Communication Algorithms for Spatio-Temporal Cooperation in Multi-Robot Systems

A Dissertation

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Doctor of Philosophy

in

Robotics Engineering

by

Nathalie Majcherczyk

December 2020

APPROVED:

Professor Giovanni Beltrame Dissertation Committee Polytechnique Montréal	Professor Jie Fu Dissertation Committee Worcester Polytechnic Institute
Professor Alexander Wyglinski Dissertation Committee Worcester Polytechnic Institute	Professor Carlo Pinciroli Dissertation Advisor Worcester Polytechnic Institute

Abstract

Swarm robotics has the potential to offer key solutions for large-scale, time-sensitive and dangerous applications, such as wildfire fighting and disaster response. Teams of robots promise capabilities beyond the reach of single-robot solutions by distributing intelligence, sensing and actuation at a large scale. This opportunity comes with the challenge of dealing with large amounts of data which are physically distributed across robots. Therefore, a key precondition for the swarm to coordinate successfully is the ability of the robots to store and exchange information efficiently.

This dissertation proposes a novel modular framework for organizing communication in highly mobile robotic swarms. Its first part addresses how to maintain continuous connectivity between robots via distributed motion coordination. This ensures that there are communication paths between any two robots to exchange information. The second part presents the design of a distributed data structure for low-memory, low-bandwidth, highly mobile swarms. The final part leverages previous insights and contributions to tackle distributed learning and collective perception applications.

The outcomes of this work include: *(i)* scalable connectivity maintenance algorithms tested extensively in realistic simulation and with real robots, *(ii)* a general and reusable platform for storing quantities of data that exceed the memory of individual robots, while maintaining near-perfect data retention in high-load conditions, and *(iii)* algorithms for collectively learning a machine learning model and improving accuracy of predictions through cooperation.

Acknowledgements

I wish to thank my PhD advisor, Professor Carlo Pinciroli, for his invaluable guidance and continued support throughout the course of my degree. I am extremely grateful for his mentoring and encouragement.

I would also like to express my deepest appreciation to my committee, Professors Giovanni Beltrame, Jie Fu, and Alexander Wyglinski, for their insightful comments that helped improve my dissertation.

I had the great pleasure of collaborating with Adhavan Jayabalan, Nishan Srishankhar, Daniel Jeswin, and Tim Antonelli. I am very grateful for their contributions and our research discussions.

Many thanks to all the past and present members of NEST Lab for their helpful advice and good times in the lab.

Finally, I would like to recognize my family and friends. Their patience and support have made this PhD possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Swarm robotics [1] is a branch of collective robotics that studies decentralized solutions for the problem of coordinating large teams of robots. Robot swarms are a promising technology for large-scale scenarios, in which performing spatially distributed tasks would entail prohibitive costs for single-robot solutions [1].

Multi-robot teams are advantageous over single-robot systems for a variety of reasons. The first compelling reason is that sensing and action capabilities are better distributed in space. As a consequence, the overall system performance can be greatly improved with multiple robots in terms of metrics such as task completion time and energy consumption. Another advantage of multi-robot systems lies in their inherent redundancy that can provide robustness and fault-tolerance. In terms of physical robustness, multi-robot systems offer tolerance to hardware failures when robots can act interchangeably to take over tasks from failed robots. Conversely, data fusion and information sharing are two mechanisms introducing information robustness in such systems; they allow recovering the global state from noisy or partial information. Furthermore, multi-robot systems can unlock the automation of complex tasks that can only be performed through the combination of simul-

taneous actions by multiple agents. For example, large or heavy object transport may require collective pushing and/or pulling at different points of contact. Overall, robot swarms can display greater scalability, reliability and versatility. They allow for robots to combine their abilities and computation power to tackle complex missions, while tolerating the failure of one or more robots [2].

Despite the advantages of multi-robot systems, significant research is needed to introduce robot swarms in real-world applications, special care is needed in designing collective behaviors to realize the above-mentioned potential. Collective behaviors include competitive behaviors, i.e, driven by individual interest, and cooperative behaviors, i.e, driven by common interest [2]. In this thesis, we focus on communication between robots as a means for solving challenges arising in decentralized cooperative behaviors.

1.1 Multi-Robot Cooperation and Its Challenges

Multi-robot cooperation [2] stems from the interaction of multiple robots to perform a task. When cooperating, robots are working towards a common interest and share a joint goal which may involve balancing several subgoals [2]. Representative examples of cooperative behaviors include object transport [3], search and rescue [4], exploration [5], and localization [6]. Effective solutions to achieve these behaviors constitute building blocks for real-world applications such as planetary exploration [7], deep underground mining [8], ocean restoration, and agriculture.

The above-mentioned applications present many challenges. Some challenges are related to the stringent requirements of the real-world environment of the targeted applications, while others are inherent to multi-robot cooperation. Real-world challenges stem from harsh conditions such as dangerous terrain, ever-changing en-

vironment, and a lack of existing infrastructure to control robots directly. Further operational difficulties involve the large scale of the environment and the need to operate over long periods of time. These aspects call for solutions in which robots are able to function and adapt to changes autonomously in a decentralized manner, i.e., independently of external infrastructure or operators.

Multi-robot cooperation introduces challenges linked to resource conflicts [2]. One such type of conflict manifests in space sharing wherein robots need to navigate while avoiding collisions, congestion, and deadlocks. More generally, conflicts occur when resources, e.g. space, are limited and robots act according to their local goal. Coordination is required to reconcile individual goals with group goals and "overcome groupthink or individual cognitive biases" [2]. This coordination can be achieved through communication between robots. In brief, learning information that is inferred or observed by others can help robots cooperate more effectively [2].

Deconflicting individual goals is especially challenging in the fully decentralized setting of robot swarms. Indeed, distributed techniques are devoid of any central or server-like unit that could access all the information and coordinate individuals plans. Therefore, inter-robot communication helps fulfill many functions of a server-like entity including synchronization, task allocation, and information fusion.

In this thesis, we build around the idea of using communication as a means of coordination. In particular, we focus on how to enable the exchange of information for robots to gain a better knowledge of the global state. The aim is to provide a base upon which to develop coordinated behaviors and online multi-robot planning. In this respect, we propose a method to build shared representations of the world in Part II. In Part III, we coordinate the training of a shared machine learning model predicting robot trajectories (Chapter 5) and we build a collective semantic map from aggregated information to reduce map uncertainty (Chapter 6).

1.2 Multi-Robot Communication for Cooperation

In order to enable multi-robot cooperation, distributed algorithms may utilize explicit communication between robots so as to align individual robot interests with the common goal, build shared world representations, or distribute the computational burden between robots. It is a major challenge to design a suitable framework at the intersection of communication networking, robot mobility, and distributed computation. In the following, we examine the framework employed in Wireless Sensor Networks at large. We highlight crucial differences with robotic systems and frame our contributions within this broader context.

1.2.1 Wireless Sensor Networks

Extensive work exists in the fields of Wireless Sensor Networks (WSNs) and Mobile Ad-hoc Networks (MANETs). WSNs refer to networks of static or low-mobility sensors while MANETs refer to networks of mobile devices [9, 10]. Networking algorithms for MANETs coordinate the exchange of data between network nodes. However, they typically assume that an independent mobility layer controls the trajectories of the nodes [11, 12]. This circumvents the issue of connectivity, i.e, the existence of communication paths between any two nodes, by assuming that it is ensured separately. These networking algorithms consider that pairs of nodes that need to communicate are connected or eventually will come within range of each other [13].

The fields of WSNs and MANETs have been extensively studied. Solutions have been proposed across various layers of abstraction, allowing for the deployment of WSNs in real-world scenarios. Figure 1.1 shows the communication protocol stack commonly used for Wireless Networks. It includes the TCP/IP model layers

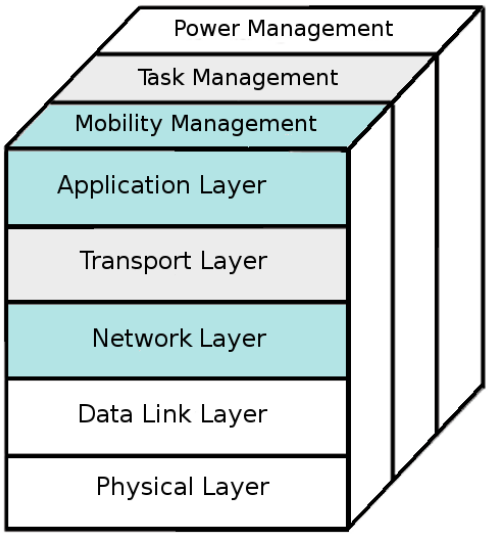


Figure 1.1: WSN protocol stack with protocol layers represented horizontally and management planes represented vertically. The blue boxes indicate areas to which this thesis provides contributions. The gray boxes indicate areas that affected some of the design choices in our approaches.

and the management planes relevant to WSNs but orthogonal to the traditional communication stack.

The *physical layer* deals with the transmission and reception of raw data between nodes through a physical communication medium [10]. This layer includes communication hardware. There exist many wireless technologies and standards; these technologies have different characteristics that make them suitable in different applications. Common wireless technologies include General Packet Radio Service (GPRS) and subsequent cellular technologies, WiFi, Bluetooth, and RFID. Some technologies are well suited to the Internet of Things (IoT), the network of smart devices [14], because they take into account energy consumption and are available at a low cost. For short communication range applications, those include Bluetooth Low Energy (BLE) [15] and ZigBee [16]. For long-range communications in that setting, the Low-Power Wide-Area Network (LPWAN) [17] is particularly well suited with technologies such as SigFox [18], NB-IoT [19], LoRa [20]. The *data link layer*

structures raw data into data frames and provides the procedure for the data transfer between nodes. It includes the Medium Access Control sublayer that manages the distributed use of the physical communication medium by the nodes [9]. The *transport layer* controls the data flow in the WSN. The *network layer* routes data coming through as controlled by the transport layer. Belding *et al.* characterize various classes of routing approaches. They also describe and classify existing routing protocols [21]. The *application layer* includes software applications related to the sensing task of the WSN [9, 10].

The *power management plane* enacts strategies for minimizing energy consumption, which may involve pausing certain functionalities. The *task management plane* allocates and plans sensing tasks for the WSN, so as to only employ the necessary nodes for sensing, while other nodes can concentrate on data aggregation and routing. In WSNs, the *mobility management plane* detects and adjusts for node movements and maintains a data route to the sink at all times [9]. In an effort to benchmark existing related work, Fang *et al.* developed an analytical method for analyzing the modeling and performance of mobility management schemes [22].

1.2.2 Robotic Networks

While WSNs have similarities with networks of communicating robots, robotic networks differ in that: *(i)* robots can physically interact with each other and the environment; *(ii)* robots can move with an intent designed by the developer, thereby offering the possibility to manage connectivity and information flow (mobility management); *(iii)* robotic networks may be self-sustained and never offload data to a server, robots may be the only users and handlers of the information and dispose of it as per operational needs; *(iv)* robots are often assigned unique identifiers to make addressing simpler; These differences motivate the need for new methods in

the aspects of networking that are affected by the above-mentioned differences.

Gosh *et al.* [10] state that "over the last decade, a handful of researchers noticed the significant disconnection between the robotics and the wireless network research communities and its bottleneck effects in the full-fledged development of a network of collaborative robots." They coin the term Robotic Wireless Sensor Networks (RWSN) to refer to the field at the intersection of robotics and wireless networks. They motivate this choice by noting that, in many applications, the primary task of robot teams is sensing. Relevant research also appears under the terms of Networked Robots or Robot Ad Hoc Networks (RANETs). There are adjacent fields that consider a specific type of robots such as: Flying Ad-hoc Networks (FANETs) [23], Vehicular Ad-hoc Networks (VANETs) [24, 25], and Aquatic Ad hoc Networks (AQNETS) [26].

We acknowledge the necessity of contributions at all levels of the communication protocol stack. In this thesis, we mostly tackle challenges in the higher layers as highlighted in blue in Figure 1.1. We describe related work at those levels of abstraction in the relevant chapters of the dissertation (Sections 3.2, 4.2, 5.2, and 6.2). In this thesis, we do not make a specific technological choice for the physical layer. Instead, we abstract away lower layers through simplifying assumptions discussed in Subsection 2.1. In practice, many types of wireless communication technologies are well suited for communication between robots. Radio Frequency (RF) technologies such as Bluetooth, and acoustic technologies such as Sonar are predominantly used in multi-robot systems. Aboveground communication mainly relies on RF technologies and, in particular, Bluetooth and infrared methods are used for short-range communication in those settings. In underwater settings, Sonar is the technology of choice as RF can not cope with turbulence that leads to high path loss and fading at radio frequencies [10].

1.3 Problem Statement

The overarching problem addressed in this thesis is how to achieve cooperation within decentralized multi-robot systems. We narrow our scope down to settings in which robots explicitly communicate to achieve a common goal. We analyze this topic at different levels of abstraction and study various facets of this *cooperation through communication*. We study a range of questions from more fundamental ones such as making sure any two robots can communicate, to more abstract ones such as collectively training Machine Learning models. We limit our investigation to decentralized systems with severe limitations discussed in the following.

Connectivity maintenance. The first research question we tackle is how to maintain connectivity in a mobile team of robots with a finite communication range. In particular, we study how to deploy a robot network from an initially connected configuration of robots, while enabling connected motion to target locations.

Data sharing. The second research question in this thesis is how to manage data in a distributed fashion given that robots are highly mobile and have limited bandwidth. Furthermore, robots allocate a limited memory to data sharing so the rest of the memory is available for the robots to perform complex perception and decision-making. A particular challenge in this problem is that the amount of data to store is much larger than this memory quota. The central aspect of this problem is: which robot should store what data and when?

Distributed applications. In a move towards distributed applications, we propose to study as a third research question how robots can learn collectively from distributed data. We specifically aim to train a global machine learning model through the cooperative action of a robot swarm with mobility and resource limitations. Finally, we consider the question of how to reduce the uncertainty of

pre-trained ML models through collective decision-making. In particular, we study how to realize data fusion of individual model outputs using a swarm of robots. This poses the challenges of collectively storing model outputs and performing fusion in a decentralized manner while coping with robot mobility and scarce resources.

1.4 Contributions

The work presented in this thesis collects several published contributions. We structure our contributions into three main parts (Figure 1.2):

Connectivity maintenance. A natural first step was to consider methods to establish communication paths between robots. In Chapter 3, we address the design of coordinated robot motion to maintain a connected network topology. This falls within the mobility management plane of Figure 1.1. Our solution creates and maintains a communication backbone through distributed policies. Our intent is not to use the underlying logical tree as the communication routing infrastructure, but as a method to minimally maintain global connectivity. In this work, we explicitly deal with constraints and requirements in the realm of robotics such as line-of-sight occlusions and realistic motion. In terms of networking, we use lightweight short-range communication between robots and make the simplifying assumptions described in Section 2.1.

Data sharing. While safety-critical applications require maintaining global connectivity of the network at all times, many applications may tolerate intermittent communication [27]. In Chapter 4, we design a shared data structure that relies solely on local and instantaneous information, making its performance less affected by topology changes. We address the challenge of dealing with large amounts of data that are physically distributed across highly mobile robots. Furthermore, we

consider that the amount of data that the robots must store is larger than the memory capacity of any individual robot. This contribution falls in the networking and transport layers of Figure 1.1.

Distributed applications. In the final part of the thesis, we study applications that rely on communication for robots to cooperate. The first application aims to jointly train a Machine Learning (ML) model for trajectory prediction by a team of robots through the Federated Learning [28] framework (Chapter 5). In our contribution, the scheduling of learning rounds is driven by the amount of data collected by the robots. Besides studying a server-based implementation, we also propose a fully decentralized method where robots communicate to schedule rounds and merge ML model weights. The second application is a problem of collective perception by a multi-robot system: the collective annotation of a map. We propose a method to store and fuse uncertain semantic annotations of objects proposed by the team of robots through voting (Chapter 6). Communications happen through the distributed data structure developed in Part II. In these applications, memory usage and communication overhead matter in evaluating overall performance. Therefore, we included those metrics in our evaluations.

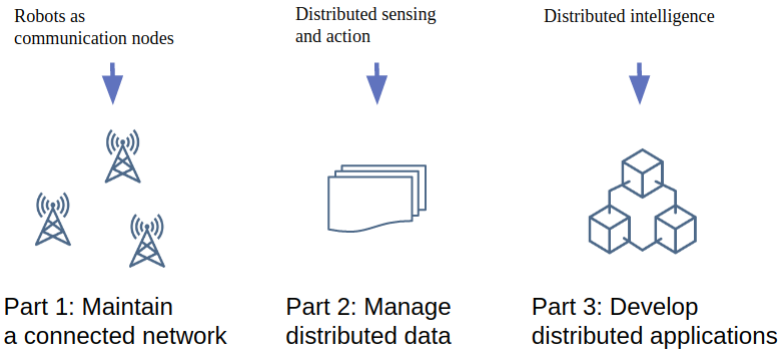


Figure 1.2: Thesis structure.

1.5 Publications

Some of the contributions of this dissertation have been submitted to or published in conference proceedings and journals. The main body of this dissertation consists of the following publications with minor changes:

1. **N Majcherczyk**, A Jayabalan, G Beltrame, C Pinciroli. "Decentralized connectivity-preserving deployment of large-scale robot swarms." 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2018. [29]

2. **N Majcherczyk**, C Pinciroli. "SwarmMesh: A Distributed Data Structure for Cooperative Multi-Robot Applications." 2020 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2020. [30]

3. **N Majcherczyk**, N Srishankar, C Pinciroli. "Flow-FL: Data-Driven Federated Learning for Spatio-Temporal Predictions in Multi-Robot Systems." Submitted to IEEE Robotics and Automation Letters. [31]

4. **N Majcherczyk**, D J Nallathambi, T Antonelli, C Pinciroli. "Distributed Data Storage and Fusion for Collective Perception in Resource-Limited Mobile Robot Swarms." Submitted to IEEE Robotics and Automation Letters. [32]

Chapter 2

General Networking Assumptions

2.1 Modeling Assumptions

In this section, we summarize the assumptions used in the dissertation when simulating inter-robot communications. We organize these assumptions into the categories represented in Figure 2.1. We adapt this figure and modeling framework from Dede *et al.*'s review of simulation tools and models for Opportunistic Networks [33]. We also include models and considerations from various surveys of robotic networks [23, 26].

Mobility Models. In ad-hoc and opportunistic networks, the mobility of nodes is controlled by external factors. Researchers use various models to simulate this ad-hoc behavior. In our case, the motion of robots is either part of our contribution as in Part I, or related to an application as in Part III. In Part I, the motion of the robots is *topology-based* since the goal is to coordinate the movements of robots continuously so as to maintain network connectivity constraints. In this model, robots coordinate based on their relative positions. In Part II and Chapter 6, we use a *random-based* mobility model in our simulations. In the context of swarms,

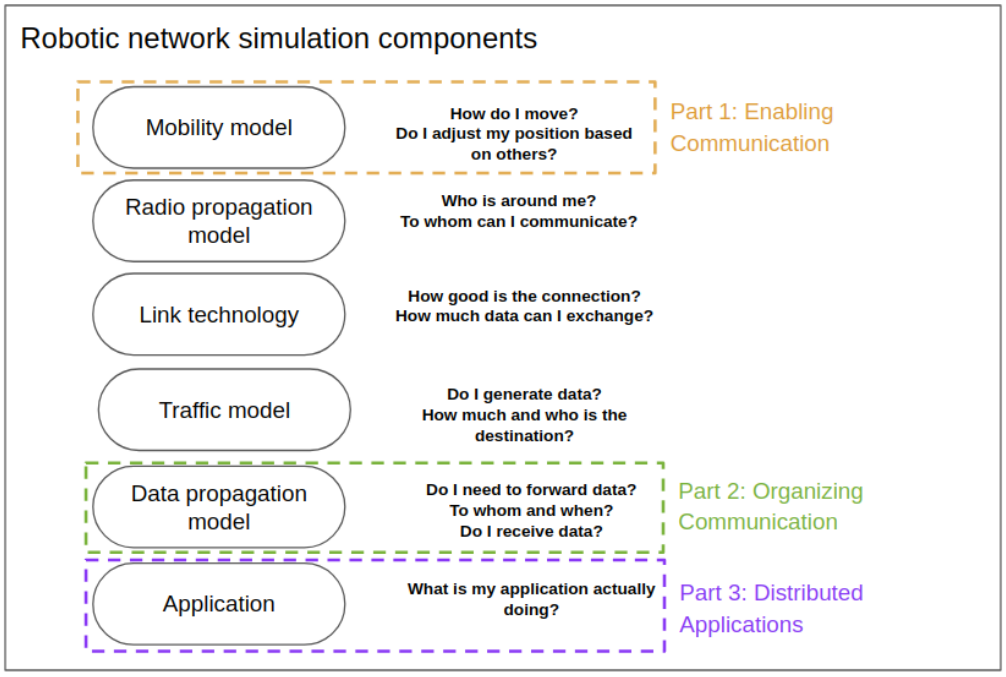


Figure 2.1: Network simulation components presented from the node perspective (figure adapted from [33]).

diffusion [34] is a common motion policy in which robots move in a random direction until they encounter an obstacle and change direction. We chose this model for simulating a random exploratory motion. In Chapter 5, we consider several mobility models based on well-studied swarm behaviors such as flocking [35] and foraging [36].

Radio Propagation Model. The power density of electromagnetic waves decreases as they travel because of various phenomena such as free-space loss, diffraction or scattering. The reduction in power density is called path loss (PL) and affects the transmission of data between devices. In free space, the PL can be calculated through the simplified Friis transmission formula:

$$PL[dB] = 20\ log_{10}\left(\frac{\lambda}{4\pi d}\right) \tag{2.1}$$

where d is the distance between transmitter and receiver, and λ is the wavelength.

When there are objects in the path of the waves, *i.e* line of sight (LoS) conditions are not met, the medium interrupting the waves causes either absorption losses or diffraction if it is transparent or opaque to electromagnetic waves respectively.

