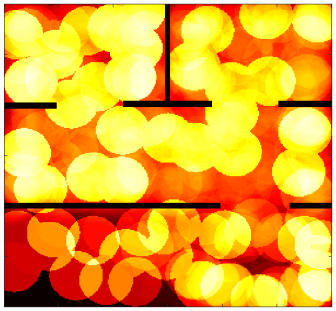
(H) BeePCo - 4-axis symmetric environment



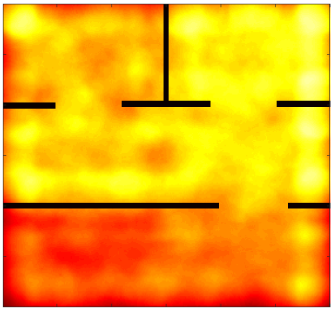
(I) HybaCo - 4-axis symmetric environment



(J) StiCo - office environment



(K) BeePCo - office environment



(L) HybaCo - office environment

FIGURE A.12: The distribution over time, in the different arenas, using 40 robots. The coverage is illustrated in a gradient color scheme. A cell is coloured white, if it is covered 100% of the run time and coloured black for a 0% coverage. The intermediate percentages are coloured accordingly to the scale at the top of the figure.

Appendix B

Hardware Design and Software/Hardware Repositories

This section gives a short overview over the hardware design, the required component and the required software repositories. All mentioned repositories include a basic description for they usage, if needed. In order to connect the DWM1000 (UWB module) to the crazyflie drone, we connect the SPI-interfaces of both devices together. Since the regulator is not able to provide enough current to run the module, we add an additional 3.3v regulator (TPS731). Figure B.1 illustrates the complete schematic with all required capacitors and resistors, all components are additionally listed in Table B.1. The pcb-design is shown in Figure B.2. All files required to replicate the hardware are downloadable under: `https://github.com/bbroecker/dmrc_hardware`. The altered crazyflie driver including the developed communication protocol, can be found under the following repository `https://github.com/bbroecker/dbmr-crazyflie-firmware`. The additional crazyflie contoller can be found under `https://github.com/bbroecker/sl_crazyflie_dwm1000` and the software to train and test the neural network are stored under: `https://github.com/bbroecker/dmrc_training`.

Schematic ID	**Description**	**Value**
R4	Resistor	100K
C1	Capacitor	10uF
C2	Capacitor	10nF
C3	Capacitor	10nF
TPS731	Regulator	TPS731 3.3v regulator

Table B.1: Used components and their specification.