In this thesis, we restrict ourselves to a simple Unit Disk Graph (UDG) radio propagation model. We consider that two nodes are connected if they are within a certain communication range C of each other. Within the communication range, the path loss is assumed to be zero. Out of range, robots can not communicate and the path loss is assumed high enough to render any transmission impossible [33]:

$$PL(\text{distance})[dB] = \begin{cases} 0, & \text{if distance} \leq C \\ \infty, & \text{otherwise} \end{cases} \tag{2.2}$$

In their review of tools for simulating opportunistic networks, Dede *et al.* [33] present several more sophisticated synthetic propagation models including the Radio Irregularity Model, the Log Distance Path Loss and Ray Tracing. Dede *et al.* run simulations with the Nakagami model [37] and compare it to the UDG model. In that particular instance, there is an improved overall network performance with the more sophisticated Nakagami model including lower delivery delays. The increased model complexity requires higher computational resources for the simulation. Furthermore, whether the more sophisticated model is closer to reality than the UDG model remains an open question [33]. Given these insights and our intent to propose methods independent from any specific link technology, we elected to use the UDG model and performed some of our experiments with random packet drops and complete path loss in case of LoS obstruction.

Link Technology. In wireless networks, the link layer refers to the physical layer and the data link layer described in Section 1.2.1. This layer has the role of adapting the data from the higher layers of the protocol stack for the commu-

nication media used. The modeling of the link layer can potentially influence the simulation of network performance in terms of delivery rate and delivery delays because it deals with buffer management, data retransmissions and connections [33]. We described the possible wireless technologies and their suitability for different use cases in Section 1.2.1. In practice, it is impossible to completely and exactly model a real communication scenario in a set-up with obstacles and moving robots. In many studies and simulators, this leads researchers to use an 'ideal link layer' so as to simulate high-level protocols and data propagation [33]. In this thesis, we use this idealized view of the link layer. We consider short-range communication with or without modeling line-of-sight obstructions. We refresh the communication link existence every simulation time step which is typically fixed at 100 ms. Dede *et al.* observe that ideal and real link models yield similar results across the various simulators surveyed (OMNET++, ns-3, Adyton, and ONE). Therefore, the authors conclude in favor of using ideal link models as these models make simulation computations lighter. Using ideal link models also abstracts away the specific link technology thereby allowing the comparison of solutions regardless of the wireless medium [33].

Traffic Model. Another important aspect in simulating a network is the amount of data created and subsequently exchanged throughout the network. There are two main aspects in traffic models: the traffic size and traffic frequency. The traffic size relates to "how much data is created at once" and the traffic frequency considers "how often data is created" [33]. In terms of performance, the creation instant of a network message affects the message's delivery delay since the network load may be time-dependent. In Part II, we use a traffic model in which the creation time for each new message is drawn from a Poisson distribution. This type of model is commonly used for simulating user-driven network traffic such as text messaging for

instance [33]. In Part III, the traffic of the network is directly determined by the underlying application.

2.2 Performance Metrics

In the following, we consider several metrics typically included in benchmarks of networking research. In this dissertation, we include such measures at the level of abstraction considered in each chapter.

Delivery Delays. In networks, delivery delays measure the time between when a message is sent out and when it is delivered to its intended recipient(s). This metric can be quantified at the level of nodes or network-wide. Factors that influence this metric include the network topology and density, node mobility, and network traffic [33]. In Part II, we characterize our proposed distributed data structure in terms of delivery delays by showing distributions of time delays for write and read operations. In Part III, we record both timings due to the traffic frequency generated by the application and delays due to the propagation of data in the shared data structure.

Delivery Rate. The delivery rate, also called reception ratio, characterizes the proportion of data packets delivered to intended recipients before they are dropped from the network. Main reasons for data and messages to be removed from a network include expiration time (e.g. time-outs of requests) and memory overflows [33]. In Part II, we consider the fraction of data discarded due to node memory overflows given different network data loads. We also study how often read operations return all the data of interest.

Overheads. Various overheads can be studied to quantify the performance of a network with respect to a baseline. In Chapter 4, we consider the communica-

tion overhead associated with using a distributed storage as opposed to using a centralized memory. In Chapter 6, we study the memory utilization overhead with the proposed distributed data structure compared to an optimal memory usage as defined by a Bin Packing problem.

2.3 Summary

In this chapter, we reviewed models and metrics relevant for benchmarking wireless networks. We addressed the choice of assumptions and performance metrics made in the remainder of this thesis. Therefore, this choice is in accordance with the level of abstraction of our proposed contributions. We considered standard practices in Opportunistic Network simulation and trade-offs in terms of simulation complexity and specificity against accuracy.

Part I

Enabling Communication

Chapter 3

Connectivity Maintenance

3.1 Introduction

A common aspect in the cooperation scenarios described in Section 1.1 is the necessity to maintain a coherent state across the swarm. Many basic coordination problems can be solved assuming low-bandwidth, occasional communication or even no communication. However, global connectivity is an asset when information must be exchanged in a timely manner, either to optimize a global performance function, or to aggregate data in a sink. Task allocation scenarios with stringent space and time constraints, such as warehouse organization and search-and-rescue operations [38] are prime examples of this category of problems. In these scenarios, it is desirable for the robot network to allow both short-range and long-range information exchange.

In this chapter, we tackle the problem of deploying a robot network in a decentralized fashion, under the constraint that long-range information exchange must be possible at any time during a mission. We assume that robots must reach a number of distant locations. While navigating to these locations, the robots must spread without splitting the network topology in disconnected components. The robots

must achieve a final configuration in which data can flow between any two target locations, using the robots as relays.

It is important to notice that it is not required for all of the robots to take part in the final topology. Rather, it is desirable that as few robots as possible are engaged in connectivity maintenance, as this would free any extra robot for other tasks or to act as occasional replacement for any damaged robot in the topology. In contrast, the robots that are part of the final topology must form a persistent communication backbone that can be used by any robot when necessary.

This aspect sets apart our work from existing research on connectivity maintenance, which generally requires *all* robots to be part of the connected topology. The literature on this topic can be broadly divided in two classes: algorithms in which the robots must attain a final, static structure to maximize coverage [39], and algorithms in which global connectivity is enforced while navigating to a specific location as a single unit (flocking) [40]. Our work, in contrast, aims to create a dynamic, decentralized communication infrastructure that connects specific locations and uses as few robots as possible.

Our approach assumes that the robots are initially deployed in a compact, connected cluster. The robots then form a logical *tree* over the physical network topology. By growing the tree over time, the distribution of the robots progressively and dynamically extends to reach the target locations. The final configuration is a star-like topology, in which data can flow between any two target locations.

The main contributions of this work are:

1. The formalization of two algorithms to form and grow logical tree topologies that connect multiple target locations;

2. A comparative study of the algorithms, based on extensive physics-based sim-

ulations;

3. The validation of our findings through a large set of real-robot experiments.

The rest of this chapter is organized as follows. In 3.3 we formalize the problem statement. In 3.4 we present our methodology. In 3.5 we report an evaluation of the algorithms.

3.2 Related Work

Extensive literature exists on approaches for connectivity preservation. These approaches can be generally organized in three categories: graph-theoretic methods, edge selection methods, and

In graph-theoretic methods, the goal is to exploit a well-known node importance measure called *algebraic centrality* or *Fiedler value*. This measure indicates how much information flows through each node of the network and thus which nodes must are most important in connectivity maintenance. Yang *et al.* [41] introduced a decentralized algorithm to estimate the Fiedler value and use it to maintain connectivity while moving towards a target location. This algorithm was later refined by Sabattini *et al.* [42] and Williams *et al.* [43]. Further extensions include inter-robot collision avoidance [44] and multi-target exploration [40]. The main advantage of this family of approaches is that they allow navigation with arbitrary topologies. However, accurate decentralized computation of the Fiedler value is not easy in realistic settings in which messages might be lost due to communication interference [45]. In addition, computing the Fielder value in a decentralized manner involves network-wide power iteration methods [46], the slow convergence of which makes them suitable only for small teams of robots [43, 47]. It should also be

noted that all of the above algorithms, with the exception of [44], have only been demonstrated in simulated environments.

A second family of methods select a communication sub-graph and aim to preserve its edges through some form of global consensus. Hsieh *et al.* [48] devised a reactive control law based on radio signal and bandwidth estimation, in which links between robots can be activated and deactivated as the topology changes over time. Michael *et al.* [49] employed distributed consensus and auctions algorithms to establish which links to activate and deactivate over time. Cornejo *et al.* [13, 50] proposed a distributed algorithm for link selection in which the robots undergo a number of motion rounds, during which the selected links must be preserved. Being based on achieving global consensus before any topology modification can be finalized, these algorithms are not scalable and work best when teams involve a small number of robots.

A third class of connectivity-preserving algorithms assumes that a certain structure is pre-existing. The dynamic structure is some form of logical tree, dynamically built and updated over the physical links of the robot network. Our work falls into this category. Krupke *et al.* [51] employed a Steiner tree as a pre-existing structure, and use spring-like virtual forces to balance connectivity and cohesiveness while reaching distant targets. A number of works, which constitute our main source of inspiration, utilized minimum spanning trees as structures to preserve. Aragues *et al.* [39] focused on a distributed coverage strategy with connectivity constraints, and proposed a method based on maintaining a network-wide minimum spanning tree. Analogously, Soleymani et al. [52] proposed a distributed approach that constructs and preserves a network-wide minimum spanning tree, allowing for tree switching. Schuresko *et al.* [53] studied a theoretical approach for distributed and robust switching between minimum spanning trees. All these works were only demonstrated in

numerical simulations. The main advantage of these methods is the ease and speed with which spanning trees can be built and updated in a distributed manner. However, as discussed in this chapter, spanning trees do not scale well with the number of robots involved.

3.3 Problem Statement

3.3.1 Robot Dynamics

We consider N robots with linear discrete dynamics

$$x_i(t+1) = Ax_i(t) + Bu_i(t)$$

where $x_i(t) \in \mathbb{R}^{2M}$ is the state of robot i at time t, $u_i(t) \in \mathbb{R}^{2M}$ is the control signal, and $A, B \in \mathbb{R}^{2M \times 2M}$. The state $x_i(t)$ is defined as $[p_i(t), v_i(t)]$, where $p_i(t) \in \mathbb{R}^M$ designates the position of robot i and $v_i(t) \in \mathbb{R}^M$ its velocity. State and controls are subject to the convex constraints

$$\forall t \geq 0 \quad x_i(t) \in \mathcal{X}_i \quad u_i(t) \in \mathcal{U}_i.$$

In this work we focus on 2-dimensional navigation ($M = 2$).

3.3.2 Robot Communication

We assume that the robots are capable of *situated communication*. This is a communication modality in which robots broadcast data within a limited range C, and upon receiving data, a robot is able to estimate the relative position of the data sender with respect to its own local reference frame.

We define the *communication graph* $\mathcal{G}_C = (\mathcal{V}, \mathcal{E}_C)$, where $\mathcal{V}$ is the set of robots $\{1, \ldots, N\}$, and $\mathcal{E}_C \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges connecting the robots. An edge (i, j) between two robots exists at time t if their distance is within their communication range C, i.e., $\| p_i(t) - p_j(t) \| \leq C$.

Definition 1 (Graph connectivity) *A graph is* connected *is there exists a path between any two nodes.*

Graph connectivity can be verified through well-known concepts in spectral graph theory. From the definition of the graph adjacency matrix

$$A_{ij} = \begin{cases} 1 & \text{if } (i,j) \in \mathcal{E}_C \\ 0 & \text{otherwise} \end{cases}$$

and of the graph degree matrix

$$D_{ij} = \begin{cases} \sum_k A_{ik} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

we can derive the Laplacian matrix $L = D - A$. The graph is connected if and only if the second smallest eigenvalue of L is greater than 0. For this reason, this eigenvalue is called *algebraic connectivity* or *Fiedler value* [54]. We will employ algebraic connectivity as a performance measure in the experiments of 3.5.

3.3.3 Objectives

The objective of this work can be stated as follows: we aim to create a progressive deployment strategy that can reach an arbitrary number of geographically distant tasks while satisfying connectivity constraints. In particular, the final configura-

tion of the network topology must allow communication between any two target locations.

3.4 Approach

3.4.1 Roles

In both algorithms, we assume that the robots are initially deployed in a fully connected cluster. Subsequently, the robots must form a tree by dynamically assuming a specific role in the process.

In both tree-forming algorithms, the robots can have four possible roles: *root*, *worker*, *connector*, or *spare*. The *root* robot corresponds to the tree root, and at any time during the execution, only one robot can assume this role. The *worker* robots are the tree leaves, and they correspond to robots that must reach the target locations, forcing the tree to grow progressively. The *connector* robots dynamically join the tree to support its growth, leaving the pool of available *spare* robots.

3.4.2 High-Level Behavior Specification

The algorithms can be formalized through a high-level state machine that encodes the behavior of every robot, as depicted in Figure 3.1.

Every robot starts in state INIT. We assume that a process that assigns the role of *worker* to the robots closest to the targets has been already executed, through, e.g., a task allocation algorithm or a gradient-based algorithm. In addition, a random robot is assumed assigned the role of *root*. The other robots are initially *spare*.

The START TREE state is triggered by the root, which propagates a signal throughout the robot network. This state signifies that a new tree must be cre-

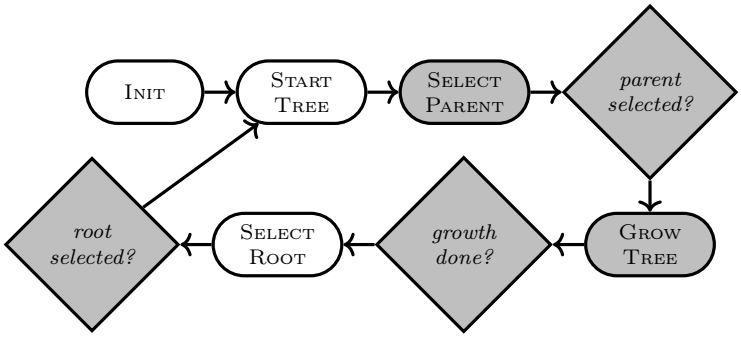


Figure 3.1: The high-level Finite State Machine that formalizes the individual robot behaviors in the two tree-formation algorithms. Rounded rectangles denote states, and diamonds denote *barriers*, i.e., conditions that all robots must meet before proceeding to the next state. States filled in white are common among both algorithms; states and barriers filled in light gray differ across algorithms.

ated. As the message propagates throughout the network, the robots estimate their distance from the root. This is possible because of situated communication—every robot can estimate a relative vector to each of its immediate neighbors.

Robots receiving a "start tree" signal switch to Select Parent. In this state, each robot must identify a new parent to attach to. The selection of a new parent aims to create the shortest possible paths between the root robot and the *worker* robots, i.e., the leaf nodes in the tree. The specifics of this state are different in the *outwards* and *inwards* algorithms, and are explained in 3.4.4 and 3.4.5. At the end of this state, a robot is part of two trees—the one from the previous iteration of the algorithm (excluding the very first iteration), and a new one that reflects the new parent.

Once every robot has selected a new parent, the robots switch to the Grow Tree state, in which the robots forget the tree from the previous iteration and *spare* robots are accepted to join an edge. The algorithms differ in the implementation of this state, and details are reported in 3.4.4 and 3.4.5.

Once the growth state is complete, the robots switch to the Select Root state. As the tree grows, the initial choice of the root robot (which is random) or an

uneven distribution of target locations might render the tree topology nonoptimal. By selecting a new root, the swarm can balance the tree branches, thus fostering even growth over time. The design of this state is illustrated in 3.4.3.

Finally, the new assigned root switches to state START TREE and broadcasts a new "start tree" signal.

In Figure 3.1, certain state transitions are marked with diamonds. These transitions, which we call *barriers*, are special in that they correspond to "wait states" in which the robots must stay until a certain condition is verified for every robot. The specific implementation of these conditions depends on the algorithms. However, the general principle is that the root aggregates the information necessary to evaluate a certain condition, and then broadcasts a "go" signal throughout the tree. The "go" signal triggers a state transition in the robots that receive it.

3.4.3 Selection of a New Root

The purpose selecting a new root is to balance the tree, which fosters better growth and compensates for an uneven distribution of target locations. In addition, balancing the tree has positive effects on the scalability of our algorithms. Every state in our algorithms involves some form of diffusion/aggregation process across the tree, with a time complexity that is linear with the depth of the tree. By balancing the tree, we also shorten its depth, thus lowering the time for diffusion/aggregation processes to complete.

These considerations suggest that the best location for the root is as close as possible to the centroid of the distribution of robots. The selection of a new root occurs at the end of a tree configuration loop, but the data upon which the process depends is collected in state SELECT PARENT, when the robots select a new parent.

The algorithm provides an estimate of the centroid in the root reference frame

by adding up each robot contribution from the leaves to the root. The algorithm is formalized in Algorithm 1. An intuitive explanation of this algorithm proceeds as follows. Since each robot only knows its relative position to other robots, it must send to its parent an accumulation vector $\mathbf{q}_i$ which aggregates its contributions and that of all its descendants in the tree, according to its own reference frame. Figure 3.2 reports an example with three robots, where robot 0 is the root, robot 2 is a *worker*, and robot 1 is a *connector*.

To perform the final calculation of the centroid, Algorithm 1 needs the number of robots in the swarm. A tree-based distributed algorithm to count the number of robots currently committed in the tree is reported in Algorithm 2. This algorithm requires the robots to aggregate a partial count, denoted with c_i, from the tree leaves to the root.

In our implementation, both Algorithm 1 and Algorithm 2 are executed in parallel in state SELECT PARENT. In SELECT ROOT, the current root compares its position and the position of its neighbors to the centroid estimate (all are expressed in its reference frame). If the current root is the closest to the centroid, it remains the root and restarts a new tree loop. Otherwise, it designates a new root and sends the centroid vector and the angle to the new root. When the new root receives this message, it sends an acknowledgment message to the old root, and then it expresses the centroid in its own reference frame. The process is repeated until the root is the closest robot to the centroid estimate.

3.4.4 The Outwards Algorithm

The intuition behind the outwards algorithm is to build a logical spanning tree over the entire robot network. The process starts at the root, and robots join the tree progressively.

Algorithm 1 Distributed centroid estimation algorithm executed by robot i: $\mathbf{a}_i$ denotes an accumulator value; $\mathbf{q}_i$ denotes the contribution of robot i to the estimation algorithm; c_i and d_i denote the number of robots in the swarm estimated by robot i and the tree depth of robot i, respectively; and $\mathbf{p}_i^{\text{parent}}$ is the vector from robot i to its parent.

$\mathbf{a}_i = 0$
for all child j **do**
    $\mathbf{q}_j^i =$ express $\mathbf{q}_j$ in i's reference frame
    $\mathbf{a}_i = \mathbf{a}_i + \mathbf{q}_j^i$
if robot i has a parent **then**
    $\mathbf{q}_i = \mathbf{a}_i - (\underbrace{c_i - d_i}_{\text{nb descendants}} + 1) \cdot \mathbf{p}_i^{\text{parent}}$
if robot i is the root **then**
    $\mathbf{q}_i = \mathbf{a}_i / \underbrace{c_i}_{\text{robot count}}$

Algorithm 2 Tree-based count algorithm for robot i. The depth of robot i in the tree is denoted as d_i. The depth of the tree root is set to 1. The count calculated by robot j is denoted as c_j.

switch number of children **do**
    case 0
        return d_i
    case 1
        return c_{child}
    default
        return $\sum_{\text{neighbors } j}(c_j - d_i) + d_i$
end switch

In state SELECT PARENT, robot i considers its neighbors as potential candidates. Viable candidates are non-*workers* already in the tree and at a distance smaller than the communication range. Among these, the robot selects the closest robot. The robot commits to the tree and starts broadcasting its parent id, which indicates to the parent robot that robot i is a child and that i is a *connector*. Each *connector* maintains its list of children and checks for obstructions of line-of-sight with respect to its parent. If a robot can not receive data from its selected parent, it selects another parent and updates its data.

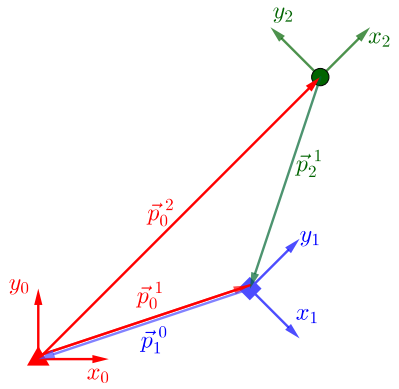


Figure 3.2: The red triangle represents robot 0 with the root reference frame. The blue square represents robot 1, which is a child of robot 0 and a parent of robot 2, in turn represented by the green circle.

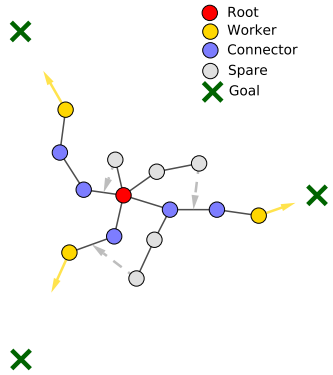


Figure 3.3: Spare management in the *outwards* algorithm. The useful tree edges (blue nodes) are extended by pruning useless tree branches (grey nodes).

In state GROW TREE, the robots undergo two main phases: first, they discard the information about the old tree; second, they prune tree branches that contain no *workers*. To establish whether a branch contains a *worker*, when a *worker* selects a parent (state SELECT PARENT), the latter propagates this information upstream towards the root.

The branches not containing a *worker* are considered "useless" and the robots that are part of them take the *spare* role. To disband a useless branch, *spare* robots leave it starting from the leaves. The leaves curl the branch back towards the root, and upon entering in contact with another branch might decide to join it. The logic

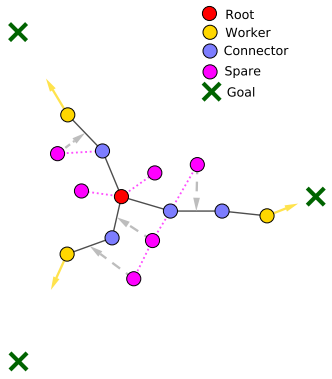


Figure 3.4: Spare management in the *inwards* algorithm. Useful tree edges (blue nodes) are extended by adding *spare* robots (purple nodes)

for spares to join a branch is explained in 3.4.6

3.4.5 The Inwards Algorithm

The intuition behind the *inwards* algorithm is that the robots join the tree starting from the workers towards the root. Growth is therefore directed, and the final topology is a *sparse* tree, in that only a subset of the robots takes part in it. The *spare* robots, in contrast to the *outwards* algorithm, do not form branches; rather, they disperse along the tree and select a robot to use as reference.