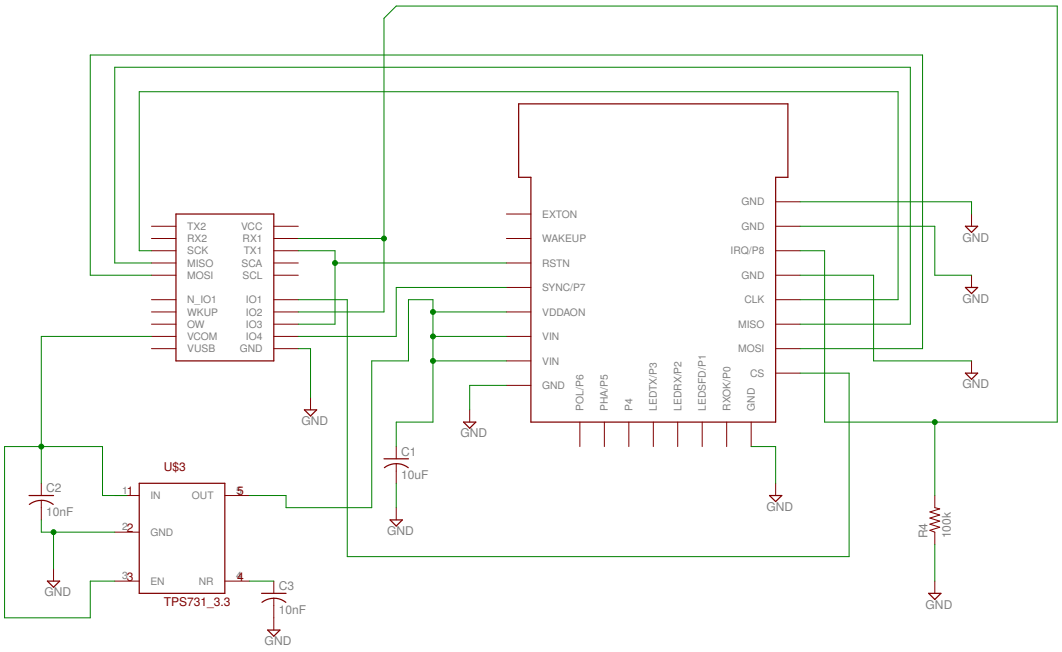


FIGURE B.1: Layout of the board. Files can be found under: `https://github.com/bbroecker/dmrc_hardware`

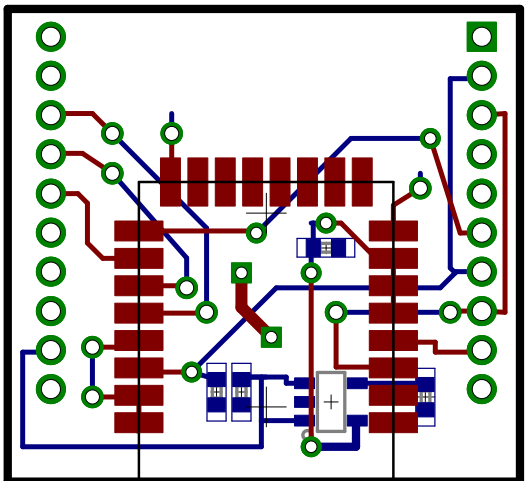


FIGURE B.2: Schematic showing the connections required to combine the crazyflie with the DWM1000 module. (See repository at `https://github.com/bbroecker/dmrc_hardware`)

Bibliography

[1] Mazda Ahmadi and Peter Stone, *A multi-robot system for continuous area sweeping tasks*, Proc. of the IEEE Int. Conf. on Robotics and Automation, May 2006, pp. 1724–1729.

[2] Bahriye Akay and Dervis Karaboga, *Parameter tuning for the artificial bee colony algorithm*, pp. 608–619, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[3] ———, *Parameter tuning for the artificial bee colony algorithm*, Computational Collective Intelligence. Semantic Web, Social Networks and Multiagent Systems (NgocThanh Nguyen, Ryszard Kowalczyk, and Shyi-Ming Chen, eds.), Lecture Notes in Computer Science, vol. 5796, Springer Berlin Heidelberg, 2009, pp. 608–619 (English).

[4] Isaac Amundson, Janos Sallai, Xenofon Koutsoukos, Akos Ledeczi, and Miklos Maroti, *Rf angle of arrival;based node localisation*, Int. J. Sen. Netw. **9** (2011), no. 3/4, 209–224.

[5] Tom M Apostol, *Mathematical analysis; 2nd ed.*, Addison-Wesley Series in Mathematics, Addison-Wesley, Reading, MA, 1974.

[6] Nando de Freitas & Neil Gordon Arnaud Doucet (ed.), *Sequential Monte Carlo methods in practice*, Springer, 2001.

[7] Farshad Arvin, Tomás Krajník, Ali Emre Turgut, and Shigang Yue, *CosΦ: Artificial pheromone system for robotic swarms research*, 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015, Hamburg, Germany, September 28 - October 2, 2015, 2015, pp. 407–412.

[8] T. Balch and M. Hybinette, *Behavior-based coordination of large-scale robot formations*, MultiAgent Systems, 2000. Proc. 4th Int. Conf. on, 2000, pp. 363–364.

[9] M. Basiri, F. Schill, D. Floreano, and P. Lima, *Audio-based Relative Positioning System for Multiple Micro Air Vehicle Systems*, Robotics: Science and Systems RSS2013, 2013.

[10] Randal Beard, *Quadrotor Dynamics and Control Rev 0.1*, `http://scholarsarchive.byu.edu/cgi/viewcontent.cgi?article=2324&context=facpub`, 2008, [Online; accessed 21-September-2016].

[11] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling, *The arcade learning environment: An evaluation platform for general agents*, CoRR **abs/1207.4708** (2012).

[12] M. Bernas and B. Płaczek, *Energy aware object localization in wireless sensor network based on wi-fi fingerprinting*, pp. 33–42, Springer International Publishing.

[13] P. Boonma and J. Suzuki, *Monsoon: A coevolutionary multiobjective adaptation framework for dynamic wireless sensor networks* (Undetermined).

[14] A. Breitenmoser, M. Schwager, J. C. Metzger, R. Siegwart, and D. Rus, *Voronoi coverage of non-convex environments with a group of networked robots*, Proc. of the International Conference on Robotics and Automation (ICRA 10), May 2010, pp. 4982–4989.

[15] Bastian Broecker, Ipek Caliskanelli, Karl Tuyls, Elizabeth Sklar, and Daniel Hennes, *Social insect-inspired multi-robot coverage*, Proc. of Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS), 2015.

[16] Bastian Broecker, Ipek Caliskanelli, Karl Tuyls, Elizabeth I. Sklar, and Daniel Hennes, *Hybrid insect-inspired multi-robot coverage in complex environments*, Towards Autonomous Robotic Systems - 16th Annual Conference, TAROS 2015, Liverpool, UK, September 8-10, 2015, Proceedings, 2015, pp. 56–68.

[17] I. Caliskanelli, B. Broecker, and K. Tuyls, *Multi-robot coverage: A bee pheromone signalling approach*, Int. Conf. on Artificial Life and Intelligent Agents, 2014.

[18] I. Caliskanelli, J. Harbin, F Indrusiak, L. Polack, P. Mitchell, and D Chesmore, *Runtime optimisation in wsns for load balancing using pheromone signalling*, NESEA, 2012. 3rd IEEE Int. Conf. on, Dec. 2012.

[19] I. Caliskanelli and L. Indrusiak, *Search-based parameter tuning on application-level load balancing for distributed embedded systems*, Proc. of the 11th IEEE/IFIP Int. Conf. on Embedded and Ubiquitous Computing (EUC), 2013.

[20] I. Caliskanelli and L. S. Indrusiak, *Using mobile robotic agents to increase service availability and extend network lifetime on wsrns*, 2014 12th IEEE International Conference on Industrial Informatics (INDIN), July 2014, pp. 388–393.

[21] ———, *Using mobile robotic agents to increase service availability and extend network lifetime on wsrns*, 2014 12th IEEE International Conference on Industrial Informatics (INDIN), July 2014, pp. 388–393.

[22] Ipek Caliskanelli, James Robert Harbin, Leandro Soares Indrusiak, Paul Daniel Mitchell, Fiona A C Polack, and David Chesmore, *Bio-inspired load balancing in large-scale wsns using pheromone signalling*, Int. Journal of Distributed Sensor Networks (2013).

[23] 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** (1987), no. 3, 469–475.

[24] Matthew Collett, Thomas S Collett, Sonja Bisch, and Rüdigerïœ Wehner, *Local and global vectors in desert ant navigation*, Nature **394** (1998), no. 6690, 269–272.

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

[26] N. Correll and A. Martinoli, *Collective inspection of regular structures using a swarm of miniature robots*, Experimental Robotics IX (Jr. Ang, MarceloH. and Oussama Khatib, eds.), Springer Tracts in Advanced Robotics, vol. 21, Springer Berlin Heidelberg, 2006, pp. 375–386 (English).

[27] J. Cortes, S. Martinez, T. Karatas, and F. Bullo, *Coverage control for mobile sensing networks*, Robotics and Automation, IEEE Transactions on **20** (2004), no. 2, 243–255.

[28] M. Dorigo, *Optimization, learning and natural algorithms*, Thesis report, Politecnico di Milano, Italy, 1992.

[29] M. Dorigo, M. Birattari, and T. Stutzle, *Ant colony optimization*, Computational Intelligence Magazine, IEEE **1** (2006), no. 4, 28 –39.

[30] ———, *Ant colony optimization: Artificial ants as a computational intelligence technique*, Computational Intelligence Magazine, IEEE **1** (2006), no. 4, 28–39.

[31] M. Dorigo and T. Stützle, *Ant colony optimization*, A Bradford book, BRADFORD BOOK, 2004.

[32] Marco Dorigo and Christian Blumb, *Ant colony optimization theory: A survey*, Theoretical Computer Science **344** (2005), 243–278.