In state SELECT PARENT, viable candidates for parent selection are non-workers in the tree or robots not in the tree which are at a distance smaller than the communication range C. Among these, a robot selects a neighbor with the smallest distance to the root. When the robot i commits to the tree, it broadcasts its parent id, which indicates to the parent robot that robot i is a child and that i is a *connector*. In the *inwards* algorithm, by definition, all branches are useful because they all terminate with a *worker* as leaf node.

In state GROW TREE, *spare* robots attempt to join a branch. The logic for branch joining is the same as in the *outwards* algorithm, and it is explained in 3.4.6.

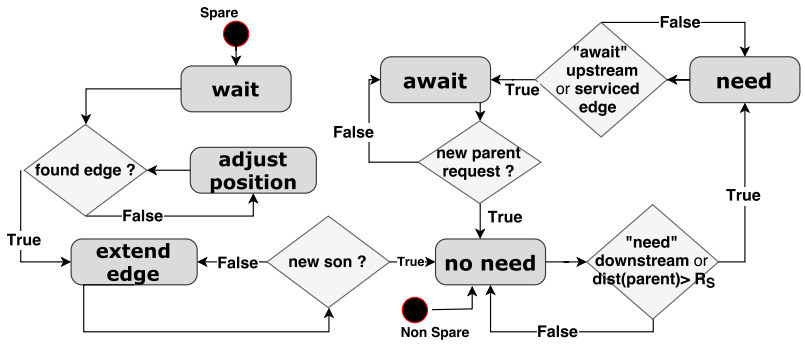


Figure 3.5: Interaction between *spare* and *non-spare* robots.

3.4.6 Spare Management

The state machine diagram in Figure 3.5 describes the part of the GROW TREE state that concerns the interaction between *spare* robots and non-*spare* robots (i.e., *connectors*, *workers*, and *root*).

Non-spare robots enter the NO NEED state when they have no need for a spare robot. They exit this state either if their distance to their parent becomes smaller than the *safe communication range* S, or if at least one of their children's state is the NEED state. In the NEED state, each robot continuously checks if it is in an edge selected by a *spare* robot, or if their parent is in the AWAIT state. If one of these conditions is fulfilled, the robot transitions to the AWAIT state. In the AWAIT state, the robot is waiting the insertion of a spare robot either in one of its edges or upstream in the tree.

Spare robots enter the WAIT state and look for an edge to extend. They transition to the EXTEND EDGE state or the ADJUST POSITION state after performing a search for edges in need among their neighbors. In the ADJUST POSITION state, spare robots rotate around their parent if they are within the safe radius or move towards their parent in a straight line otherwise. In the EXTEND EDGE state, spare robots head for the middle of the edge to be extended.

3.4.7 Robot Motion

The integrity of the tree over time is ensured by constraining the robots' motion. We enforce the constraints by expressing the robot motion as a sum of virtual potential forces (we omit time dependency for brevity of notation):

$$u_i = \begin{cases} u_i^{\text{tree,old}} + u_i^{\text{tree,new}} & \\ \qquad + f_i(d_i^{\text{parent}})(u_i^{\text{target}} + u_i^{\text{avoid}}) & \text{if } d_{i,j} \leq E \\ \mathbf{p}_i^{\text{parent}} & \text{otherwise} \end{cases}$$

where $d_{i,j} = \| p_i - p_j \|$, $E < C$ is the *emergency* range beyond which a robot is dangerously distant from its parent, and

- $u_i^{\text{tree,old}}$ and $u_i^{\text{tree,new}}$ indicate the interaction law between robots (i, j) in a parent-child relationship, in either the old or the new tree. We use the control law

  $$u_i^{\text{tree}} = \frac{\epsilon}{d_{i,j}} \left(\left(\frac{\delta}{d_{i,j}} \right)^2 - \left(\frac{\delta}{d_{i,j}} \right)^4 \right)$$

  where $\delta = E$ and ϵ are parameters to set at design time.

- u_i^{target} is a control law that attracts a robot to a target, promoting tree growth. For workers, this is a force that points the assigned target location l_i and calculated with

  $$u_i^{\text{target}} = \tau \frac{l_i - p_i}{\| l_i - p_i \|}$$

  where τ is a design parameter. Workers propagate to their parents the calculated u_i^{target}, and connectors apply it in turn.

- u_i^{avoid} is a repulsive force for obstacle avoidance between neighbors not in a parent-child relationship.

Table 3.1: Optimized design parameters.

Type	Symbol	Meaning	Outwards	Inwards	Unit
Outgoing bandwidth		Maximum of outgoing bytes	3	3	kB/s
Motion	S	Safe range between parent and child	138.93	135.25581	cm
	A	Non-parent-child avoidance range	43.16	40.99	cm
	δ	Ideal distance between parent and child	190	154.0841	cm
	ϵ	Factor gain in parent-child interaction	10	10	
	τ	Magnitude of attraction to target	0.49	0.2539	
Tree Growth	R	Reconfiguration period	38.8	44.0	sec
	I	Information liveness period	1.2	0.5	sec
Uncommitted Management	E	Distance threshold for *spare* recruitment	132.09	132.1353	cm
	J	Distance threshold to switch to *connector*	9.79	6.6395	cm

- $f_i(d_i^{\text{parent}})$ is a function defined as follows:

$$f_i(d_i^{\text{parent}}) = \begin{cases} 1 & \text{if } d_i^{\text{parent}} \leq S \\ 0 & \text{otherwise} \end{cases}$$

  where d_i^{parent} is the distance between a robot and its parent and $S < E$ is the *safe communication range*. Through this function, a robot can ignore navigation to target and obstacle avoidance to perform emergency maneuvers when the distance to its parent becomes unsafe.

3.5 Evaluation

3.5.1 Parameter Setting

The dynamics and the performance of our algorithms depends on the design parameters reported in 3.1. To set their value, we used a genetic algorithm. We ran multiple instances of the optimization process for both *inwards* and *outwards*, and 3.1 reports the best values we found.

Every instance of the optimization was executed for 100 generations. We set this number as a reasonable margin after observing that, across instances, after

about 50 generations the optimization process would find a plateau beyond which no improvement was found.

Every generation consisted of trials in which 9 Khepera IV robots[1] were placed in the arena in a tight cluster. We configured two types of trials:

- 2 target locations on a circle with a radius of 2.3 m at 180 ° from each other;
- 3 targets on a circle with a radius of 1.6 m at 120 ° from each other.

We ran the trials in the ARGoS multi-robot simulator [55], and maximized a two-step performance function. The first step (performance 0 to 1) promoted connectivity maintenance by penalizing the time spent with disconnected robots; the second step (performance 1 to 2) was activated when no disconnections occurred, and higher values corresponded to lower times to reach the targets.

3.5.2 Simulated Experiments

We tested the performance of the algorithms by varying three parameters: the target radius, the redundancy factor, and number of targets. We placed multiple targets on a circle with equal angles between each other. The *target radius* is the radius of the circle. We chose radii of 3, 6 and 9 meters corresponding to small, medium and large scales. The *redundancy factor* is the factor by which we multiply the minimum required number of robots needed to reach all the targets given our communication range. We tested the values of 2, 3 and 4 for this parameter. The *number of targets* was 2, 3, and 4. The largest configuration we considered involved 94 robots. Each scenario was executed with 50 different random seeds. We ran all the experiments for both algorithms with and without activating line-of-sight obstructions in the communication models of ARGoS, to test the effect of this aspect.

[1] `https://www.k-team.com/mobile-robotics-products/khepera-iv`

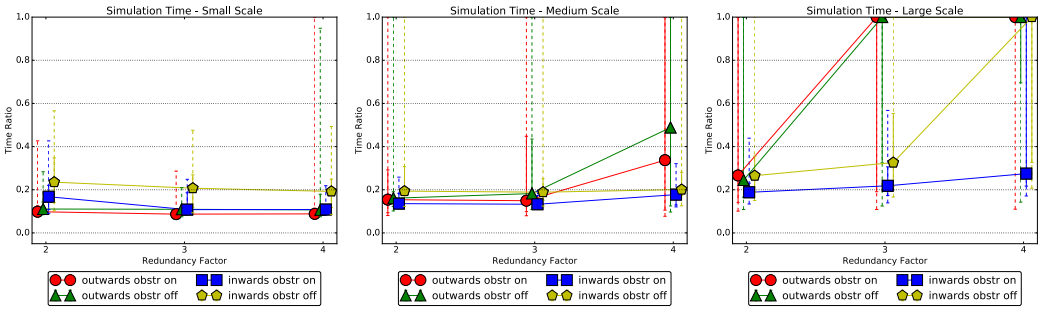


Figure 3.6: Assessment of mission completion time in simulation.

Simulation Time

We studied the time performance of both algorithms, and declared an experiment finished when all workers reach their targets. To compare results across different scales, we normalized the mission duration by the maximum allowed time. The maximum allowed time was computed by considering the time for a robot to reach a target from the center of the arena; this time was then multiplied by 10. The results are reported in Figure 3.6. For small scales, the *outwards* algorithm outperforms the *inwards* algorithm. However, as the scale of the experiment is increased, the directed growth of the *inwards* algorithm is increasingly advantageous. In addition, with the *outwards* algorithm, some missions do not reach their targets in the allotted time limits when higher redundancy factor is employed. This is due to the increased interference that too many useless branches create in robot navigation. This effect is not prominent in the *inwards* algorithm because the robots are added to the tree only when it is necessary.

Disconnected Time

We studied the ability to maintain connectivity by considering the following metrics: (i) The *disconnected time ratio*, defined as the number of time steps (over the total experiment time) with at least a broken edge in the tree; (ii) The *Fiedler value time*

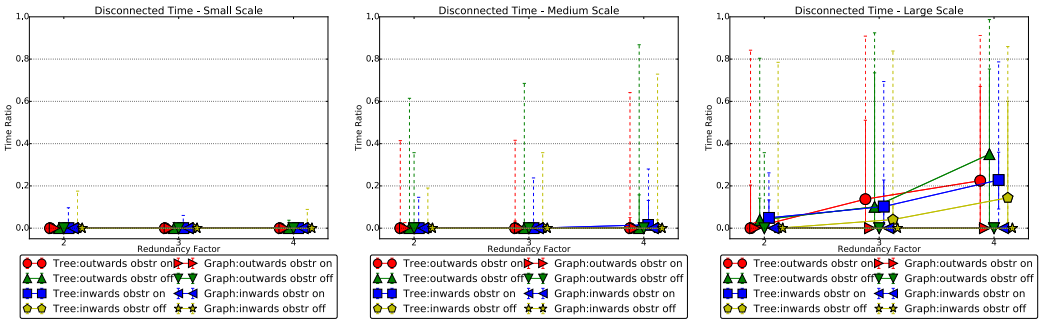


Figure 3.7: Assessment of connectivity loss.

ratio, defined as the number of time steps (over the total experiment time) with swarm-wide Fiedler value lower than 10^{-3}. The results are reported in Figure 3.7. In small-scale scenarios, in only two experiments out of 50 have positive disconnected time, and the global communication graph always stays connected. In medium-scale scenarios, larger numbers of redundant robots cause occasional line-of-sight obstructions that delay messages exchanges, but connectivity is generally maintained throughout the duration of the experiment. In large-scale scenarios, the disruptive effect of a large number of redundant robots is prominent for both algorithms. With fewer robots, the inwards algorithm is capable of maintaining global connectivity in all of the experiments, despite occasional breaking of tree edges (in less than 5% of the experiments).

Tree Selection Complexity

To understand the scalability of each algorithm we analyzed how long it takes build a tree using both algorithms. Figure 3.8 shows the line fitting of data points for the inwards algorithm.

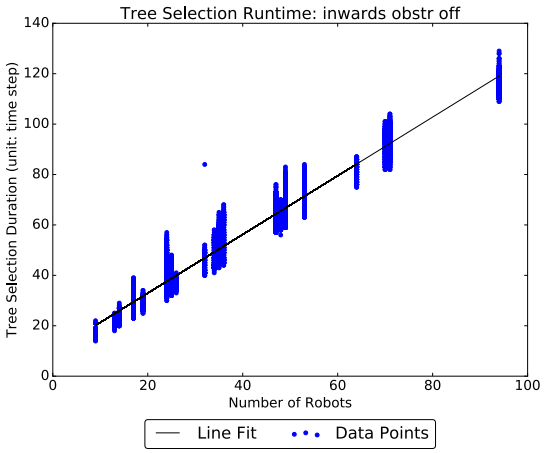


Figure 3.8: Tree selection complexity.

3.5.3 Real-Robot Validation

To validate the simulated results simulations, we tested our algorithms with 9 Khepera IV robots (see Figure 3.9). A Vicon motion capture system was used to track the position and orientation of the robots throughout the duration of the experiments, and to simulate situated communication. We employed 2 experimental scenarios: (i) 2 targets on a circle with a radius of 2.3 meters at approximately 180 degrees from each other; (ii) 3 targets on a circle with a radius of 1.6 meters at approximately 120 degrees from each other. We rescaled the distance-related parameters in Table 3.1 to fit the arena and accommodate for the small number of robots involved. We repeated these experiments 15 times for setup (i) and 10 times for setup (ii) with robots starting from the same positions and orientations, to allow for better comparison. We also performed the same experiments in simulation, with the same initial positions.

Figure 3.10 shows that real-robot and simulated experiments follow analogous trends. In particular, we verified that for small-scale experiments with low redundancy factor (in these experiments it was set to 1) the *outwards* algorithm has better performance than the *inwards* algorithm.

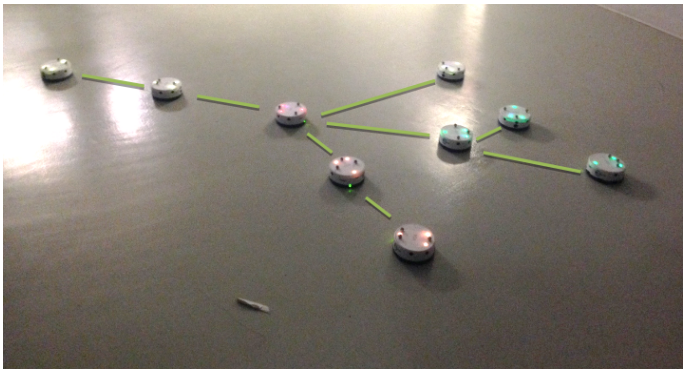


Figure 3.9: Experimental setup with 9 Kheperas IV robots.

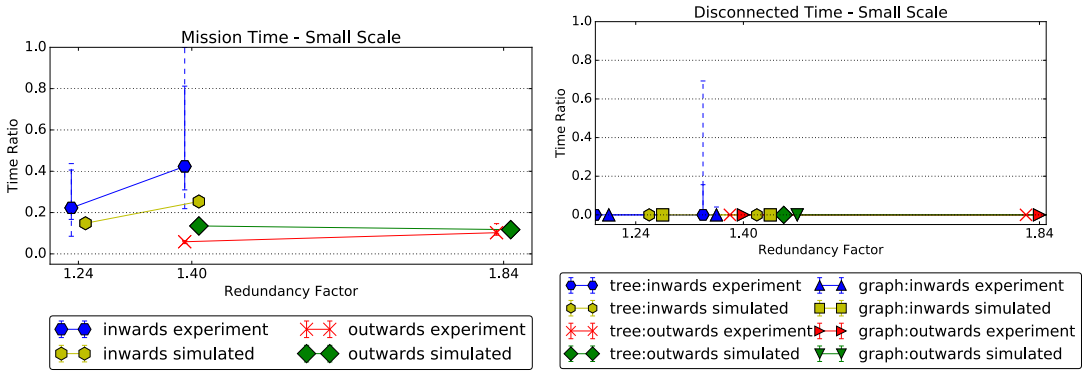


Figure 3.10: Results of real-robot evaluation.

3.6 Summary

In this chapter, we presented two algorithms to construct a long-range communication backbone that connects multiple distant target locations. The algorithms are decentralized and based on the idea of constructing a logical tree over the set of physical network links.

We performed a large set of experiments, both in simulation and with real robots, to assess the performance of the algorithms according to various experimental conditions. Our results show that, in small-scale scenarios, *outwards* tree growth, corresponding to spanning tree formation, is a viable approach. However, as the scale of the environment and the number of robots involved increase, a more directed, *inwards* growth from target locations towards the tree root, is a preferable approach.

Our results also show that, as the number of unnecessary robots increases, the benefit of redundancy is voided by the increased physical interference in navigation. While a better spare robot strategy could diminish this phenomenon, our results suggest that a more progressive approach to deployment might be a better idea.

3.7 Open Problems

The presence of a reasonable number of spare robots offers the opportunity to tackle the problem of maintaining *persistent* long-range global connectivity despite individual limitations in the energy supply of individual robots.

In addition, possible extensions of our work include the presence of moving targets, rather than static ones, and the presence of obstacles in the environment.

Thirdly, a future direction of study is how to make this approach *robust* or *resilient* to robot failures. In the current implementation, individual failures have consequences of different severity depending on the role assumed by the robot and the swarm state at the time of failure. The most severe failure is the loss of the current root robot and results in an unrecoverable state. The loss of other robots is mitigated by the periodic reconfiguration of the logical tree. Special care should therefore be paid to safeguard the root robot.

Finally, multi-robot simulations could be extended to include more sophisticated networking models. This would reduce the gap between practical networked robotic systems and simulated systems. To the best of our knowledge, networking libraries are not directly available in multi-robot simulators. Some work has been proposed to interface network simulators and multi-robot simulators. Kudelski *et al.* [56] proposed a framework to interface ARGoS and ns3. This could be used to test a particular hardware implementation of our algorithms.

Part II

Organizing Communication

Chapter 4

Distributed Data Sharing

4.1 Introduction

Recent work studies the integration of multi-robot systems with centralized computation platforms, such as databases or cloud computing systems [57]. This approach enables one to aggregate information in a central location and perform efficient map merging, task allocation, and global state estimation – in other words, combining data storage with computational capabilities. This approach is particularly effective in indoor environments, such as warehouses, production chains, and hospitals, in which communication with a central system can be expected to be reliable.

However, many applications are not easily amenable to this approach. Mapping in remote locations, space applications, and disaster recovery are examples in which access to a centralized infrastructure is problematic, limited, or even impossible. In these applications, rather than envisioning a multi-robot system as *part* of a larger infrastructure, it would be desirable for it to *be* the infrastructure. These applications also entail the collection of large amounts of data, whose storage might exceed the capacity of any individual robot.

As a step in this direction, we study the realization of a decentralized data structure for storing, managing, and performing computation with shared data. We make three basic assumptions:

- Every robot devotes a quota of memory and bandwidth to storing and routing data. The amount of memory can change across robots;
- The amount of data that the robots must store is larger than the memory capacity of any individual robot;
- The network topology is dynamic due to robot motion.

Given these assumptions, we study how to distribute the data across the swarm. In designing a solution, we realized that in many applications certain features of the data play an important role for mission success. For example, mission-critical data should be stored in well-connected robots—in case of a temporary disconnection this data would be as widely available as possible. Analogously, the physical location of the data might suggest that certain robots are more suitable for storage than others.

The rest of this chapter is organized as follows. In Section 4.2 we discuss related work. The design of our data structure is presented in Section 4.4. We report the results of our performance evaluation in Section 4.5, and conclude the chapter in Section 4.6.

4.2 Related Work

4.2.1 Peer-to-Peer Networks

In peer-to-peer networks, common implementations of data sharing involve Distributed Hash Tables (DHTs). DHTs couple a distributed key partitioning algorithm

and a structured overlay network to provide a self-organized data storage service. Information is abstracted in the form of tuples which are `(key, value)` pairs. The fundamental problem is to decide how to distribute tuples between nodes for storage. The key partitioning algorithm assigns ownership of a set of keys to each node in the network. The overlay network imposes a routing structure that makes for efficient search across the nodes. Comparative surveys [58],[59] highlight the main features of these protocols. The Content-Addressable Network (CAN) protocol partitions the key space by splitting a virtual toroidal space into zones. CAN maps tuples to points owned by nodes in the virtual space using a uniform hash function [60]. In the Chord [61], Pastry [62], and Tapestry [63] protocols, nodes determine NodeIDs according to the structure of the desired overlay network. Tuples are then addressed directly to NodeIDs or partial NodeIDs. These distributed data structures provide self-organizing, scalable and addressable storage. However, node additions and removals are costly as the topology needs to be maintained through reorganization. Furthermore, they can cause local network failures. Because they form relations between nodes randomly, unstructured overlay networks such as Gnutella and BitTorrent [64] provide alternatives when the network participant turnover is high. These protocols offer robustness to node removals at the cost of increased degree of centralization or loss of guarantees when locating data.

In the above mentioned protocols, the selection of neighboring nodes in the overlay network lacks physical grounding. This means that neighbors in the network could be far away from each other. Since routing information over longer geographical distances increases energy consumption and latency, there has been an effort to incorporate node location into overlay networks. Three main trends exist within this body of research: (1) Geographic layout, which constructs the overlay network so that neighbors are close in the physical space [65], [66], [67], [68]; (2) Proxim-

ity routing, which considers node proximity while routing in the existing overlay network [69]; (3) Proximity neighbor selection, which weighs in proximity between neighbors when constructing the overlay network [70]. These methods add a notion of node locality. However, the network topology only changes to accommodate node additions and removals but not motion. Therefore, they fail to capture the inherent dynamicity of robotic systems.

4.2.2 Mobile Ad-Hoc Networks

There is a vast body of research in Mobile Ad-Hoc Networks (MANETs) that seeks to address communication between mobile interconnected devices. Some sensor networks have motion and fall in that category. One trend in those systems has been to use naming and data-centric routing and storage. This means that a name is associated to given data and that name determines to which node the data is addressed [71]. Similarly to swarm systems, the main features of sensor networks are that they are limited in energy, memory and computational power. Sensor networks perform distributed data processing and storage. However, the goal is to eventually offload the processed data to a base station. Furthermore, sensors typically do not act on the environment or perform cooperative and autonomous decision-making.