[33] Marco Dorigo and Av FD Roosevelt, *Swarm robotics*, Special Issue", Autonomous Robots, Citeseer, 2004.

[34] Stuart Dreyfus, *The numerical solution of variational problems*, Journal of Mathematical Analysis and Applications **5** (1962), no. 1, 30 – 45.

[35] Nasser A. El-Sherbeny, *Vehicle routing with time windows: An overview of exact, heuristic and metaheuristic methods*, Journal of King Saud University - Science **22** (2010), no. 3, 123 – 131.

[36] Y. Elor and A. M. Bruckstein, *Multi-a(ge)nt graph patrolling and partitioning*, 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology, vol. 2, Sept 2009, pp. 52–57.

[37] Aaron Jaech et al., *Domain adaptation of recurrent neural networks for natural language understanding*, CoRR (2016).

[38] B. Ranjbar-Sahraei et al., *Bio-inspired multi-robot systems*, Biomimetic Technologies, 2015, pp. 273 – 299.

[39] D. Hennes et al., *Multi-robot collision avoidance with localization uncertainty*, Proceedings of AAMAS 2012, 2012, pp. 147–154.

[40] Volodymyr Mnih et al., *Playing atari with deep reinforcement learning*, vol. abs/1312.5602, 2013.

[41] Paolo Fiorini and Zvi Shiller, *Motion planning in dynamic environments using velocity obstacles*, The International Journal of Robotics Research **17** (1998), no. 7, 760–772.

[42] E. Folgado, M. Rincn, J.R. lvarez, and J. Mira, *A multi-robot surveillance system simulated in gazebo*, Nature Inspired Problem-Solving Methods in Knowledge Engineering (Jos Mira and JosR. lvarez, eds.), Lecture Notes in Computer Science, vol. 4528, Springer Berlin Heidelberg, 2007, pp. 202–211 (English).

[43] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun, *Monte carlo localization: Efficient position estimation for mobile robots*, Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence (Menlo Park, CA, USA), AAAI '99/IAAI '99, American Association for Artificial Intelligence, 1999, pp. 343–349.

[44] R. Fujisawa, H. Imamura, T. Hashimoto, and F. Matsuno, *Communication using pheromone field for multiple robots*, Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on, Sept 2008, pp. 1391–1396.

[45] Yoav Gabriely and Elon Rimon, *Spanning-tree based coverage of continuous areas by a mobile robot*, Annals of Mathematics and Artificial Intelligence **31** (2001), no. 1-4, 77–98 (English).

[46] Yoav Gabriely and Elon Rimon, *Competitive on-line coverage of grid environments by a mobile robot*, Computational Geometry **24** (2003), no. 3, 197 – 224.

[47] D. W. Gage, *Command control for many-robot systems*, Naval Command control and ocean surveillance centre rtd San Diego CA, 1992.

[48] Arnaud Glad, Olivier Simonin, Olivier Buffet, and François Charpillet, *Theoretical study of ant-based algorithms for multi-agent patrolling*, Proceedings of the 2008 Conference on ECAI 2008: 18th European Conference on Artificial Intelligence (Amsterdam, The Netherlands, The Netherlands), IOS Press, 2008, pp. 626–630.

[49] S. Goss and J.L. Deneubourg, *Harvesting by a group of robots*, Proceedings of the First European Conference on Artificial Life, 1992, pp. 195–204.

[50] R. Gross, M. Bonani, F. Mondada, and M. Dorigo, *Autonomous self-assembly in swarm-bots*, IEEE Transactions on Robotics **22** (2006), no. 6, 1115–1130.

[51] Yi Guo and Zhihua Qu, *Coverage control for a mobile robot patrolling a dynamic and uncertain environment*, Intelligent Control and Automation, 2004. WCICA 2004. Fifth World Congress on, vol. 6, June 2004, pp. 4899–4903 Vol.6.

[52] F. Gustafsson and F. Gunnarsson, *Positioning using time-difference of arrival measurements*, Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03). 2003 IEEE International Conference on, vol. 6, IEEE, April 2003, pp. VI–553–6 vol.6.