4.2.3 Vehicular Ad Hoc Networks

Vehicular Ad Hoc Networks (VANETs) are systems of interconnected cars and road stations. Different types of routing protocols have been studied within that field, they can be divided into: proactive routing, reactive routing and position-based routing which depends on beaconing and forwarding [72], [25], [24]. These systems share some similarities with swarms but differ in that they have specific topologies and mobility patterns. Typically, cars in the back make decisions based on cars up

front. The lanes and roads are narrow so the number of direct neighbors is small.

4.2.4 Multi-Robot Networks

Several papers compare and assess the use of existing databases in multi-robot applications [73], [74] and [75]. These comparisons reveal that most existing databases rely on a central server. An exception is the work of Sun *et al.* [76], who adapted Distributed Heterogeneous Hash Tables and position-based routing to propose a solution for task allocation in a warehouse setting.

4.2.5 Swarm Networks

In the context of swarm robotics, Pinciroli *et al.* proposed a distributed tuple space called *virtual stigmergy* [77] that copes with frequent topology changes. In this approach, each robot maintains a local time-stamped copy of the data which is only accessed upon read and write operations. This mechanism works well with node mobility and limited bandwidth but it leads to full data duplication. This means that the collective memory of the system is under-utilized. The SOUL file sharing protocol [78] builds on virtual stigmergy and unstructured overlay networks to enable sharing of larger-size data in the form of `(key, blob)` pairs. SOUL involves locally storing blob meta-data on each node and splitting blobs into datagrams across different nodes. This decomposition uses a bidding mechanisms that minimize the reconstruction cost at so-called processor nodes. This method addresses the problem of managing data files with a focus on how to split, distribute and recombine them. Memory usage is improved but meta-data is still fully duplicated across nodes for each of the files. Various update and bidding processes increase latency in the network.

4.3 Problem Setup and Challenges

In this section, we describe the fundamental assumptions imposed both on the multi-robot system and on the nature of the events to record in the physical environment. We proceed by describing challenges of the distributed storage problem in this context.

4.3.1 Ad-hoc Robotic Network

We consider an autonomous and decentralized system of N robots which act as both the infrastructure for storing information and sole users of this information. We define the system across the following features:

Communication Modalities We assume that the robots have the ability to exchange data within a communication range C. This implies the existence of an ad-hoc network with each robot acting as a node. We further assume that robot communication is limited to gossiping, i.e., broadcasting messages to all neighbors within C. Because we also desire to route some messages from one robot to another using the point-to-point communication modality, we assume that the robots have a constant unique identifier $i \in \{1, \ldots, N\}$ and a variable node identifier δ_i made known to their neighbors. The knowledge of i singles out a specific robot, while δ_i enables the selection of a suitable storage node for a specific tuple.

Finite Resources We impose a realistic finite bandwidth on outgoing messages. We also limit the memory capacity M_i of each robot allotted for the self-organizing data management process. Variable $m_i(t)$ records the amount of memory used by robot i at a given time.

Dynamic Topology The robots are moving according to a logic defined by the developer. Robot motion follows linear dynamics and has a limited speed. The number of neighbors $ngbrs_i(t)$ of a robot changes over time.

4.3.2 Inputs

We consider inputs to the data structure to stem from events which have a position $\mathbf{x} \in \mathcal{R}^d$ and happen at a time $t \in \mathbb{N}$. Such events can be, e.g., records of a physical phenomenon sampled at a particular time and place, records of an internal robot state, or records of swarm-wide state. To implement a data structure in which robots can retrieve and update tuples encoding some events, each specific tuple needs a unique identifier τ meaningful to all network nodes. In particular, for updates, tuples need to have a notion of version. For convenience, we achieve versioning by time-stamping tuples with a global time. Distributed synchronization algorithms such as vector clocks [79] can be used to implement this aspect.

4.4 Methodology

4.4.1 Overall Architecture

We describe our design following the structure depicted in Figure 6.2. SwarmMesh provides algorithms across three levels of abstraction.

User-level querying

As stated in Section 4.1, robots are at the same time the networking infrastructure and users of the data stored by the network. As a user, a robot can execute different querying commands on the data structure. These operations are meant

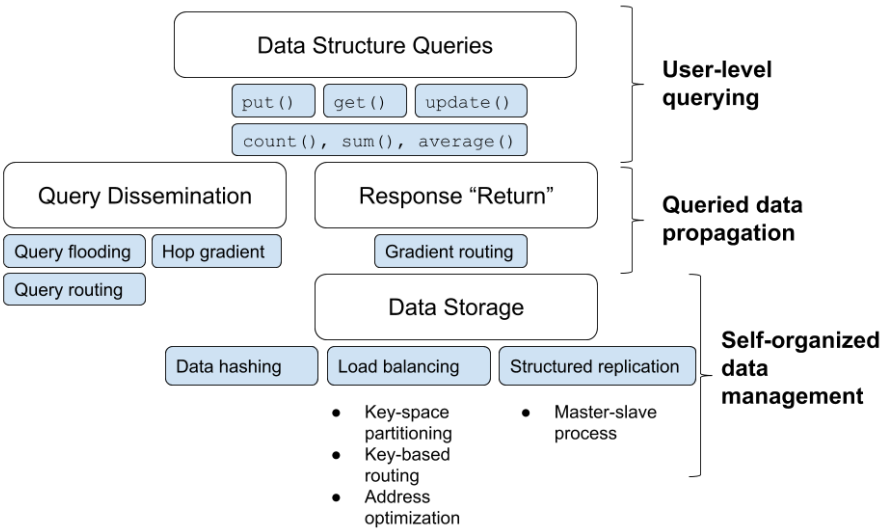


Figure 4.1: Overall architecture.

to enable modifying and retrieving information stored globally as required by the robot behavior. This behavior is defined by the developer and independent from SwarmMesh.

Queried data propagation

Another layer of SwarmMesh handles the dissemination of user read and write queries throughout the data structure. Read queries are flooded across the network. This type of query requires replies from certain nodes to be routed back to the robot emitting the query. We route write operations to a suitable node for storage in a point-to-point fashion.

Self-organized data management

The bottom layer determines how the tuples get distributed across nodes. It also ensures a certain degree of robustness by creating inactive replicas in other nodes in a controlled way. The main design intuitions driving the data distribution are that: (1) some events/tuples are more important than others (hierarchy in data hashing);

(2) some nodes are better suited to hold more valuable tuples than others (hierarchy in key-space partitioning); (3) the hierarchy of nodes changes very often and should be updated based on local information only.

4.4.2 User-level Querying

A robot user can perform the following operations:

- `put(k`[1]`,v)`: writes a tuple into the data structure. It performs an `erase(k)` to remove any potential outdated version of the tuple and a `store(k,v)` of the new tuple.
- `store(k,v)`: assigns a tuple to a particular node in the data structure.
- `erase(k)`: removes a tuple from the data structure.
- `get(k,`Δ`)`: returns all the values corresponding to keys $\in [\, k - \Delta;\, k + \Delta \,]$.
- `get(x,y,r)`: returns all the values for tuples located within a radius r of the point (x, y) expressed in a global reference frame. To use this feature, we need the added assumption of a global reference frame and the ability to locate events in this reference frame.

Robots can also perform in-network computation:

- `count(k,`Δ`)` or `count(x,y,r)`: returns the number of tuples with keys $\in [\, k - \Delta;\, k + \Delta \,]$ or located within a radius r of the point (x, y).
- `sum(k,`Δ`)` and `sum(x,y,r)`: returns the sum of values corresponding to keys $\in [\, k - \Delta;\, k + \Delta \,]$ or located within a radius r of the point (x, y).

[1]The key argument `k` has an uniquely identifying part τ and a content-dependent part (or hash) ρ_τ.

- `average(k,Δ)` or `average(x,y,r)`: returns the result of the corresponding `count()` and `sum()` operations as a pair.

- `min(x,y,r)` or `min(k,Δ)`: returns the minimum value in the associated spatial range or key range.

- `max(x,y,r)` or `max(k,Δ)`: returns the maximum value in the associated spatial range or key range.

As explained in Section 4.3.2, a write operation should be the result of some local information processing performed by robots in the vicinity of an event. Existing methods in sensornets can be applied to locally synthesize low-level sensor readings into a result describing a higher level event such as source detection [80].

In our implementation, we consider an event to be a meaningful low-level sensor reading. In order to trigger a single data structure write in the vicinity of the event, we locally elect a leader to perform a `put(k,v)` operation. The election logic can be redefined by the developer, although the specifics of this aspect are beyond the scope of this contribution.

The return values of the `count()`, `sum()` and `average()` operations percolate across nodes. The user robot which emitted the initial query must combine the intermediate return values into the final result. Monotonic (i.e., commutative and associative) operations such as `min()` and `max()` do not require combining intermediate results.

The performance of spatial queries, i.e., operations with arguments (x, y, r), and that of queries by key, i.e., operations with arguments (k, Δ), depends heavily on the way we distribute the data in the network. Our approach is meant to be modular and we present two possible data hashing functions in Section 4.4.4.

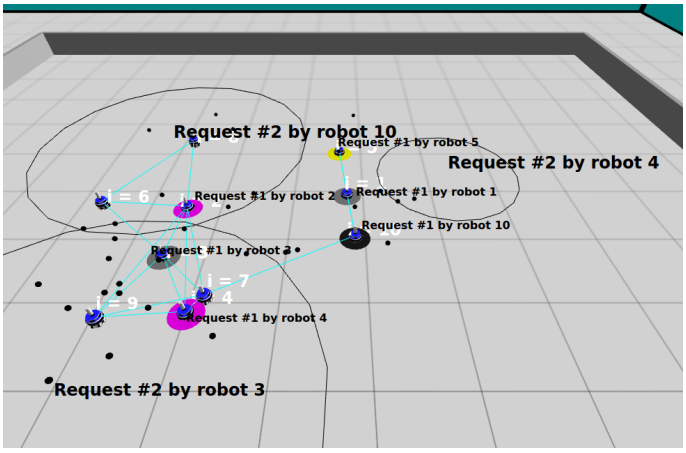


Figure 4.2: Black dots show locations of events previously written into the data structure. Queries of the type `get(x,y,r)` are drawn with black circles representing the area covered. Queries of type `get(k,` $\Delta = 0$`)` are represented by colored disks under the query source robot; the color maps to a specific key.

4.4.3 Queried Data Propagation

Read-operation flooding

Queries that aim to retrieve data from the data structure are flooded to all nodes. Each robot emitting a read query computes the query's unique identifier by concatenating the value of its query counter and its robot unique identifier.

Hop gradient

While flooding the network with a read query, we opportunistically create a gradient to the source of the query. Upon reception, every robot increments a hop counter included in the query message and broadcasts it further along. For each received request, the robot stores the query unique identifier and hop count in a circular buffer.

Reply gradient routing

The hop gradient gives us a convenient way to route replies back to the source node by forwarding replies from nodes with a higher or equal hop count. For this, we rely on the assumption that the motion of the robots preserve a gradient path to the source for long enough, which is a realistic assumption in most settings when comparing motion speed and information propagation speed.

Write-operation routing

When a robot writes the result of some local information processing to the data structure, the tuple may be routed to a different robot for storage based on its key. This algorithm is described in Section 4.4.4.

4.4.4 Self-organizing Data Management

Data Hashing

When writing a tuple using `put(k,v)`, the robot must compute the key `k`. In our protocol, a key should be in the format $k_\tau = (\tau, \rho_\tau)$ where τ is a tuple unique identifier and ρ_τ is a value that maps to one or multiple nodes which can store the tuple.

The robot assigns τ by concatenating its robot unique identifier and the count of tuples it has written into the data structure. Each field has a set number of digits so that every τ is unique. As stated previously, our design considers that events vary in importance and we use this property to distribute them across nodes.

Read queries described in Section 4.4.2 can either use $\texttt{k} = \rho_\tau$ or $\texttt{k} = (\tau, \rho_\tau)$ for tuple addressing. Queries for ρ_τ can yield multiple tuples while queries in (τ, ρ_τ) relate to a specific tuple.

The robot computes ρ_τ using a function mapping a characteristic of the event to its relative importance. We select the function such that the higher ρ_τ, the more valuable the piece of information. We propose two hashing functions:

- Category-based: A robot can register different types of events. For example, it can mean that the robot has several different on-board sensors and determines the event type by the triggered sensor. We use a ranking function $R_T(s_\tau)$ that assigns higher values to event types that we consider most important:

$$h_C : type_\tau \mapsto \rho_\tau = R_T(type_\tau)$$

.

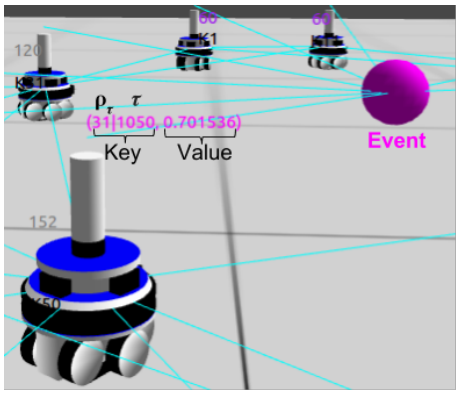


Figure 4.3: Category-based hashing.

- Spatial: We decide that in a global reference, tuples further away from the origin are the most desirable because they are difficult to discover by robots. This idea can be generalized to specific areas in any reference frame: h_{SP} : $(x_\tau, y_\tau) \mapsto \rho_\tau = \sqrt{x_\tau^2 + y_\tau^2}$.

Key Partitioning

Similarly to other distributed data structures such as DHTs, nodes partition the key space to decide which one of them needs to hold tuples corresponding to specific keys.

As stated in Section 4.4.1, we use the idea that some nodes are superior than others. Our intuition is that a robot with more neighbors $ngbrs_i$ is less likely to get disconnected from the swarm and is better positioned to dispatch tuples upon query. A second insight is that the more free data memory $m_i(t)$ a robot has, the less likely it is to overflow its memory and discard information. We also desire to have instantaneous self-organized partitioning completely based on local information. Therefore, we chose to make nodes assign themselves a node identifier δ_i as follows:

$$\delta_i(t) = \begin{cases} m_i(t) \cdot ngbrs_i(t) & \text{if } ngbrs_i(t) > 0 \\ 1 & \text{otherwise} \end{cases}$$

A node with node identifier δ_i can hold a tuple with key (τ, ρ_τ) if $\delta_i(t) > \rho_\tau$. We refer to this condition as (H) in the rest of this text. The free memory variable is in number of tuples. In order to store tuples in the data structure, we should match the frequency distributions of data hashes and node identifiers, i.e., there should be nodes with unique identifiers at least high enough to hold the hashed tuples. This has implication on the design of the hashing functions. They should map to values smaller than $\max_i(M_i) \cdot (N-1)$ and spread the data across likely node identifiers.

Key-based Routing

If a robot holds one or more tuples not satisfying (H), it places them in a routing queue. It then tries to send them starting with the highest ρ_τ to a robot with a high

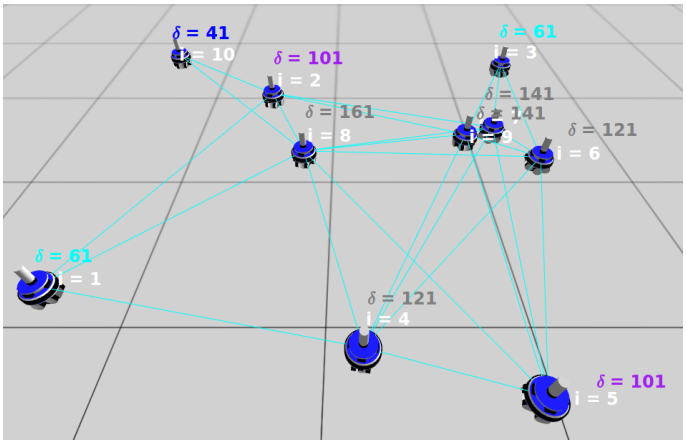


Figure 4.4: Key-space partitioning.

enough node identifier. If there are candidates satisfying (H) to receive the tuple, the sender picks one at random. If none of the neighbors satisfy the condition, the robot sends it to the neighbor of highest δ_i. We impose a limit on the memory capacity M_i and divide it into routing and storage capacities. In case of overloads on M_i, the robot discards the least important tuple, i.e., with lowest ρ_τ.

Address Optimization

When a robot has an empty routing queue and it stores a tuple with ρ_τ closer to the node identifier of a candidate neighbor, we let the robot evict the tuple to the corresponding neighbor. This is an optimization to ensure efficient access to a tuple by key. We further noticed that requiring at least a half full storage memory helps balancing the load between nodes.

Structured Replication

To ensure robustness to node failures, we make copies in neighboring nodes using a source-replica approach. The source is the robot holding the original tuple. The source picks a replica to hold an inactive copy of the tuple. Robots do not return

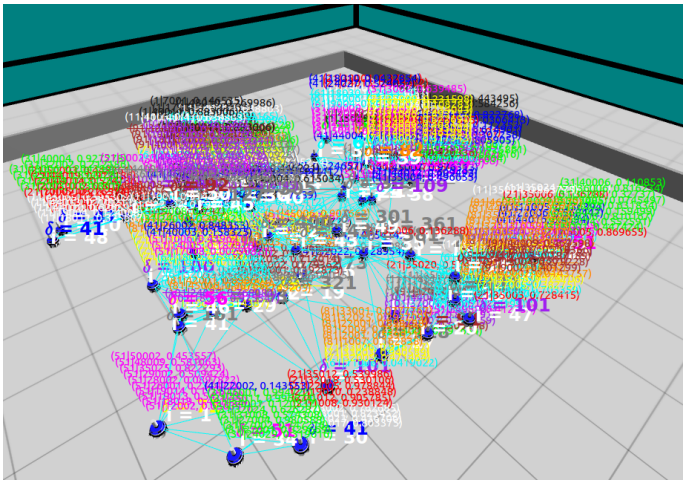


Figure 4.5: Address optimization. Tuples tend to be grouped by color. Each color represents a different category of event.

inactive tuple copies upon queries; this ensures consistency. Source and replica exchange a heartbeat signal. If the source fails to receive the heartbeat signal within a time-out duration, it picks another replica. The source can also send a kill signal to cancel the inactive copy. The source cancels the copy if the replica gets outside of a safe radius of communication ($\ll C$) or if it decides to route the active tuple to another robot. If a replica fails to receive the heartbeat within the time-out duration from the source, it activates the tuple copy.

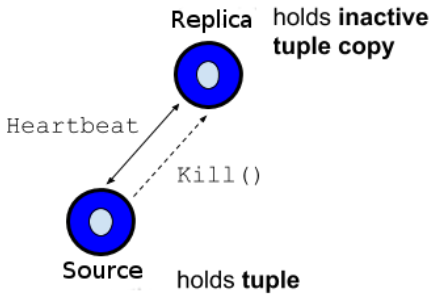


Figure 4.6: Structured replication.

4.5 Evaluation

4.5.1 Metrics and Parameters

We evaluated different aspects of our approach such as scalability, memory-related performance, and routing protocol efficiency.

To study *scalability*, we performed our simulated experiments with different numbers of robot inside an arena sized to imposed different robot densities (see Table 4.1). These densities imply that the ad-hoc network stays often connected even with diffuse robot motion. This enables us to study a system facing intermittent disconnections.

In hash tables, the *load factor* is the number of data items over the number of memory slots (buckets). This parameter indicates the load of the data structure and is typically used to decide when to partition of the memory into an increased number of buckets. For our distributed and self-organized approach, we define the load factor as $l_f = \text{number of events}/(N \cdot S)$, where S is the storage capacity. The memory capacity M includes both storage and routing capacities.

To understand the *performance* of the key-based routing algorithm, we track the number of hops and time steps for a tuple to be routed to a suitable node for storage and upon query. We implemented messaging with a queue and we imposed a limit on the bandwidth for outgoing messages. The message format is displayed in figure 4.7. The robot fills the message buffer progressively and sends the partial message if the limit is exceeded. It attempts to send the remainder by type at the next time step. We only send one tuple at a time.

To study *availability*, we considered the fraction of tuples received over the expected tuples for a `get()` query. We checked for consistency by confirming that active copies of tuples were all unique.

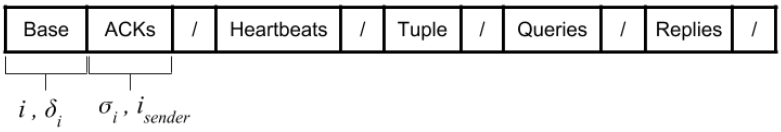


Figure 4.7: Message format.

Table 4.1: Simulation parameters.

Parameter	**Value**
Number of Robots N	$\{10, 50, 100\}$ robots
Communication range C	2 m
Memory capacity M_i	20 tuples $\forall\ i$
Storage capacity S	10 tuples
Routing capacity R	10 tuples
Time step	0.1 s
Bandwidth	5.7 kB/s
Robot density	$\{0.6, 1\}$ robot/m^2
Robot speed	$\{0, 5\}$ cm/s
Load factor	$\{0.6, 0.7, 0.8, 0.9, 1\}$
Event sensing range	1 m
Event types	12
Events generation rate	5 events/s
Query generation rate	1 query/s

4.5.2 Simulated Experiments

We tested our system using the ARGoS multi-robot simulator [55]. We ran simulations with and without robot motion. We picked a simple diffusion motion with a maximum forward speed of 5 cm/s. For the purpose of testing all available features, robots are equipped with a range and bearing sensor ($C = 2$ m), a GPS and a sensor detecting colored spheres. We disabled line-of-sight obstructions. To materialize events, we put colored spheres in the environment with each color representing a category of event. Events were generated in time according to a Poisson distribution and placed in space according to a uniform distribution.

Memory-Related Performance

In our simulations, we allocate limited memory and bandwidth to the data sharing process. Upon receiving tuples that it can store, a robot progressively fills its storage memory. The cap on bandwidth combined with the decision to route one tuple per

Table 4.2: Tuple retention for $N = 50$ across load factors.