[53] Noam Hazon and Gal A. Kaminka, *On redundancy, efficiency, and robustness in coverage for multiple robots*, Robotics and Autonomous Systems **56** (2008), no. 12, 1102 – 1114, Towards Autonomous Robotic Systems 2008: Mobile Robotics in the {UK} 10th British Conference on Mobile Robotics - Towards Autonomous Robotic Systems (TAROS 2007).

[54] Daniel Hennes, Daniel Claes, Wim Meeussen, and Karl Tuyls, *Multi-robot collision avoidance with localization uncertainty*, Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1 (Richland, SC), AAMAS '12, International Foundation for Autonomous Agents and Multiagent Systems, 2012, pp. 147–154.

[55] S. Hochreiter and J. Schmidhuber, *Long short-term memory*, Neural Comput. **9** (1997), no. 8, 1735–1780.

[56] Rosen Ivanov, *Indoor navigation system for visually impaired*, Proceedings of the 11th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing on International Conference on Computer Systems and Technologies (New York, NY, USA), CompSysTech '10, ACM, 2010, pp. 143–149.

[57] Boyoon Jung and GauravS. Sukhatme, *Tracking targets using multiple robots: The effect of environment occlusion*, Autonomous Robots **13** (2002), no. 3, 191–205 (English).

[58] D. Karaboga and B. Basturk, *On the performance of artificial bee colony (ABC) algorithm*, Applications Soft Computing **8** (2008), no. 1, 687–697.

[59] Dervis Karaboga, *Artificial bee colony algorithm*, scholarpedia **5** (2010), no. 3, 6915.

[60] Dervis Karaboga and Bahriye Basturk, *A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm*, Journal of Global Optimization **39** (2007), no. 3, 459–471.

[61] Sanza T. Kazadi, *Swarm engineering*, Ph.D. thesis, Pasadena, CA, USA, 2000, AAI9972612.

[62] M. Kempka and et al., *Vizdoom: A doom-based AI research platform for visual reinforcement learning*, CoRR (2016).

[63] O. Khatib, *Real-time obstacle avoidance for manipulators and mobile robots*, Robotics and Automation. Proc. 1985 IEEE Int. Conf. on, vol. 2, Mar. 1985, pp. 500–505.

[64] Alexander Kleiner, *Mapping and Exploration for Search and Rescue with Humans and Mobile Robots*, Tech. report, Link"opings universitet, Department of Computer and Information Science, 2009, This is a Ph.D. thesis originally defended at University of Freiburg.

[65] Richard Klíma, Daan Bloembergen, Rahul Savani, Karl Tuyls, Daniel Hennes, and Dario Izzo, *Space debris removal: A game theoretic analysis*, Games **7** (2016), no. 3, 20.

[66] Mey Lean Kronemann and Verena V. Hafner, *Lumibots - making emergence graspable in a swarm of robots*, The ACM Designing Interactive Systems Conference, 2010, pp. 408–411.

[67] Mauri Kuorilehto, Marko Hännikäinen, and Timo D. Hämäläinen, *A survey of application distribution in wireless sensor networks*, EURASIP J. Wirel. Commun. Netw. **2005** (2005), no. 5, 774–788.

[68] Dimitrios Lambrinos, Ralf Mller, Thomas Labhart, Rolf Pfeifer, and Rdiger Wehner, *A mobile robot employing insect strategies for navigation*, Robotics and Autonomous Systems **30** (2000), no. 12, 39 – 64.

[69] N.P.-P.M. Lemmens, *Bee-inspired distributed optimization*, (2011).

[70] Nyree Lemmens, Steven De Jong, Karl Tuyls, and Ann Nowe, *A bee algorithm for multi-agent systems: Recruitment and navigation combined*, In Proceedings of ALAG, an AAMAS workshop, 2007.

[71] Nyree Lemmens and Karl Tuyls, *Stigmergic landmark optimization*, Advances in Complex Systems **15** (2012), no. 8.

[72] Jun S. Liu and Rong Chen, *Sequential monte carlo methods for dynamic systems*, Journal of the American Statistical Association **93** (1998), 1032–1044.

[73] P. Long, W. Liu, and J. Pan, *Deep-learned collision avoidance policy for distributed multiagent navigation*, IEEE Robotics and Automation Letters **2** (2017), no. 2, 656–663.

[74] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis, *Human-level control through deep reinforcement learning*, Nature **518** (2015), no. 7540, 529–533.

[75] Shervin Nouyan and Marco Dorigo, *Chain based path formation in swarms of robots*, Ant Colony Optimization and Swarm Intelligence (Berlin, Heidelberg) (Marco Dorigo, Luca Maria Gambardella, Mauro Birattari, Alcherio Martinoli, Riccardo Poli, and Thomas Stützle, eds.), Springer Berlin Heidelberg, 2006, pp. 120–131.

[76] Eli Packer, *Robust geometric computing and optimal visibility coverage*, Ph.D. thesis, Stony Brook, NY, USA, 2008, AAI3338238.

[77] C.E. Perkins and E.M. Royer, *Ad-hoc on-demand distance vector routing*, Mobile Computing Systems and Applications, 1999. Proc. WMCSA '99. 2nd IEEE Workshop on, feb 1999, pp. 90 –100.

[78] A Pirzadeh and W. Snyder, *A unified solution to coverage and search in explored and unexplored terrains using indirect control*, Robotics and Automation, 1990. Proc., 1990 IEEE Int.l Conf. on, May 1990, pp. 2113–2119 vol.3.

[79] R. L. Jhala P.V. Savsani, *Optimal motion planning for a robot arm by using artificial bee colony (abc) algorithm*, International Journal of Modern Engineering Research (IJMER) www.ijmer.com Vol. 2, Issue. 5, Sep.-Oct. 2012 p (2012).

[80] B. Ranjbar-Sahraei, K. Tuyls, I. Caliskanelli, B. Broeker, D. Claes, S. Alers, and G. Weiss, *Bio-inspired multi-robot systems*, Biomimetic Technologies (Trung Dung Ngo, ed.), Woodhead Publishing Series in Electronic and Optical Materials, Woodhead Publishing, 2015, pp. 273 – 299.

[81] B. Ranjbar-Sahraei, G. Weiss, and A. Nakisaee, *A multi-robot coverage approach based on stigmergic communication*, Multiagent System Technologies, Lecture Notes in Computer Science, vol. 7598, Springer, 2012, pp. 126–138.

[82] Bijan Ranjbar-Sahraei, Gerhard Weiss, and Ali Nakisaee, *An adaptive stigmergic coverage approach for robot team*, 24th Benelux Conference on Artificial Intelligence (BNAIC), 2012, pp. 210–217.

[83] Bijan Ranjbar-Sahraei, Gerhard Weiss, and Ali Nakisaee, *Stigmergic coverage algorithm for multi-robot systems (demonstration)*, Proc. of the 11th Int. Conf. AAMAS, vol. 3, 2012, pp. 1497–1498.

[84] J. Roberts and et al., *3-d relative positioning sensor for indoor flying robots*, Autonomous Robots **33** (2012), no. 1-2, 5–20 (English).

[85] M. B. V. Roberts, *Biology: a functional approach*, Nelson, 1986.

[86] Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal, *Programmable self-assembly in a thousand-robot swarm*, Science **345** (2014), no. 6198, 795–799.

[87] M. H. Saffari and M.J. Mahjoob, *Bee colony algorithm for real-time optimal path planning of mobile robots*, Soft Computing, Computing with Words and Perceptions in System Analysis, Decision and Control, 2009. ICSCCW 2009. Fifth International Conference on, Sept 2009, pp. 1–4.

[88] Erol Şahin, *Swarm robotics: From sources of inspiration to domains of application*, Swarm robotics, Springer, 2005, pp. 10–20.

[89] RashmiRanjan Sahoo, AbdurRahaman Sardar, Moutushi Singh, Sudhabindu Ray, and SubirKumar Sarkar, *Trust based secure and energy efficient clustering in wireless sensor network: A bee mating approach*, Pattern Recognition and Machine Intelligence (Pradipta Maji, Ashish Ghosh, M.Narasimha Murty, Kuntal Ghosh, and SankarK. Pal, eds.), Lecture Notes in Computer Science, vol. 8251, Springer Berlin Heidelberg, 2013, pp. 100–107 (English).

[90] H. Sak and F. Beaufays, *Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition*, CoRR (2014).