Load Factor			**.6**	**.7**	**.8**	**.9**	**1**
category hashing	static topology	min	1	1	1	.996	.95
		mean	1	1	1	.999	.983
		max	1	1	1	1	1
	dynamic topology	min	1	1	1	.996	.986
		mean	1	1	1	.999	.992
		max	1	1	1	1	1
spatial hashing	static topology	min	.993	.932	.973	.909	.758
		mean	.999	.99	.987	.948	.899
		max	1	1	1	.98	.954
	dynamic topology	min	.997	.994	.985	.984	.94
		mean	.999	.999	.996	.993	.985
		max	1	1	1	1	.998

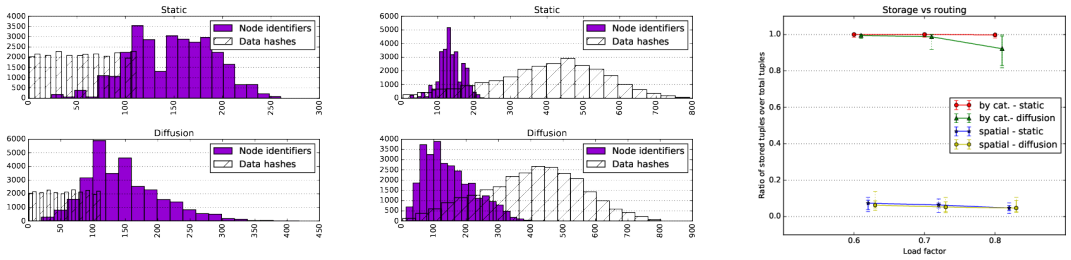
time step results in some tuples being temporarily placed in a routing memory. The goal is to keep the storage memory under a value S and the routing memory under a value R. However, we allow either memory to temporarily cross that threshold provided that the combined memory usage stays under the memory capacity M_i. Any memory overflow leads to robots discarding tuples of lowest rank. To assess the ability to retain large amounts of information in the data structure, we generate different numbers of inputs corresponding to load factors between 0.6 and 1.0. We repeated simulations with and without robot motion and using either the h_C or h_{SP} hash function. Tables 4.2 and 4.3 show that the fraction of retained tuples is almost always equal to 1, even with high load factors. This indicates that the collective memory is properly utilized with robots sharing the data load. In comparison, an approach that uses full duplication and the same individual memory constraints would retain N times less tuples (excluding the routing memory).

In order to evaluate key partitioning, we show histograms of node identifiers and data hashes across all simulations with 100 robots in Figure 4.8. In Figure 4.8a, we use the category-based hash function h_C with a mapping of 12 types of events to values in $\{1, 11, 21, \ldots, 121\}$. We generated the types of events uniformly in the simulations which naturally leads to the white dashed bar graphs in Figure 4.8a. Node identifiers are the product of the current node degree in the communication

Table 4.3: Tuple retention for $N = 100$ across load factors.

Load Factor			**.6**	**.7**	**.8**	**.9**
category hashing	static topology	min	1	1	1	.997
		mean	1	1	1	.999
		max	1	1	1	1
	dynamic topology	min	1	1	.991	.977
		mean	1	1	.999	.996
		max	1	1	1	1
spatial hashing	static topology	min	.955	.912	.841	.72
		mean	.982	.97	.927	.866
		max	1	.999	.989	.977
	dynamic topology	min	.995	.993	.994	.972
		mean	.999	.999	.998	.997
		max	1	1	1	1

graph and the node's remaining storage memory. Therefore, the node identifier distribution depends on a combination of communication graph topology and load allocation. Both situations represented for the category-based have the node identifier distribution to the right of the data hashes. In the upper graph, robots are static and their spatial coordinates are sampled in uniform distributions. In the bottom graph, robots diffuse in an arena sized to impose certain robot densities (see Table 4.1). In Figure 4.8b, we use the spatial hash function h_{SP} and we show a situation where events are uniformly generated up to 8 m from the origin of the global reference frame. The function h_{SP} maps the distance in cm to ρ_τ which yields the Gaussian distribution represented by the white bar graphs. In both the static and moving case, the node identifier distribution is to the left of the data keys distribution. This means that, given the key partitioning condition (H), suitable nodes for storing tuples are scarce or non-existent. As evidenced by Table 4.3, we were still able to retain tuples with high load factors even in this situation. The reason is that robots shift the load from their storage memory to their routing memory. This is apparent in Figure 4.8c in which the number of tuples in storage memory normalized by the total number of tuples shows the difference between the use of h_C and h_{SP}. With the latter, most tuples remain in routing and bounce between robots with more free memory. This is not a desirable solution as it increases the



(a) Node identifiers δ_i distribution in category-based hashing.

(b) Node identifiers δ_i distribution in spatial hashing.

(c) Ratio of stored to routed tuples.

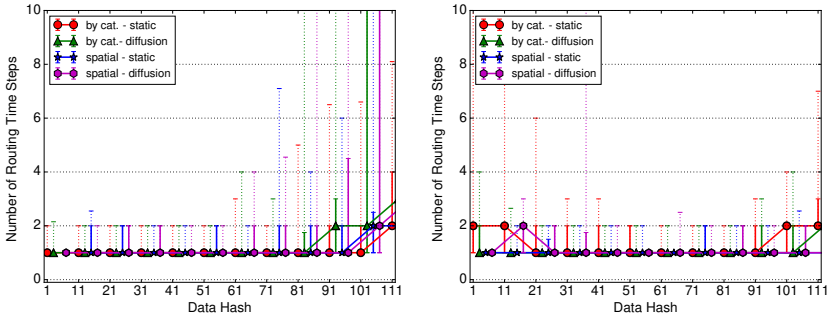
Figure 4.8: Performance of key partitioning with $N = 100$ robots.

communication overhead. However, it demonstrates a certain tolerance and seamless adaptation to inappropriate node partitioning. In practice, with a guess of the environment scale and typical distances, h_{SP} can be scaled so as to provide mapping to a range matching the node identifiers.

Routing Performance

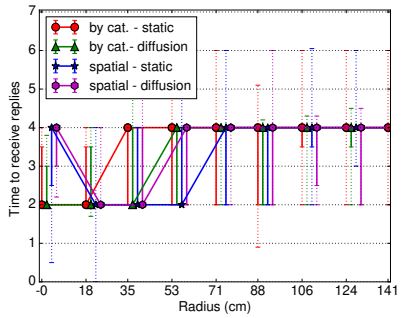
In our approach, routing mechanisms depend on the type of query. A write operation `put()` triggers a flooded `erase()` operation and a `store()` propagated through key-based routing (see Section 4.4.4). The timing for storing a tuple depends on how difficult it is to reach a suitable node given the tuple key. Figure 4.9 shows the median routing time with 10 and 100 robots for the `store()` operation. The time tends to increase with the key indicating that lower range keys find a match faster.

Read operations of any type generate a message flooded to all robots. Replies come back through gradient routing (see Section 4.4.3). Figure 4.9 reports the median duration between a robot emitting a spatial `get()` query and receiving the last reply to the query. This duration tends to increase with the network size and with the radius of the query.

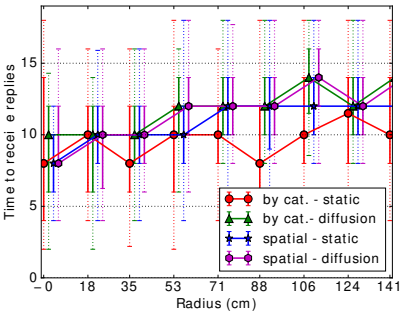


(a) $N = 10$.

(b) $N = 100$.



(c) $N = 10$.



(d) $N = 100$.

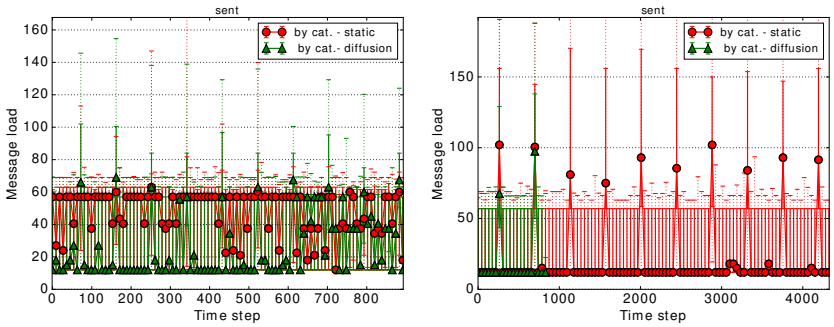
Figure 4.9: Number of time steps (100 ms each) for the completion of `store(k, v)` operations (a), (b) and `get(x, y, r)` queries (c), (d).

Message Load

The outgoing bandwidth was set to 570 bytes per time step for each robot, with a time step covering 100 ms. However, this allowance was rarely needed. Figure 4.10 shows the median bandwidth usage across simulations over time, which remains well below the limit.

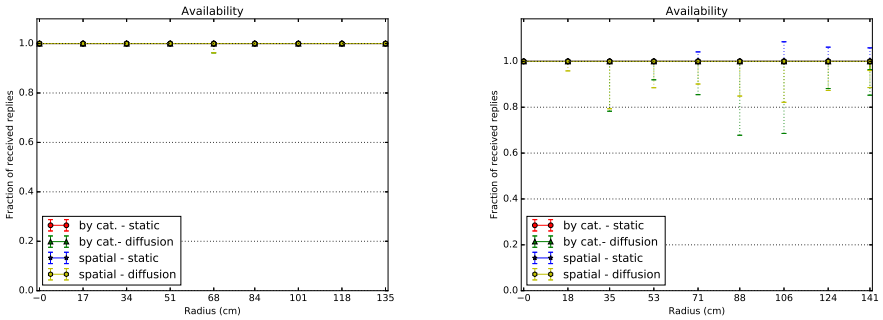
Delivery Rate

We recorded the delivery rate of replies, i.e. the fraction of received replies over the number of expected replies. As can be seen in Figure 4.11, the delivery rate remains close to 1.0 with very few outliers across configurations.



(a) $N = 10$ (b) $N = 50$

Figure 4.10: Bandwidth usage in bytes over time. Time is measured in time steps (100 ms).



(a) $N = 10$ (b) $N = 100$

Figure 4.11: Delivery rate of spatial queries.

4.6 Summary

We presented SwarmMesh, a distributed data structure for low-memory, low-bandwidth, highly mobile multi-robot systems. The main insight in the design of SwarmMesh is that the features that characterize the data items are key in deciding where to store the items. Our design is modular, in that the logic that governs data distribution can be chosen by the user. We proposed two methods of distributing storage responsibility, one based on the category of the data items (for applications in which certain data types are more important than others), and another based on the position of each data item. Our evaluation shows that SwarmMesh displays near-perfect levels

of data retention even for extremely high load factors, adaptively switching from static storage in the robot memory when load factors are low, to dynamic storage through frequent data exchange when load factors are severely high.

4.7 Open Problems

Future work involves using SwarmMesh for applications such as task allocation in dynamic environments and collaborative mapping. For the latter, a question that remains open is how to incorporate the size of the data items as a factor in the data redistribution logic.

Another possible extension of this work includes managing large data items that need to be split because of bandwidth limitations. This involves finding strategies for dividing and re-combining data packets in order to route this data to the destination node.

Finally, our approach lends itself to privacy and security considerations, whereby the decision on where to store certain data depends on the reputation of the robots [81].

Part III

Communication in Distributed Applications

Chapter 5

Data-Driven Federated Learning

5.1 Introduction

Federated Learning (FL) [82] is a recent approach to distributed machine learning that takes advantage of distributed data sets by partitioning learning on several machines. Data sets are partitioned either for excessive size, or because of the necessity to ensure data privacy among different data sources. In FL, individual clients collect data and calculate a local model update that is subsequently aggregated into a shared model by a dedicated server.

In a multi-robot setting, FL is a natural approach because it takes advantage of the inherently local fashion in which robots collect and process data. However, a practical approach to realizing FL in this setting is currently missing [28].

In this chapter, we conduct a study of the design space of FL solutions in a multi-robot setting. We compare two variants. In the first, inspired by the original FL idea, a server aggregates the model updates calculated by the robots. In the second, the server is replaced by a shared data structure.

In both cases, a central problem is how to synchronize the learning process.

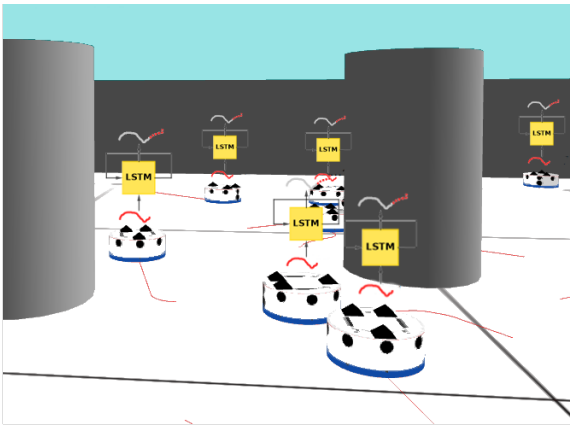


Figure 5.1: Federated Learning for collective trajectory forecasting in a multi-robot setting.

Data is typically collected at diverse rates across a team of robots. This affects the frequency at which robots contribute to the update of the shared model. To cope with this issue, we propose a data-driven approach in which only the robots that have collected sufficient data calculate a model update and share it with the rest of the system. Being dependent on the data flow, we named our fully distributed approach *Flow-FL*.

We envision our approach to be effective in scenarios in which data varies across both time and space. To validate this insight, we consider multi-robot navigation in densely occupied environments. In these environments, the intent of other moving entities may be unknown, and the density and motion patterns might vary over time. Recent work focuses on the creation of machine learning approaches to model motion trajectories that enable predictive navigation [83]. In our approach, the robots build a shared trajectory model using locally collected data.

The main contributions of this chapter are:

- We conduct an exploration of the design space of FL in robotics settings, comparing a variant in which models are aggregated on the server with a variant in which data is aggregated in a serverless, shared data structure;

- We apply the proposed approaches to a trajectory prediction problem and make our open-source federated dataset available for the research community;
- Using a centralized implementation as a baseline, we study the effects of staggered online data collection, variations in dataflow, number of participating robots, and time delays introduced by the decentralization of the framework in a multi-robot setting.

The chapter is organized as follows. We formalize the problem in Section 5.3. The design of our framework is presented in Section 5.4. We report the results of our performance evaluation in Section 5.5, and conclude the paper in Section 5.6.

5.2 Related Work

Early attempts at distributed machine learning were based on centrally stored datasets which were processed by multiple clients [84]. In these approaches, data partitioning occurs through assignment mechanisms that promote desirable features such as statistical independence and load balancing across the clients. Model updates are calculated on data that is physically copied from the server to the clients. Other early attempts applied local learning in settings in which naturally distributed data displayed analogous properties such as statistical independence and load balancing [85].

The inception of FL was motivated by the observation that data collected in mobile devices is often not statistically independent nor necessarily balanced across clients. In addition, copying data is often undesirable due to privacy concerns.

The seminal paper on FL by McMahan *et al.* [82] proposes `FedAvg`, which demonstrates the efficiency of averaging local model updates on a server when clients deal with non i.i.d. data and experience intermittent communication.

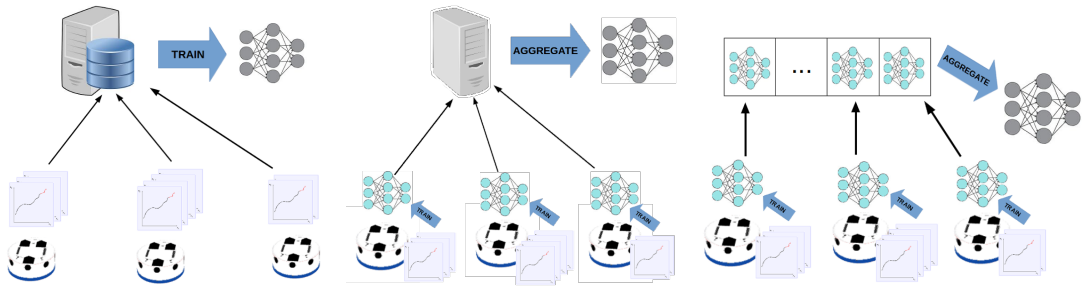
An important challenge in the implementation of FL is communication with the server. Particularly in distributed robotics, the presence of a server introduces the potential for a single point of failure that could endanger the success of the learning process. Therefore, decentralized approaches have surfaced which replace the server with a distributed mechanism. Notable works include using average consensus [86] and Bayesian methods [87, 88], which trade convergence time with resilience to individual failures. The approach of George *et al.* [89] offers convergence speed comparable to a server-based approach at the cost of assuming the communication topology of the clients to be fixed and predetermined.

Another important challenge is orchestrating the phases of local model update and global merging of the shared model. In traditional FL, the server takes care of this task by sending messages to the clients [82]. In a decentralized setting, Savazzi *et al.* [86] assume intermittent access to a centralized server and Lalitha *et al.* [87, 88] allow for asynchronous communication under strong connectivity constraints.

A final key problem is selecting the clients that contribute to the model update at each learning iteration. McMahan *et al.*'s research [82] reveals that the convergence performance of FL depends on the significance of the client updates. The significance is related to the quantity and age of the data that was used to calculate an update. Ideally, an update should be based on a large enough amount of recent data.

A recent trend in decentralizing FL is the use of conflict-free replicated data structures (CRDT) [90]. BAFFLE [91] is an approach based on a blockchain which offers resilience to intermittent communication and the possibility to avoid global consensus at the cost of increased computational cost and local storage requirements.

Our work furthers this line of research by using a lightweight CRDT called virtual stigmergy [77]. In *Flow-FL*, this structure is used both to schedule the learning iterations, and to store and merge the shared model.



(a) Centralized Machine Learning. (b) Federated Learning. (c) Distributed Federated Learning.

Figure 5.2: Learning frameworks.

5.3 Preliminaries

In this section we formalize the federated learning problem and justify our application scenario.

5.3.1 Federated Learning

In machine learning, the objective function is usually of the form:

$$\min_{\mathbf{\Theta}} L(f(x;\mathbf{\Theta}),y) := \frac{1}{n}\sum_{i=1}^{n} L(f(x^{(i)};\mathbf{\Theta}),y^{(i)}) \tag{5.1}$$

where the aim is to train a model $f(\cdot)$ with weights $\mathbf{\Theta}$, mapping an input $x \in \mathbb{R}^I$ to an output $y \in \mathbb{R}^O$. Using a training data set of n samples $(x,y) \in \mathbb{R}^{I\times O}$, we can adjust the weights of the model to minimize a loss function $L(\cdot)$ that expresses the error between the inferred output $\hat{y}$ and the true output y.

Federated Learning. FL [82] is an optimization problem that considers a modified version of Equation 5.1 where the training examples are now stored across K clients and a global server attempts to minimize a loss function obtained from aggregated local weights (Figure 5.2b). In our case, clients refer to robots that collect data. The global function that is minimized is the summation of the local losses

obtained on the k-th robot, $L_k(\mathbf{\Theta})$, weighted by the number of samples observed n_k over the total number of samples n in the dataset:

$$\min_{\mathbf{\Theta}} L(\mathbf{\Theta}) = \min_{\mathbf{\Theta}} \sum_{k=1}^{K} \frac{n_k}{n} \times L_k(\mathbf{\Theta}), \tag{5.2}$$

The local loss of the robot $L_k(\mathbf{\Theta})$ is similar to that in the traditional machine learning case,

$$L_k(\mathbf{\Theta}) = \frac{1}{n_k} \sum_{h=1}^{n_k} \ell(x_h, y_h; \mathbf{\Theta}) \tag{5.3}$$

where $\ell(x_h, y_h; \mathbf{\Theta})$ is the loss of the predicted model over the n_k samples (x_h, y_h) observed by the robot k, using global model parameters $\mathbf{\Theta}$.

Assumptions. FL is typically based on a number of key assumptions on the data: *(i)* It is non-i.i.d. (independent and identically distributed); *(ii)* It is stored across several clients; and *(iii)* It is partitioned in an imbalanced manner, resulting in clients that handle more data than others. In addition, communication among clients is assumed to be intermittent. In traditional FL the presence of a server ensures synchronization, however serverless settings are also possible. In this chapter, we maintain all of the above assumptions. In particular, we study the effect that the presence or absence of a server has on the performance of the learning process.

5.3.2 Application: Trajectory Forecasting

In this work we consider trajectory prediction as an application example of *Flow-FL*. Trajectory forecasting is typically conducted with pedestrian data. However, existing literature on trajectory forecasting [92, 93, 94] focuses on pedestrian data collected from a single point of view which is usually an overhead camera. As such, the available datasets are not easily amenable to an FL setting. To the best of

our knowledge, there are no federated datasets of trajectories collected by multiple robots. For this reason, we generated a novel federated dataset from artificial navigation data in four different multi-robot settings (see Section 5.4.2). In future work, we will use real robots to collect motion data of real dynamic obstacles such as pedestrians.

Trajectory forecasting is a compelling problem for machine learning applications. This problem is about creating a model that allows robots to predict the trajectories of dynamic obstacles nearby within a short time horizon. Research has shown that navigation is significantly more efficient when using a machine learning model than with purely reactive methods, such as ORCA and RVO [95, 96]. While numerous neural networks have been benchmarked [92], a simple Long Short-Term Memory (LSTM) model [97] has been shown to yield the lowest Average Displacement Error on the standard datasets. Therefore, we use this model architecture in our evaluation.

5.4 Methodology

5.4.1 System Design

In a traditional FL setting, a server communicates with data-holding clients to enable training of a global ML model (see Figure 5.2b). The server does not aggregate data, as opposed to a fully centralized approach (see Figure 5.2a). However, in traditional FL, the server has the important roles of: *(i)* orchestrating learning rounds periodically by selecting a subset of learners and sending them model parameters; *(ii)* aggregating the results of a round of learning into a global model. In our approach, depicted in Figures 5.2c and 5.3, we replace the central server with a distributed algorithm. The scheduling of learning rounds is data-driven, and it

happens when a sufficient number of robots have collected enough data. To keep the rounds sequential and distinct, a global state is maintained in a gossip-based shared memory. Model weights are written to this shared memory at the end of each round. The aggregation happens in the subsequent round, when robots with enough data pull the weights from the shared memory to instantiate their local models.