[91] Muhammad Saleem and Muddassar Farooq, *Beesensor: A bee-inspired power aware routing protocol for wireless sensor networks*, Applications of Evolutionary Computing (Mario Giacobini, ed.), Lecture Notes in Computer Science, vol. 4448, Springer Berlin Heidelberg, 2007, pp. 81–90.

[92] Jürgen Schmidhuber, *Deep learning in neural networks: An overview*, CoRR **abs/1404.7828** (2014).

[93] Eric Schneider, Elizabeth I. Sklar, Simon Parsons, and A. Tuna Özgelen, *Auction-based task allocation for multi-robot teams in dynamic environments*, pp. 246–257, Springer International Publishing, Cham, 2015.

[94] F.E. Schneider, D. Wildermuth, and H.-L. Wolf, *Motion coordination in formations of multiple mobile robots using a potential field approach*, Distributed Autonomous Robotic Systems 4 (LynneE. Parker, George Bekey, and Jacob Barhen, eds.), Springer Japan, 2000, pp. 305–314 (English).

[95] Mac Schwager, Daniela Rus, and Jean-Jacques Slotine, *Decentralized, adaptive coverage control for networked robots*, The International Journal of Robotics Research **28** (2009), no. 3, 357–375.

[96] Mac Schwager, Jean-Jacques Slotine, and Daniela Rus, *Unifying geometric, probabilistic, and potential field approaches to multi-robot coverage control*, pp. 21–38, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[97] J. Senthilkumar and M. Chandrasekaran, *Improving the performance of wireless sensor network using bee's mating intelligence*, **55** (2011).

[98] Weihua Sheng, Qingyan Yang, Jindong Tan, and Ning Xi, *Distributed multi-robot coordination in area exploration*, Robotics and Autonomous Systems **54** (2006), no. 12, 945 – 955.

[99] J. Snape, J. van den Berg, S. J. Guy, and D. Manocha, *Independent navigation of multiple mobile robots with hybrid reciprocal velocity obstacles*, 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, Oct 2009, pp. 5917–5922.

[100] Richard S. Sutton and Andrew G. Barto, *Introduction to reinforcement learning*, MIT Press, Cambridge, MA, USA, 1998.

[101] Guy Theraulaz and Eric Bonbeau, *A brief history of stigmergy*, Artif. Life **5** (1999), no. 2, 97–116.

[102] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics (intelligent robotics and autonomous agents)*, The MIT Press, 2005.

[103] J. van den Berg, Ming Lin, and D. Manocha, *Reciprocal velocity obstacles for real-time multi-agent navigation*, 2008 IEEE International Conference on Robotics and Automation, May 2008, pp. 1928–1935.

[104] Karl von Frisch, *The dance language and orientation of bees*, Harvard University Press, 1993.

[105] IA Wagner, M. Lindenbaum, and AM. Bruckstein, *Distributed covering by ant-robots using evaporating traces*, Robotics and Automation, IEEE Transactions on **15** (1999), no. 5, 918–933.

[106] HorstF. Wedde, Muddassar Farooq, and Yue Zhang, *Beehive: An efficient fault-tolerant routing algorithm inspired by honey bee behavior*, Ant Colony Optimization and Swarm Intelligence, LNCS, vol. 3172, Springer Berlin Heidelberg, 2004, pp. 83–94.

[107] A F T Winfield and Julien Nembrini, *Safety in numbers: Fault tolerance in robot swarms*, International Journal on Modelling Identification and Control **1** (2006), no. 1, 30–37.

[108] Alan F. T. Winfield, Christopher J. Harper, and Julien Nembrini, *Towards dependable swarms and a new discipline of swarm engineering*, Swarm Robotics (Berlin, Heidelberg) (Erol Şahin and William M. Spears, eds.), Springer Berlin Heidelberg, 2005, pp. 126–142.

[109] Vladimir Yanovski, Israel A. Wagner, and Alfred M. Bruckstein, *A distributed ant algorithm for protect efficiently patrolling a network*, Algorithmica **37** (2003), no. 3, 165–186.

[110] Robert Zlot, Anthony Stentz, M. Bernardine Dias, and Scott Thayer, *Multi-robot exploration controlled by a market economy.*, ICRA, IEEE, 2002, pp. 3016–3023.