Application scenario. We consider a scenario with K moving robots that communicate and track each other's relative positions within a limited range. We tackle the problem of learning to predict the robot trajectories for a certain time horizon (see Figure 5.1 for examples of the robot trajectories). In our setup, robots are moving, collecting data and performing learning concurrently. This is in contrast with existing frameworks for multi-robot learning, which require some sort of synchronicity in switching through these activities. To the best of our knowledge, this is an item of novelty in our framework. In our setup, the motion behavior is determined by a pre-programmed controller. We chose well-studied swarm behaviors such as flocking [35], foraging [36], and diffusion with obstacle avoidance [34]. The data collected by the robots consists of the spatial coordinates of neighboring robots expressed in a fixed frame local to the sensing robot. A training sample is one continuous trajectory recorded for a fixed duration at regular time steps. Each robot builds its own dataset over time from its local observations.

State machine of the learning framework (Figure 5.3). The operation of the robots is organized in two tiers. The learning tier runs separately from the application tier that performs the swarm behavior. The robots start in IDLE state. Each robot has knowledge of the model architecture, but it starts with the same random weights and no data.

The transition to the first learning round is conditioned by the flow of data. When a robot collects a certain *quota* of training samples, it marks itself as ready

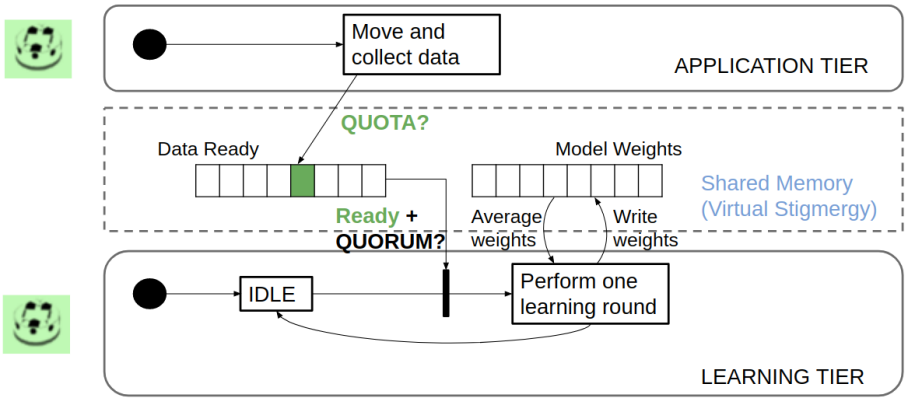


Figure 5.3: The state transition of the learning tier for the green robot is conditioned by the data flow of its application tier and a global state tracked in the shared memory.

in the shared memory. When a sufficient number of robots have marked themselves ready, we say that a *quorum* has been reached and the ready robots collectively transition to the learning round. These robots, the *learners*, perform local training on the data they had gathered and forget their samples.

The next transition brings the robots back to the IDLE state in the learning tier. The transition happens after all the learners finish sharing their new ML model weights with the rest of the swarm. Each learner shares its weights after performing one training epoch on its samples.

From now on, the transitions out of the IDLE state proceed in a data-driven fashion as the first transition. However, the learning rounds start differently: in this case, the learners must first retrieve and aggregate the most recent weights from the shared memory.

Shared memory. The shared memory has two purposes: (i) It holds a global list of ready robots which is used to synchronize state transitions when the quorum of ready robots is achieved; and (ii) It stores the updated model weights in a dedicated global list. We implemented data sharing through Virtual Stigmergy (VS) [77]. VS is

a lightweight, distributed tuple space designed to share a collection of `(key,value)` pairs. A local Lamport-clock-stamped copy of each tuple is stored on each robot. This copy is updated upon both `read()` and `write()` operations through network flooding. In this way, while the network topology changes, the data structure is kept up-to-date.

Transition mechanism. The state transitions are achieved through a *barrier* mechanism making use of the VS. We outline the steps for this count-based consensus protocol in Algorithm 3.[1]

ML model architecture. The ML model used for the application is a standard architecture for time series prediction. We use one Long Short-Term Memory (LSTM) layer [97] with a hidden dimension of 16 without returning sequences, followed by a dense layer and Dropout of 0.2 . The network looks at a history of 32 prior (x, y) observations (corresponding to 3.2 s) and directly outputs a prediction for the next 48 time steps (4.8 s). We used a Mean Squared Error (MSE) loss MSE $= \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i - \tilde{\mathbf{x}}_i)^2$ as a training metric that is then optimized using RMSProp [98].

ML model aggregation. Similar to the implementation in `FedAvg` [82] the *learners* update the global list with their learned weights after performing a gradient descent step on local data. During the next time step, the new weights for the server are the average of all the learned weights on the list weighted by the number of samples encountered by each *learner*. McMahan *et al.* also found that, for large local epochs, `FedAvg` can plateau or diverge. They recommend either using fewer local epochs or decaying the number of local computations over time. We do not study this aspect, and let the *learners* perform only a single local epoch per iteration and flush this data before collecting new data during the next iteration.

[1] A first implementation was proposed in `https://the.swarming.buzz/ICRA2017/barrier/`.

Algorithm 3 Barrier - robot k, barrier VS β_k, neighbors $\mathcal{N}_k$

procedure BARRIER_SET
    initialize barrier $\beta_k \leftarrow list()$ for robot k
    robot state $\leftarrow$ **barrier_wait**
    activate ON_BARRIER_UPDATE
procedure ON_BARRIER_UPDATE($\mathcal{N}_k$)
    receive $\{i\}_{i \in \mathcal{N}_k}$ ▷ RX from neighbors
    if $i \notin \beta_k$ **then**
        $\beta_k \leftarrow \beta_k \cup i$
        send (i) ▷ TX to neighbors
procedure BARRIER_READY($\mathcal{N}_k$)
    receive $\{i\}_{i \in \mathcal{N}_k}$ ▷ RX from neighbors
    $\beta_k \leftarrow \beta_k \cup k$
    if element_count(β_k) $\geq$ threshold **then**
        robot state $\leftarrow$ **next_state**
    send $(\Delta\beta_k)$ ▷ TX to neighbors

5.4.2 Datasets

To the best of our knowledge, this contribution is the first to provide an open-source federated dataset of swarm motion with information on the communication graph. We generated multiple synthetic datasets of swarm motion across four distinct behaviors using ARGoS, a realistic physics-based simulator [99].

Data format. Each behavior dataset consists of:

- A *trajectory file* that records the robot's id, neighbor id, and position across time `(robot id, neighbor id, t, x, y, z)`. Each robot records neighbor trajectories for 50,000 time steps (5,000 s). A trajectory sample, within a setup, is 100 time steps (10 s) long and is separated from other samples by an *end-of-line* character. Each trajectory is expressed in the local reference frame of the robot at the start of the sample recording. This is a fixed frame of reference independent of subsequent robot motion.

- A *communication graph file*, structured as `(t, robot id, neighbor id)`,

that logs information about the neighboring robot IDs that are in range at every time step. This information encodes the communication graph at every time step. The communication graph file is more complete than the trajectory file because when recording the trajectory file, we drop interrupted trajectories.

Parameter setting. The experimental settings are kept consistent across datasets as shown in Table 5.1. The settings include: *(i)* the trajectory sampling period; *(ii)* additive noise for positional data on each robot neighbor sampled from a normal distribution; and *(iii)* wheel actuation noise, also sampled from a normal distribution. The noise parameters are rounded estimates from realistic samples taken from real-world Khepera robots. We executed the experiments for swarms with $K = \{15, 60\}$ robots to enable the study of the effect of different swarm sizes.

Table 5.1: Experiment settings

Parameter	**Value**
Trajectory duration	10 s
Communication range	2 m
Sensing range	2 m
Sensing noise $n_k(t)$	$n_k(t) \sim \mathcal{N}(0, 0.01)$ m
Drive bias e_k	$e_k \sim \mathcal{N}(0, 0.0001)$ m

Behavior types. We evaluate our methodology with four different swarm behaviors [100] (see Figure 5.5): *(i)* Obstacle-avoidance [34] (Figure 5.4 top row) in a dense environment with uniformly distributed static obstacles; *(ii)* Foraging [36] (Figure 5.4 bottom row) for resources in which robots decide whether to explore or stay in the nest according to energy considerations; *(iii)* Phototaxis and flocking based on artificial physics [35], with a light whose position is changed to prevent stagnation; and *(iv)* A mixed behavior in which robots perform one of the previous behaviors depending on their location in the environment. The datasets and corresponding simulation videos can be found at:

https://www.nestlab.net/doku.php/papers:mrs_fl_dataset. Table 5.2 provides the total number of samples in each dataset as well as statistics about the distribution of samples between robots. The table reveals diversity across the behaviors in the number of samples collected, both total and per robot, also considering total time and 10-minute windows. In particular, the standard deviation shows how different behaviors result is different levels of imbalance in the number of samples collected by the robots.

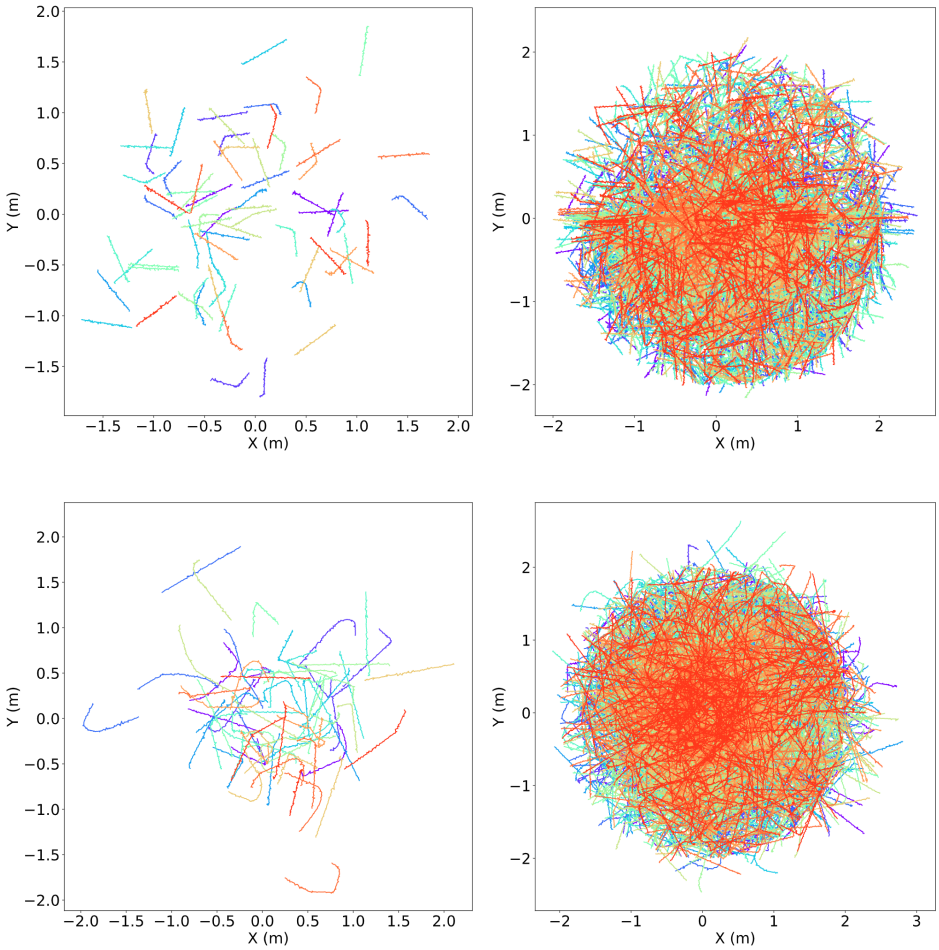
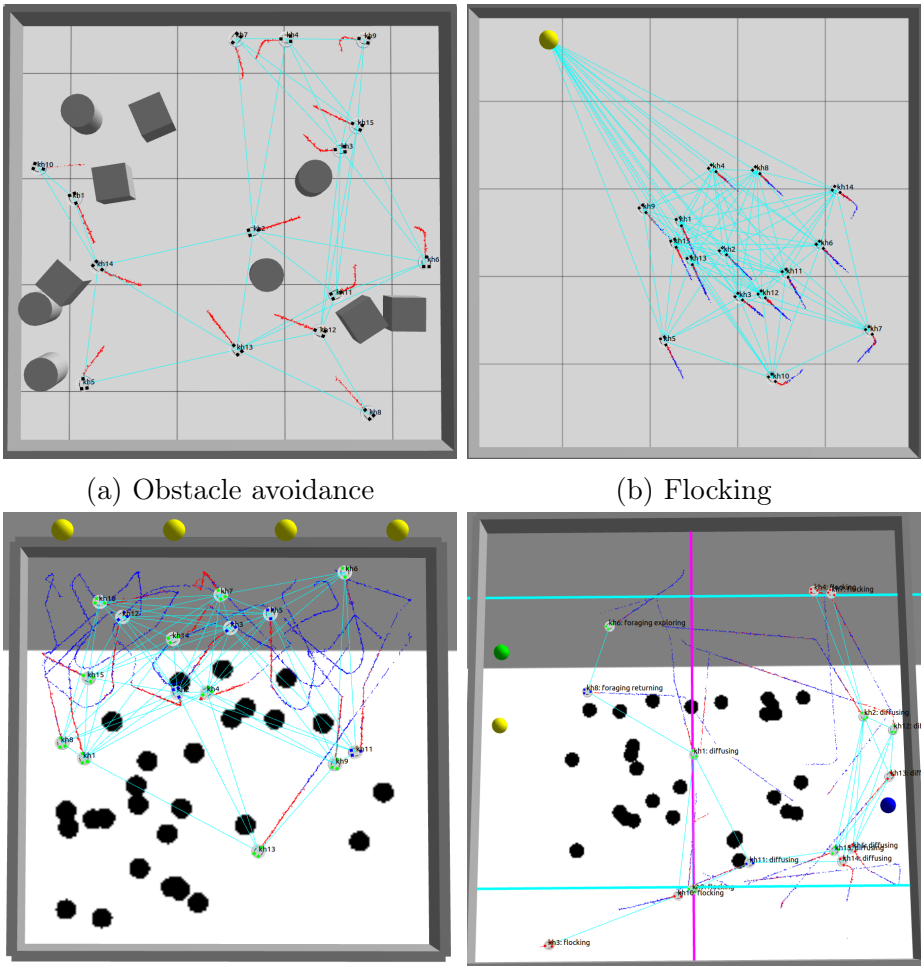


Figure 5.4: Avoidance (top) and foraging (bottom) for $K = 15$. Left to right: increased number of displayed samples.



(a) Obstacle avoidance (b) Flocking

(c) Foraging (d) Mixed behaviors

Figure 5.5: Swarm behaviors in ARGoS simulator.

Table 5.2: Statistics for swarm motion federated datasets.

Dataset	Robots	Samples	Samples/robot		Samples/robot /10 min	
			Mean	Stdev	Mean	Stdev
Avoidance	15	21,227	1,415	206	169	40
	60	216,582	3,610	397	429	74
Flocking	15	49,009	3,267	427	390	78
	60	333,111	5,552	436	659	112
Foraging	15	35,854	2,390	46	284	19
	60	187,304	3,122	65	371	25
Mixed	15	30,627	2,042	290	242	66
	60	174,863	2,914	89	346	29

5.5 Evaluation

5.5.1 Parameters of Interest

Quorum and quota. The main parameters of interest in our empirical study are the *quorum* of learners and the *quota* of data. These parameters condition the transition to a learning state for a subset of robots at times dictated by the data collection flow (see Figure 5.3). We set the quorum as a fraction q_F of the total number of robots. We perform our study with datasets from experiments with a fixed duration of 5,000 s. Quorum and quota determine the total number of learning rounds in each study configuration as they split the data in time.

Number of robots. We also study the influence of the *number of robots* by using 15- and 60-robot datasets. We expect this parameter to act on different aspects of multi-robot communication and learning as it changes: *(i)* the total amount and frequency of data collected, which changes the time between learning rounds (Section 5.5.3); *(ii)* the robot communication network size and topology, which change the latency of the shared data structure and affect the timing of learning rounds (Section 5.5.3); *(iii)* the partitioning of federated data across clients, which

influences the convergence rate (Section 5.5.2).

Federated datasets. We use multiple *federated datasets* detailed in Section 5.4.2 with different behaviors (see Figure 5.4). We provide and compare results for these 8 datasets.

Learning hyperparameters. We set the learning *hyperparameters* for the local machine learning model as detailed in Section 5.4.1. In terms of standard FL hyperparameters, we set number of local epochs E to 1. Increasing E reduces communication overhead at the cost of increased individual computational load. McMahan *et al.* [82] provide an empirical study of the effect of this parameter. They show that high values of E can lead the FL algorithm to diverge. In the same paper, McMahan *et al.* also vary the quorum fraction but refer to it as client fraction. In this chapter, we focus on the effect of controlling the data flow through q_F and quota rather than changing the amount of local computation through E.

Dataset split. To separate our data into training/validation/test samples, we proceed in the following way: we take 80 % (first 4,000 s) of the experiment data for training and validation and keep the remaining for testing; within a learning round, each learner splits the data into a set of training samples (the first 80 % of trajectories), and validation samples (the last 20 % of trajectories). We verified that behaviors do not change at the end of the experiment so as to have an appropriate testing set.

Table 5.3: Evaluation parameters

Parameter	**Value**
Number of robots K	$\{15, 60\}$ robots
Quorum fraction q_F	$\{0.2, 0.6\}$
Quorum	$q_F \cdot K$ robots
Quota	$\{20, 60\}$ samples
Local epochs E	1

5.5.2 Convergence Analysis

An important aspect of our FL framework is the effect of scheduling rounds according to quorum and quota on the learning convergence. We want to study the following aspects empirically across several datasets:

- which (q_F, *quota*) configuration requires the *least learning rounds* to achieve convergence. Reducing the number of learning rounds reduces communication rounds thereby decreasing communication overhead;
- which (q_F, *quota*) configuration requires data spread across the *least time steps* to achieve convergence. Reducing the number of steps gives us a final model earlier on in the experiment;
- which (q_F, *quota*) configuration gives us the *best trade-off* between the two above situations.

Varying quorum and quota is effectively re-partitioning the data in time within the same dataset. With respect to learning, this affects the rate of model updates as well as the number of participating clients $\bar{K}$ and the number of local examples n_k used for the update at each round.

We study the effect of (q_F, *quota*) on the federated validation loss $L(\mathbf{\Theta})$ from Equation 5.2. To make the comparison fair with *Flow-FL*, we implemented a data-driven FL approach with a server scheduling rounds according to q_F and *quota*, i.e., starting a round as soon as $q_F \cdot K$ robots have a *quota* of samples. In the distributed version Flow-FL, the learning rounds are scheduled the same way, but they occur with a delay after the quorum/quota condition is met. This delay is due to the latency imposed by the update of the shared data structure. Thus, a different number of learners may qualify at the same time. To compare convergence

in a consistent way, we define the stopping round as the round where the windowed average of loss changes less than a threshold of 0.0001. The averaging window was set to 5 rounds.

Discussion. Figure 5.6 shows the federated validation loss curves. We also show the centralized validation loss as a baseline that uses all the data collected by the end of the experiment at once. To compare the loss over iterations, we show epochs for the centralized loss and learning rounds for the federated loss. We include as many epochs for the baseline as the number of learning rounds in *Flow-FL*. Table 5.4 reports the final loss value and stopping points for all the behaviors. The final loss is similar across configurations, but it tends to increase as the total number of iterations decreases. With higher (q_F, $quota$), we have fewer learning rounds because it takes more data to move to the next training round. We also note fewer oscillations of the learning curve with higher (q_F, $quota$).

Stopping rounds and times. Table 5.4 shows the lowest stopping rounds and times across configurations in bold:

- We see that, while changing across behaviors, the lowest number of learning rounds occurs with higher thresholds of (q_F, $quota$) than the minimum $(0.2, 20)$;
- We get lower numbers of time steps at stopping with lower thresholds for (q_F, $quota$). However, the stopping criterion sometimes selected an early stopping round with some oscillations occurring later in the curve. Those instances are denoted by an asterisk.
- The best trade-off between number of rounds and time steps is at (q_F, $quota$) $= (0.2, 60)$.

Table 5.4: Convergence data across quorums and quotas for different behaviors ($K = 60$).

(q_F, **quota**)		**(0.2 20)**	**(0.2, 60)**	**(0.6, 20)**	**(0.6, 60)**
Flocking					
Validation loss	C	0.002	0.002	0.002	0.002
	FL	0.002	0.001	0.001	0.001
	Flow-FL	0.002	0.002	0.002	0.002
Stopping round	FL	17	8	16	**6**
	Flow-FL	17	**8**	13	**8**
Stopping time(s)	FL	**134***	157	297	292
	Flow-FL	**136***	157	247	402
Foraging					
Validation loss	C	0.011	0.011	0.012	0.016
	FL	0.014	0.013	0.017	0.014
	Flow-FL	0.012	0.013	0.016	0.015
Stopping round	FL	21	**7**	13	8
	Flow-FL	15	10	16	**7**
Stopping time(s)	FL	**257**	**257**	448	731
	Flow-FL	**192**	326	562	636
Avoidance					
Validation loss	C	0.003	0.004	0.005	0.007
	FL	0.006	0.004	0.006	0.007
	Flow-FL	0.005	0.006	0.007	0.007
Stopping round	FL	12	9	15	**8**
	Flow-FL	16	10	15	**8**
Stopping time(s)	FL	**149***	250	421	592
	Flow-FL	**184***	271	435	599
Mixed					
Validation loss	C	0.006	0.007	0.009	0.012
	FL	0.008	0.008	0.010	0.010
	Flow-FL	0.010	0.009	0.012	0.011
Stopping round	FL	20	**8**	19	**8**
	Flow-FL	16	**8**	15	**8**
Stopping time(s)	FL	**232**	242	611	705
	Flow-FL	**192**	241	503	711

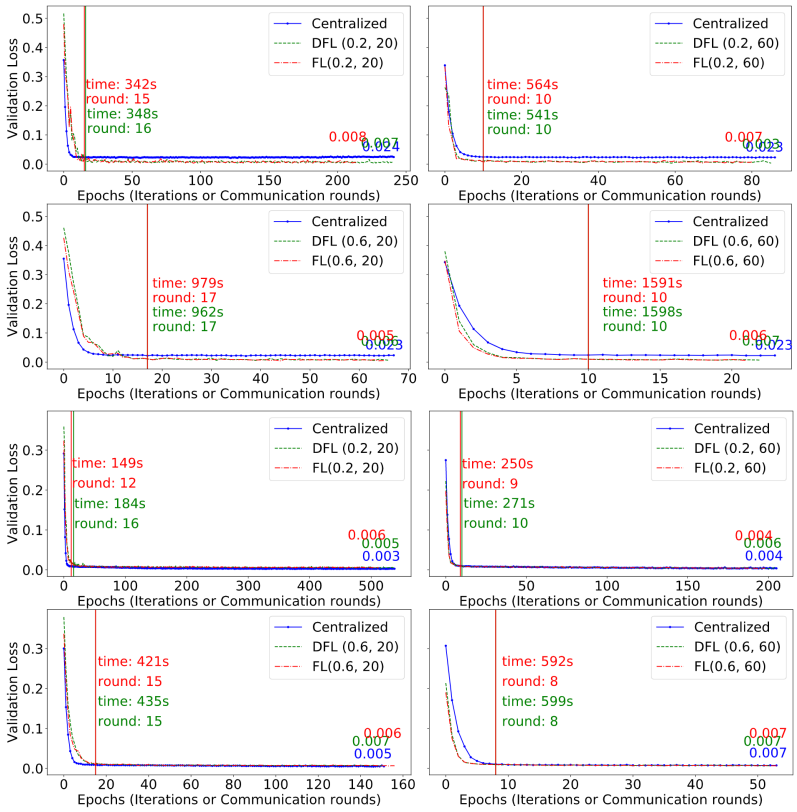


Figure 5.6: Validation loss $L(\mathbf{\Theta})$ for the avoidance behavior with 15 (left), 60 (right) robots (DFL=*Flow-FL*).

5.5.3 Learning Round Timing

The scheduling of learning rounds is data-driven both in our version of FL and in *Flow-FL*. The time between learning rounds increases when we increase either the learner quorum or the sample quota. Figure 5.7 shows the time between rounds over the duration of the experiment. This time depends on the flow of data in the experiment and the length of the time series we consider as samples. The graph shows that, for a given (quorum,quota) setting, the inter-round duration is similar between FL and *Flow-FL*. This is due to the short time spent by the robots in the barrier with *Flow-FL*. With $K = 15$, the average time spent in the *barrier* is 15 time steps (1.5 s) and, with $K = 60$, it is 51 time steps (5.1 s). A wider empirical study of the latency of virtual stigmergy is reported in [77].

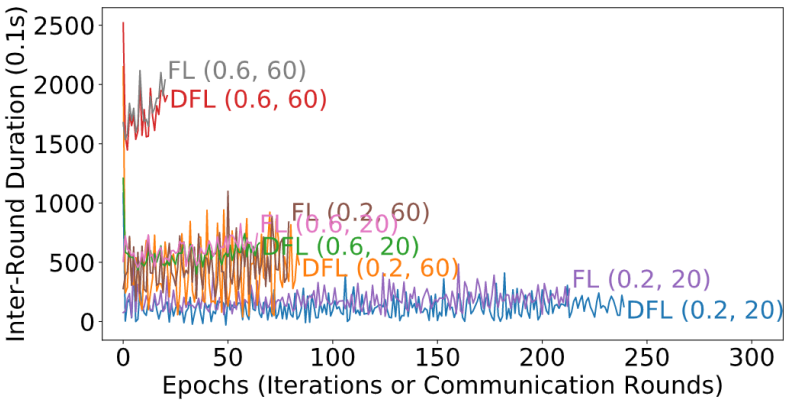


Figure 5.7: Timing for avoidance with 15 robots (DFL=*Flow-FL*).

5.5.4 Prediction Quality

We quantify the prediction quality of trajectory forecasting based on two metrics: *(i)* The Average Displacement Error (ADE) in Equation 5.4, that is a summation of the L2-norm between the ground truth $\mathbf{y_{i,t}}$ and the predicted trajectory $\tilde{\mathbf{y}}_{\mathbf{i,t}}$ over the predicted horizon T for all trajectory samples N_{traj}; *(ii)* The Final Displacement Error (FDE) in Equation 5.5 which is the summation of the L2-norm of the final positions between the predicted $\tilde{\mathbf{y}}_{\mathbf{i,T}}$ and the ground truth $\mathbf{y_{i,T}}$ over all trajectory samples.

$$\text{ADE} = \frac{1}{N_{\text{traj}} \cdot T} \sum_{i=1}^{N_{\text{traj}}} \sum_{t=1}^{T} \|(\mathbf{y}_{i,t} - \tilde{\mathbf{y}}_{i,t})\| \tag{5.4}$$

$$\text{FDE} = \frac{1}{N_{\text{traj}}} \sum_{i=1}^{N_{\text{traj}}} \|(\mathbf{y}_{i,t=T} - \tilde{\mathbf{y}}_{i,t=T})\| \tag{5.5}$$

We report the ADE, FDE metric results in Table 5.5 to 2 decimal places since this corresponds to centimeter-level granularity. We maintain consistency across the evaluations in the case of centralized training by setting the number of epochs trained to the number of rounds performed by *Flow-FL* for the (quorum, quota) pair. We note that trajectories are forecast similarly by the three methods across

Table 5.5: Trajectory reconstruction across quorums and quotas for different behaviors. ($K = 60$)

($\mathbf{q_F}$, **quota**)		**(0.2, 20)**	**(0.2, 60)**	**(0.6, 20)**	**(0.6, 60)**
Flocking					
FDE (m)	Centralized	0.08±0.04	0.08±0.04	0.08±0.04	0.09±0.04
	FL	0.08±0.04	0.08±0.04	0.09±0.05	0.08±0.04
	Flow-FL	0.09±0.05	0.08±0.04	0.09±0.05	0.09±0.04
ADE (m)	Centralized	0.05±0.03	0.05±0.03	0.05±0.03	0.05±0.03
	FL	0.05±0.03	0.05±0.03	0.05±0.03	0.05±0.03
	Flow-FL	0.06±0.03	0.05±0.03	0.06±0.03	0.05±0.03
Foraging					
FDE (m)	Centralized	0.22±0.11	0.23±0.11	0.23±0.11	0.27±0.13
	FL	0.25±0.12	0.23±0.12	0.27±0.13	0.25±0.12
	Flow-FL	0.24±0.12	0.23±0.12	0.27±0.13	0.25±0.12
ADE (m)	Centralized	0.11±0.05	0.12±0.05	0.12±0.06	0.14±0.06
	FL	0.13±0.06	0.12±0.06	0.14±0.07	0.13±0.06
	Flow-FL	0.13±0.06	0.12±0.06	0.14±0.07	0.13±0.06
Avoidance					
FDE (m)	Centralized	0.11±0.05	0.13±0.05	0.15±0.06	0.19±0.06
	FL	0.17±0.06	0.14±0.05	0.19±0.06	0.18±0.06
	Flow-FL	0.16±0.06	0.14±0.05	0.19±0.06	0.18±0.07
ADE (m)	Centralized	0.06±0.02	0.07±0.03	0.08±0.03	0.10±0.04
	FL	0.09±0.04	0.07±0.03	0.10±0.04	0.10±0.04
	Flow-FL	0.08±0.03	0.07±0.03	0.10±0.04	0.10±0.04
Mixed					
FDE (m)	Centralized	0.15±0.10	0.17±0.11	0.18±0.12	0.21±0.15
	FL	0.21±0.14	0.18±0.13	0.24±0.17	0.21±0.15
	Flow-FL	0.20±0.14	0.18±0.12	0.24±0.17	0.21±0.14
ADE (m)	Centralized	0.08±0.05	0.09±0.05	0.10±0.06	0.11±0.07
	FL	0.11±0.07	0.09±0.06	0.13±0.09	0.11±0.07
	Flow-FL	0.11±0.07	0.09±0.06	0.13±0.09	0.11±0.07

all four behaviors with no notable differences. We identify that, in general, reconstructing flocking trajectories which are goal-oriented to a light source is much easier than reconstructing foraging or avoidance trajectories which need additional context information from neighbors (such as distance between neighbors in the case of obstacle avoidance, or location of resources/the nest for foraging). The generated output trajectories are shown in Figure 5.8. Data such as a robot's motion model, higher-derivative information (such as velocity), or a particular agent's goals would help robustly predicting turning or in some cases matching the distance traveled by a robot (even though the orientations are predicted accurately). These improvements are out of the scope of our work.

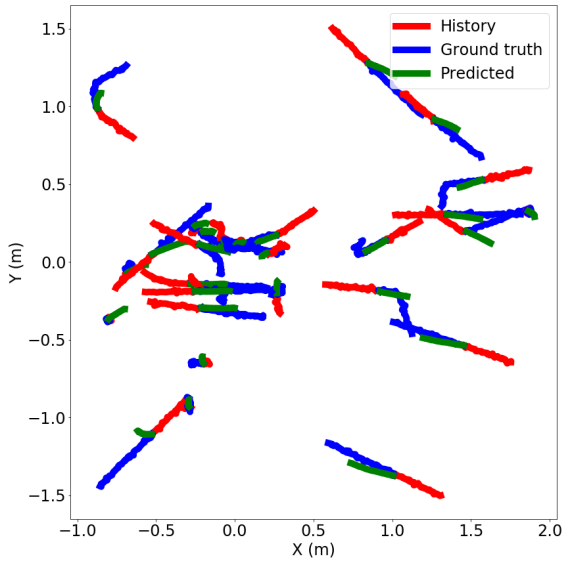


Figure 5.8: Fifteen trajectories predicted using *Flow-FL* for one robot with mixed behaviors in a swarm for $K = 60$.

5.6 Summary

In this work, we explore the design space of Federated Learning in a robotics setting. Our study includes two versions of Federated Learning (one server-dependent and

one serverless), and one centralized approach as our baseline. We provide a practical realization of fully distributed Federated Learning, *Flow-FL*, in a multi-robot setting. We propose a way to schedule model updates based on data flow by considering two parameters: the *quota*, i.e., the minimum number of data samples for a robot to qualify for a model update, and the *quorum*, i.e., the minimum number of robots to start a learning round.

We studied the role of several parameters of practical relevance, such as staggered online data collection, number of participating robots, and time delays introduced by decentralization. As we envision our approach to be useful in learning dynamic spatio-temporal datasets, we considered a well-known case study with compelling dynamics in both space and time: trajectory forecasting. Due to the lack of usable datasets in the literature, we created the first federated dataset from artificial data collected from a representative set of multi-robot behaviors.

5.7 Open Problems

In order to enable the deployment of *Flow-FL* in a real-world setting, its performance should be evaluated on real-world pedestrian data collected with a team of robots.

While we studied the effect of the (quorum, quota) configuration on the learning convergence empirically, a formal analysis of the effect of these parameters would help inform their choice. In particular, this analysis would determine the conditions for which the trends observed across our several datasets hold.

In addition, the role of the communication topology in keeping communication delays limited when the number of robots increases, has yet to be determined.

Finally, the effect of aggressive communication loss on the convergence of *Flow-FL* remains to be quantified and analyzed.

Chapter 6

Collective Semantic Annotation

6.1 Introduction

Decentralized collective perception is one of the main activities that a swarm of robots must perform. A major challenge in collective perception is dealing with partial and uncertain knowledge. To deploy robots in safety-critical applications, such as disaster recovery or autonomous driving, researchers strive to increase the reliability of the information used for decision-making. In the context of resource-constrained robot swarms, significant effort has been devoted to solving this problem, with particular focus on the best-of-n formulation [101], [102].

In this chapter, we deal with a collective perception scenario related to, but more complex than, the best-of-n formulation. We assume that individual robots perform observations, from which they individually derive annotations on the environment through a machine learning algorithm. The algorithm, however, might produce incorrect annotations. The aim of our work is to take advantage of the fact that multiple robots can combine individual annotations to derive a set of more accurate consolidated annotations.

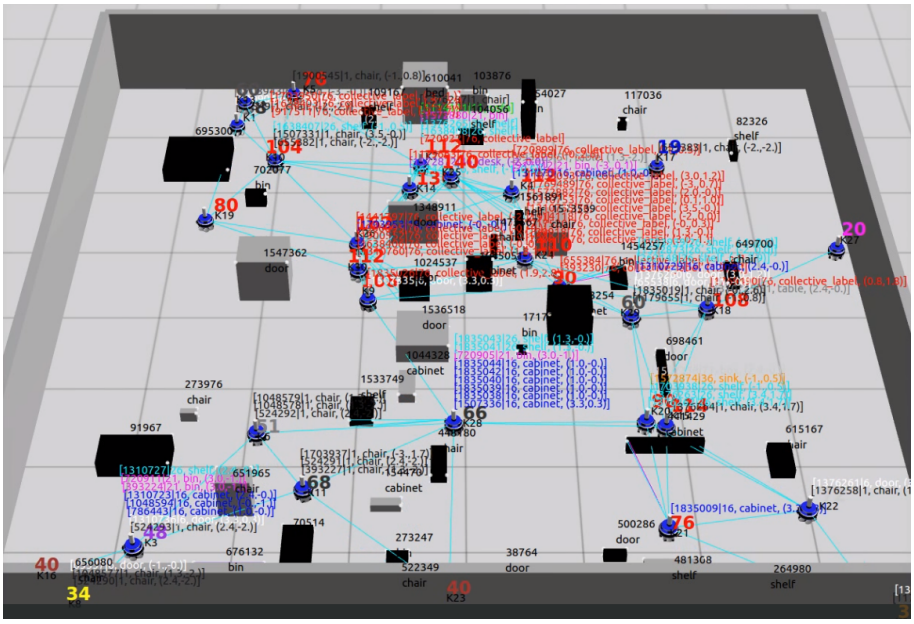


Figure 6.1: Collective semantic annotation application with label consolidation.

To showcase our approach, as shown in Figure 6.1 and Figure 6.2, we consider the case in which the robots must collectively construct and store a coherent set of semantic annotations on features found in the environment, such as pieces of furniture, trees, or doorways. To consider low-memory constraints, we assume that the objects are internally stored as bounding boxes, rather than through more detailed representations such as voxels or graphs. These annotated bounding boxes constitute the essence of a distributed *semantic map* [103] of the environment. This problem is different from the best-of-n problem. In the latter, the robots must agree on a single choice for the entire swarm; in our problem, the robots must collectively construct a coherent set of annotations.

Semantic mapping poses interesting challenges in the context of resource-constrained collective robotics. Constructing a semantic map entails the collection and storage of large amounts of data. This data is typically processed by the robots individually at first. The result is then shared and fused with the results of other robots to produce a coherent representation of the environment. An effective approach to solve

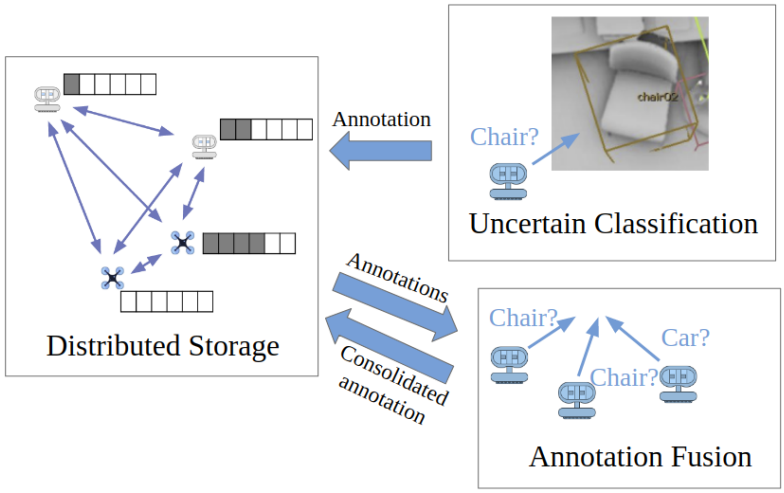


Figure 6.2: Collective semantic annotation application with annotation consolidation.

these problems is capable of coping with low memory availability, mobility, noise, uncertainty, and bandwidth limitations [104, 105, 106].

Our approach to semantic mapping draws inspiration from research in ensemble learning [107]. Ensemble learning reduces the variability of individual models through information fusion [108]. In ensemble learning, fusion is achieved through bagging and stacking techniques that aggregate models, typically classifiers, by combining their outputs through an averaging process or through a meta-model trained for that purpose [107, 109].

This chapter presents two main contributions. First, we present an algorithm to store localized semantic annotations across a swarm of mobile robots, under the form of a shared global memory. Second, we propose an approach to online fusion of uncertain semantic annotations, formalizing the problem and deriving a solution from first principles. We consider a scenario in which the user has already trained a multi-class classifier on an extensive dataset with multiple annotations of various objects [110]. This classifier is deployed on each robot and used to identify objects for the map, which are shared with the rest of the swarm. The semantic

map is constructed by consolidating multiple observations of the same objects into single ones. Through a voting mechanism, our approach copes with the fact that classifiers are imperfect and might mislabel objects. The final result of our approach is a semantic map whose accuracy is superior to that achievable by any individual robot alone. We also provide a closed-form equation to assess the accuracy of a consolidated annotation as a function of the number of votes and the individual classifier accuracy.

This chapter is organized as follows. In Section 6.2 we discuss related work. The components of our framework are described in Section 6.4. We report the results of our performance evaluation in Section 6.5, and conclude the chapter in Section 6.6.

6.2 Related Work

Ensemble learning. In the context of ensemble learning, Kuncheva summarizes existing methods for combining the outputs of classifiers and provides proof of their optimality under certain assumptions. [109]. In particular, the book chapter presents plurality voting, Bayesian combination (Naive Bayes), and multinomial methods ("Behavior Knowledge Space"). The plurality voting approach is the simplest and gives the optimal accuracy in combining classifier outputs when: *(i)* the classifiers give independent outputs for a given class label, *(ii)* the classifiers give any incorrect class label with the same probability, and *(iii)* all classes have the same prior probabilities. The Bayesian combiner only requires assumptions *(i)* and *(ii)* to be optimal. In practice, the amount of data available to tune the combiner also plays a role in the performance of an approach. In particular, the more complex the method the more data is needed to estimate its parameters. Kuncheva concludes that "the success of a particular combiner will depend partly on the validity of the assump-

tions and partly on the availability of sufficient data to make reliable estimates of the parameters" [109].

Best-of-n problem. Previous work in swarm robotics has tackled collective classification problems for one global environmental property. These problems have commonly been referred to as best-of-n [111]. Valentini *et al.* proposed a honeybee-inspired approach for robots to collectively decide whether a black and white colored environment has a majority of white or black tiles [101]. Ebert *et al.* applied Bayesian estimation to the same problem with the added explicit group-wide agreement on the output [102]. Robust formulations have also been studied in which robots might be affected by noise [112] or act in an adversarial manner [113]. As discussed in the introduction, semantic mapping is a more complex problem, in that it can be seen as a repeated best-of-n problem in which the annotations must also be coherent with each other.

Estimation of continuous variables. Other work has considered the collective estimation of a continuous value. Early work focused on average consensus [114, 115, 116], in which the robots must agree on the average of individual initial estimates of a quantity of interest. More recently, Albani *et al.* proposed a collaborative weed mapping application that considers inter-robot networking and motion planning [117]. Robots communicate to improve the confidence interval of weed density predictions for each cell in a discrete environment. They assume that robots know the confidence value for each independent prediction and keep the prediction with the maximum confidence. In our work, in contrast, we do not assume the confidence of the prediction known, and resort to a voting process to identify the most likely annotation.

Distributed mapping and semantic classification. Notable recent work in collective perception include distributed mapping, such as DOOR-SLAM [118]. In

DOOR-SLAM, the robots construct a graph-based map (but not a semantic one) in a decentralized fashion, coping with spurious observations by identifying and rejecting outliers. In distributed semantic classification [119], effort have concentrated on cloud-based method rather than decentralized ones, such as RoboEarth [103]. In the work of Ruiz *et al.* [120], a centralized knowledge base is used to build semantic map. Classification uncertainty is solved through Conditional Random Fields, which estimate beliefs on the annotations by combining spatial and contextual knowledge.

6.3 Problem Statement

In this section, we explain our main modelling assumptions and formulate the distributed storage and collective annotation problems.

6.3.1 Assumptions

Multi-Robot System

We consider a system of N robots, each with a unique identifier $i \in \{1, \ldots, N\}$. Each robot can exchange messages within a communication range C with its neighbors $\mathcal{N}_i(t)$. Outgoing messages are limited in size at each time step t to model a finite broadcasting bandwidth. Furthermore, each robot has a limited memory capacity M_i devoted to the housekeeping of a shared data structure. This capacity is subdivided into storage capacity S_i and routing capacity R_i. The available memory of a robot at a given time is denoted by $m_i(t)$. Robots move according to a diffusion policy with obstacle avoidance [34]. We assume that each robot can localize itself in a global frame. We further assume robots to be equipped with a perception module. We abstract away the inner workings of the module and simulate its behavior through the following assumptions: *(i)* the perception module uses a sensor with

physical specifications constraining its viewing frustum (see Figure 6.3); *(ii)* the perception module is able to determine the center position of objects perfectly; *(iii)* the perception module can annotate objects probabilistically (see Section 6.3.1).

Object Representation and Detection

We consider robots to be in an environment with non-uniformly distributed objects of different types. We use 3D bounding boxes to represent these objects in space. A bounding box is fully described by its center, dimensions, and orientation [121]. We assume that the sensor of the perception module can detect and localize objects when their front face corners fall inside the sensor viewing frustum.

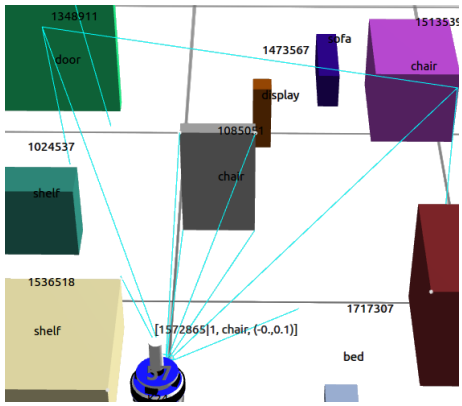


Figure 6.3: The front corners of an object are inside the sensor viewing frustum.

Simulated Classifier

Within the perception module, we simulate the statistical behavior of a trained multi-class object classifier. We leverage the work of Carillo *et al.* [122] deriving the posterior distribution of the balanced accuracy conditioned on predictions of a trained classifier on a sample test dataset. We use this posterior distribution to generate either correct or incorrect object annotations λ_ν upon each object detection. To distinguish cases of erroneous labels, we further modelled the leftover probability

distributions using the confusion matrices for the classifier. The statistical data used for these probabilistic models comes from the SceneNN benchmarking data [123]. SceneNN provides raw outcomes and accuracies for several multi-class classifiers tested extensively on a catalog of point clouds. We use the PB-T50-RS dataset with translation, rotation and scaling perturbations of the bounding boxes to compute the posterior distribution modelling the classifier statistical behavior.

6.3.2 Distributed Storage Problem Formulation

The first goal of our work is to distribute object annotation data across robots in a way that makes efficient use of the collective memory. Any data item is stored in the form of a tuple. Each robot can hold a defined number of tuples specified by its memory capacity M_i. Besides memory constraints, we want to account for communication-related costs.

Since we want to pack data items into bins (robots) with different properties, we formulate the distributed storage problem as an heterogeneous bin packing optimization problem. In particular, we consider the variant known as the Variable Cost and Size Bin Packing Problem (VCSBPP) [124].

Given a set of items $\mathcal{T}$ with different volumes v_τ to be loaded into a set of available bins $\mathcal{I}$ with different capacities M_i and costs c_i, we aim to minimize the

total cost of the bins selected to store items:

$$\min_{a,s} \quad f(a,s) = \sum_{i \in \mathcal{I}} c_i s_i \tag{6.1a}$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{I}} a_{\tau i} = 1 \qquad \forall \tau \in \mathcal{T} \tag{6.1b}$$

$$\sum_{\tau \in \mathcal{T}} v_\tau a_{\tau i} \leq M_i s_i \qquad \forall i \in \mathcal{I} \tag{6.1c}$$

$$a_{\tau i} \in \{0,1\} \qquad \forall \tau \in \mathcal{T},\ \forall i \in \mathcal{I} \tag{6.1d}$$

$$s_i \in \{0,1\} \qquad \forall i \in \mathcal{I} \tag{6.1e}$$

with $\{s_i\}_{i \in \mathcal{I}}$ denoting bin-selection variables; s_i equals 1 when a bin i is selected for storing items. $\{a_{\tau i}\}_{t \in \mathcal{T}, i \in \mathcal{I}}$ are item-to-bin assignment variables; $a_{\tau i}$ equals 1 if item τ is assigned to bin i. In our variant of the problem, we define bin costs c_i as follows:

$$c_i = \frac{1}{|\mathcal{N}_i| \cdot m_i} = \frac{1}{|\mathcal{N}_i| \cdot (M_i - \sum_{\tau \in \mathcal{T}} a_i)} \tag{6.2}$$

We select bin cost to be inversely proportional to the product of the number of neighbors and the memory left after the assignment. This is to model practical considerations such as potential for disconnections and subsequent memory overflows, which are not otherwise modelled in the VCSBPP.

6.3.3 Annotation Consolidation Problem Statement

Our second goal is to improve the map annotation accuracy by using at least V votes for each object to identify in the environment. The ensemble probability of success in predicting correct class $p_{ens}(V, p)$ should therefore be greater than the classifier per-class accuracy p for the class.

6.4 Methodology

6.4.1 Overview

Figure 6.2 shows the interconnection between the different components of the collective semantic annotation application with label consolidation. Robots move in an environment with objects to annotate. Upon detecting an object and deciding to record an observation, each robot assigns an uncertain class annotation for the location. It writes it to a shared memory as described in Section 6.4.2. Upon meeting a local condition, robots query the shared memory about a certain location to retrieve all the related observations. If they receive enough observations back from the query, they write the most frequent label for that location into the shared memory as a *consolidated annotation*. The exact voting operation and the resulting ensemble accuracy are formalized in Section 6.4.3. Over time, as robots explore the entire environment, the shared memory stores a full map of the annotated objects in the form of consolidated annotations. Section 6.4.4 describes the local routine run asynchronously by the robots to perform the various actions across components.

6.4.2 Distributed Storage Through SwarmMesh

We implement a shared memory to enable distributed storage of data by the robots. The design of the particular underlying distributed data structure, *SwarmMesh*, is detailed in Chapter 5. As part of our contribution, we provided an open-source C++ implementation of the data structure [1]. *SwarmMesh* is a distributed online heuristic solution to the Bin Packing problem formulated in Section 6.3.2, Equations (6.1)-(6.2).

SwarmMesh stores data in the form of tuples, i.e `(k,v)` pairs, in robots with

[1] `https://github.com/NESTLab/SwarmMeshLibrary`

different NodeIDs, $\delta_i(t)$. The key enables tuple addressing and consists of two parts: *(i)* a unique tuple identifier τ, *(ii)* a tuple hash ρ_τ that depends on the value to be stored. The value can be any application-specific payload. The data structure relies on the partitioning of robots according to $\delta_i(t)$ computed by them locally and instantaneously. The main design idea lies in assigning the most important tuple to the most desirable robots. This is achieved by matching values of tuple hashes with values of NodeIDs. This means that in *SwarmMesh*, *items* are assigned to *bins* according to a heuristic rule assuming a hierarchy among them.

In our application, when a robot writes data into the data structure, it creates a tuple. The unique tuple identifier τ is the concatenation of the robot identifier i and the count of tuples created so far by the robot. The value or payload of the tuple contains the predicted object class λ_ν, the box center, the box orientation and the vector from the center to the front right corner of the box. These variables fully describe an annotated 3D bounding box.

We chose a tuple hashing function that ranks the object class in increasing order of uncertainty, i.e the more uncertain the object class the higher the tuple hash:

$$h : \lambda_\nu \mapsto \rho_\tau = R(\lambda_\nu)$$

with $R(\cdot)$, a staircase function with a spacing referred to as *hashing bucket*.

Robots compute NodeIDs as follows:

$$\delta_i(t) = \begin{cases} m_i(t) \cdot |\mathcal{N}_i(t)| & \text{if } |\mathcal{N}_i(t)| > 0 \\ 1 & \text{otherwise} \end{cases}$$

The heuristic rule routes a tuple of hash ρ_τ to the first robot of NodeID $\delta_i(t)$ to

meet the condition:

$$\delta_i(t) > \rho_\tau \tag{6.3}$$

In this application, we use the following data structure operations:

- `store(k,v)` which assigns a tuple to a suitable robot in the data structure;
- `get(x,y,r)`, which returns all the tuples located within a radius r of the point (x, y) expressed in a global reference frame;
- `erase_except_tuple(x,y,r,`τ`)`, which removes from the data structure all tuples located within a radius r of the point (x, y) in the global reference frame except for the tuple with unique identifier τ.

6.4.3 Annotation Consolidation Through Plurality Voting

Plurality Vote

Upon querying the shared memory for a particular location in space, the querying robot receives class annotations predicted by various robots. If the number of votes n is greater than or equal to the minimum number of votes V, the robot aggregates these votes into a consolidated annotation $\bar{\lambda}$ for the object through plurality voting:

$$\bar{\lambda} = \underset{\text{classes}}{\operatorname{argmax}} \sum_{\nu=1}^{n} I(\lambda_\nu = \text{class})$$

In case of ties, the consolidated annotation is selected at random from the tied options.

Ensemble Probability of Success

We seek to derive the probability of success for n independent votes, each for one of c categories where $c \geq n$. We assume that class 1 is correct and that each vote

is for class 1 with probability p. The probability of an incorrect vote is then $1-p$, and we make the simplifying assumption that each incorrect class is equally likely. We denote the total vote count for each class as a vector $z = (z_1, z_2, \ldots, z_c)$ where $z_i \in \{0, 1, \ldots, n\}$ and $z_1 + z_2 + \cdots + z_c = n$. This vector follows a multinomial distribution with probability mass function

$$\phi(z \,|\, n, p) = \binom{n}{z_1, \ldots, z_c} p^{z_1} \left(\frac{1-p}{c-1}\right)^{n-z_1}$$

where

$$\binom{n}{z_1, \ldots, z_c} = \frac{n!}{z_1! \cdots z_c!}$$

denotes the multinomial coefficient.

Without knowing the correct class, we can estimate it from an observed vector as the class with the most votes, whether a majority or plurality. In the case where several classes are tied with the highest number of votes, we select one of those classes at random. Again considering that z_1 represents the true correct class, we use z^* to denote a vector of vote counts for which $z_1 \geq z_i$, for all $i = 2, 3, \ldots, c$. The probability of identifying the correct class with n total votes is then

$$p_{ens}(n, p) = \sum_{z^*} \phi(z^* \,|\, n, p) P(\text{success} \,|\, z^*) \tag{6.4}$$

since all other vectors z result in a success probability of zero. Because the largest term in z^* appears first and all incorrect classes are chosen with equal probability, we can express (6.4) using integer partitions.

Integer Partitions An integer partition ξ is a nonincreasing sequence of positive integers $\xi_1 \geq \xi_2 \geq \cdots \geq \xi_\omega$. We say that ξ is a partition of n, denoted $\xi \vdash n$, if

$\sum_{i=1}^{\omega} \xi_i = n$, and we consider the set of all such ξ to be $\mathcal{P}$. The ξ_i are called *parts* of the partition, and we define the *length* of the partition $\ell(\xi) := \omega$. An alternative formulation is to consider the infinite sequence of multiplicities of each part $\xi = (k_1, k_2, \ldots)$ where $k_j \in \{0, 1, \ldots\}$. Here, $\xi \vdash n$ if $\sum_{j=1}^{\infty} j k_j = n$, and the length $\ell(\xi) = \sum_{j=1}^{\infty} k_j = \omega$, since all but a finite number of k_j are zero [125]. We will refer to both the parts ξ_i and multiplicities k_j of the same partition throughout.

Note that we can map each vector z^* to an integer partition $g(x^*) = \xi$, where ξ_1 represents the number of votes for the correct class, and the other ξ_i represent the positive numbers of votes for $\omega - 1$ incorrect classes. Since all incorrect classes are equally likely, each z^* that maps to a particular ξ yields the same probability of success. Thus,

$$\begin{aligned} p_{ens}(n,p) &= \sum_{\substack{\xi \vdash n \\ \xi \in \mathcal{P}}} \sum_{z^* : g(z^*) = \xi} \phi(z^* \,|\, n, p) P(\text{success} \,|\, z^*) \\ &= \sum_{\substack{\xi \vdash n \\ \xi \in \mathcal{P}}} \left| g^{-1}\big(\{\xi\}\big) \right| \phi(\xi \,|\, n, p) P(\text{success} \,|\, \xi) \end{aligned} \tag{6.5}$$

where $\left| g^{-1}\big(\{\xi\}\big) \right|$ is the cardinality of the preimage of $\{\xi\}$, i.e., the number of z^* that map to ξ.

To determine $\left| g^{-1}\big(\{\xi\}\big) \right|$, we start with the $\binom{c-1}{\omega-1}$ combinations of incorrect classes and count the ways to assign vote counts with multiplicities k_j to each. There are

$$\binom{\omega - 1}{k_1, k_2, \ldots, k_{\xi_1} - 1}$$

such assignments, where the $k_{\xi_1} - 1$ derives from the fact that one of the spots for

the largest possible vote count is taken by the correct class. Together, we have

$$\left|g^{-1}(\{\xi\})\right| = \binom{c-1}{\omega-1}\binom{\omega-1}{k_1, k_2, \ldots, k_{\xi_1}-1}$$

For the conditional probability $P(\text{success} \mid \xi)$, when ξ_1 is the strictly largest part, it is chosen with probability 1; otherwise, there is a k_{ξ_1}-way tie, from which a class is chosen at random. In both cases, the correct class is chosen with probability $P(\text{success} \mid \xi) = 1/k_{\xi_1}$. Substituting into (6.5) and simplifying gives

$$p_{ens}(n,p) = \frac{1}{c}\sum_{\substack{\xi \vdash n \\ \xi \in \mathcal{P}}} \binom{n}{\xi_1, \ldots, \xi_\omega}\binom{c}{\omega}\binom{\omega}{k_1, \ldots, k_{\xi_1}} p^{\xi_1}\left(\frac{1-p}{c-1}\right)^{n-\xi_1} \tag{6.6}$$

6.4.4 Robot Local Routine

Robots each run a local procedure in a loop. We refer to this procedure as the *control step* and outline it from the perspective of robot i in Algorithm 4. Every time step, the robot performs one iteration of this loop.

At the beginning of the procedure, the robot receives and deserializes messages from its neighbors. Certain messages are meant for specific recipients only (point-to-point) and get discarded by any other receiving robot during deserialization. The robot then computes and performs a motion increment according to a diffusion policy [34].

Next, the robot checks the output of its perception module if it is not in recording timeout. The recording timeout is a delay imposed to avoid making successive observations of the same object. If the perception module has detected an object, the robot creates a tuple and starts a `store()` operation on the shared memory. The

payload of the tuple contains the object annotation and the bounding box location and dimensions.

If the robot is not in a querying timeout, it reviews the tuples stored in its memory by bounding box location. If the robot finds multiple tuples with the same location in its own local memory, it starts a `get()` query on SwarmMesh to retrieve all the tuples currently stored with that bounding box location. It then saves the meta data for the started query.

The robot goes through the queries it has started and checks whether it has not received replies for a period longer than a time threshold. If the number of tuples is larger than the minimum number of votes, the robot creates and writes a tuple into SwarmMesh with the consolidated annotation. It also start an `erase_except_tuple()` request of all tuples with the given location except for the newly created tuple.

SwarmMesh routing at line 23 of Alg. 4 decides which requests and replies need to be routed and to which robot. A robot may have to send a reply to a request and continue propagating the request or route a neighbor's reply or discard the request or reply. The details of the routing process are reported in [30].

6.5 Evaluation

We performed the evaluation of our distributed map storage and fusion approach using the physics-based ARGoS multi-robot simulator [99] with data from the SceneNN benchmark [123]. The main parameters of interest are the minimum number of votes V and the number of robots N. The former relates to the ensemble accuracy of the map through Equation 6.6 and Figure 6.5. The latter changes the scale of the distributed system and the degree of parallelization in collecting data.

Algorithm 4 Control step - message μ, neighbors $\mathcal{N}_i$, requests $\mathcal{Q}$, replies $\mathcal{R}$

procedure CONTROL_STEP
  receive $\{\mu_j\}_{j\in\mathcal{N}_i}$ ▷ RX from neighbors
  for each $j \in \mathcal{N}_i$ **do**
    $(j, \delta_j, \mathcal{Q}_j, \mathcal{R}_j) \leftarrow \mu_j$ ▷ Deserialization
  move() ▷ Diffusion motion
  if not in recording timeout **then**
    `v` $\leftarrow$ **senseObject**()
    if object detected **then**
      start recording timeout
      `k` $\leftarrow$ **hash**(`v`)
      $\mathcal{Q}_i \leftarrow \mathcal{Q}_i \cup$ `store(k, v)` ▷ Write label
  if not in querying timeout **then**
    if holds multiple tuples from same `(x,y)` **then**
      start querying timeout
      $\mathcal{Q}_i \leftarrow \mathcal{Q}_i \cup$ `get(x, y, 0)` ▷ Ask for labels
      save query information
  for each started query **do**
    if results ready and enough votes **then**
      `v` $\leftarrow$ plurality vote ▷ Consolidated label
      `k` $\leftarrow$ **hash**(`v`)
      $\mathcal{Q}_i \leftarrow \mathcal{Q}_i \cup$ `store(k,v)`
      $\mathcal{Q}_i \leftarrow \mathcal{Q}_i \cup$ `erase_expt_tuple(x,y,0,k)`
  $(\bar{\mathcal{Q}}_i, \bar{\mathcal{R}}_i) \leftarrow$ **route**$(\mathcal{Q}_\bullet, \mathcal{R}_\bullet)$ ▷ SwarmMesh routing
  $\mu_i \leftarrow (i, \delta_i, \bar{\mathcal{Q}}_i, \bar{\mathcal{R}}_i)$ ▷ Serialization
  send (μ_i) ▷ TX to neighbors

6.5.1 Simulation Parameters

Robots and environment We study the influence of the number of robots N by running experiments with 30, 60 and 90 robots. Robots have a communication range C of 2 m and are moving at a forward speed of 5 cm/s. Robots are diffusing and avoiding obstacles in an 8x8 m^2 environment. Given these settings, the robot density is such that robots are within communication range most of the time but a significant number of intermittent disconnections occur. We do not consider line-of-sight obstructions.

In order to run experiments representative of collective annotation in an indoors environment, we import 40 objects of 13 types from the SceneNN dataset. Therefore, the objects are non-uniformly distributed in space. We made the following

adjustments when importing scene 005 of the dataset: *(i)* we limited physical dimensions to fit in the viewing frustum; *(ii)* we lowered objects to the ground level; *(iii)* we rotated them to face towards the center of each "room"; *(iv)* we removed overlapping objects; *(v)* we re-labelled "unknown" objects by drawing from the list of object classes at random. Figure 6.4 shows the imported environment with these modifications.

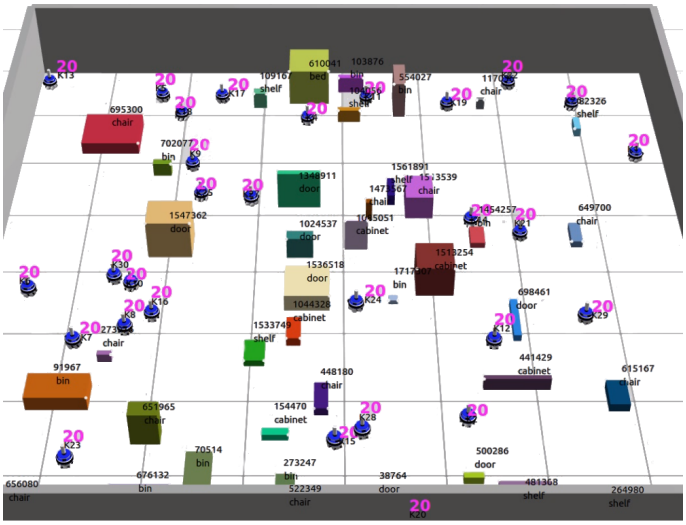


Figure 6.4: Environment adapted from the SceneNN dataset.

Simulated Classifier We simulate the statistical behavior of the BGA-DGCNN classifier using data from the SceneNN dataset [110, 123]. The procedure to generate a posterior distribution is described in Section 6.3.1. For the BGA-DGCNN, the distribution is a Gaussian distribution. Each observed annotation λ_ν in the simulation is given by drawing from this distribution.

Ensemble Accuracy In order to set V according to a desired range of probabilistic accuracy, we compute the ensemble accuracy $p_{ens}(n, p)$ for each class of objects for a given number of votes n using Equation 6.6. We use the per-class accuracy p for BGA-DGCNN shown in Table 6.1, as reported in [110]. Figure 6.5 is the resulting curve. In our experiments, we vary the threshold V in the range

between the two vertical lines in Figure 6.5.

Table 6.1: BGA-DGCNN classifier class accuracy on SceneNN PB-T50-RS dataset [110].

bin	**cabinet**	**chair**	**desk**	**display**	**door**	**shelf**
81.9	84.4	92.6	77.3	80.4	92.4	80.5
table	**bed**	**pillow**	**sink**	**sofa**	**toilet**	**overall**
74.1	72.7	78.1	79.2	91	79.7	75.7

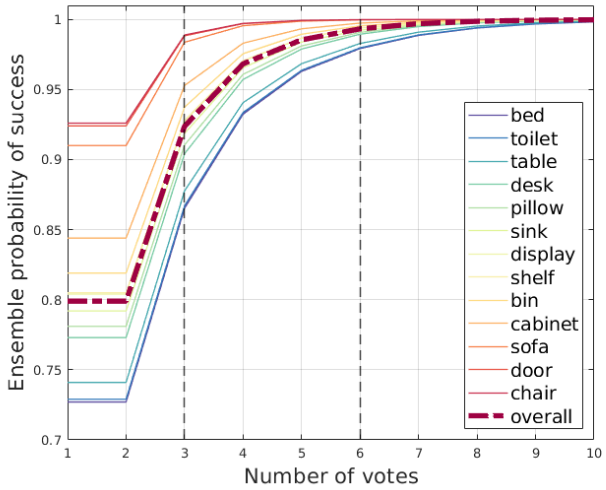


Figure 6.5: Ensemble probability of success per class of object. This graph is based on the per-class accuracies in Table 6.1.

SwarmMesh parameters We select the data structure parameters so as to maintain a reasonable load factor i.e the ratio of data to store over the available storage capacity. We set the memory capacity of robots M_i to 20 tuples with a a target of 10 tuple for storage and an allowance of 10 for the routing queue. We set the step in the hashing function to 5.

6.5.2 Mapping Performance

Effect of the number of robots. Using more robots distributes sensing spatially and leads to a more densely connected robotic network. As a result, we expect that

Table 6.2: Simulation parameters.

Parameter	**Value**
Number of Robots N	$\{30, 60, 90\}$ robots
Communication range C	2 m
Robot density	$\{0.5, 0.9, 1.4\}$ robot/m^2
Robot speed	5 cm/s
Number of objects	40
Number of classes	13
Memory capacity M_i	20 tuples $\forall\ i$
Storage capacity S_i	10 tuples $\forall\ i$
Routing capacity R_i	10 tuples $\forall\ i$
Hashing bucket	5

increasing N speeds up the collection and aggregation of data needed to complete the semantic map. In our setting, map coverage refers to the ratio of *covered* objects to the total number of objects. We make a distinction between two types of coverage: *(i)* the *observation coverage* which considers an object covered if at least one robot annotated the object; *(ii)* the *consolidation coverage* which considers an object covered if there exists a consolidated annotation $\bar{\lambda}$ for the object in the shared memory. Figure 6.6 shows the temporal curve of both types of coverage across different values of N and a set threshold of votes $V = 3$. The curves of map coverage over time increase faster with a higher number of robots. The consolidation coverage rises slower than the observation coverage since it depends on the consolidation process that queries the shared memory as described in Section 6.4.4. With $N = 90$, the delay for consolidating all the annotations is merely 14 s from the time all objects have been observed.

Effect of the number of votes. The observation coverage is independent of V. Figure 6.7 shows the map accuracy and the consolidation coverage over time across values of V. The experimental map accuracy is the ratio of correct consolidated annotations to the number of consolidated annotations. We observe that the map accuracy increases with V at the cost of a slower coverage.

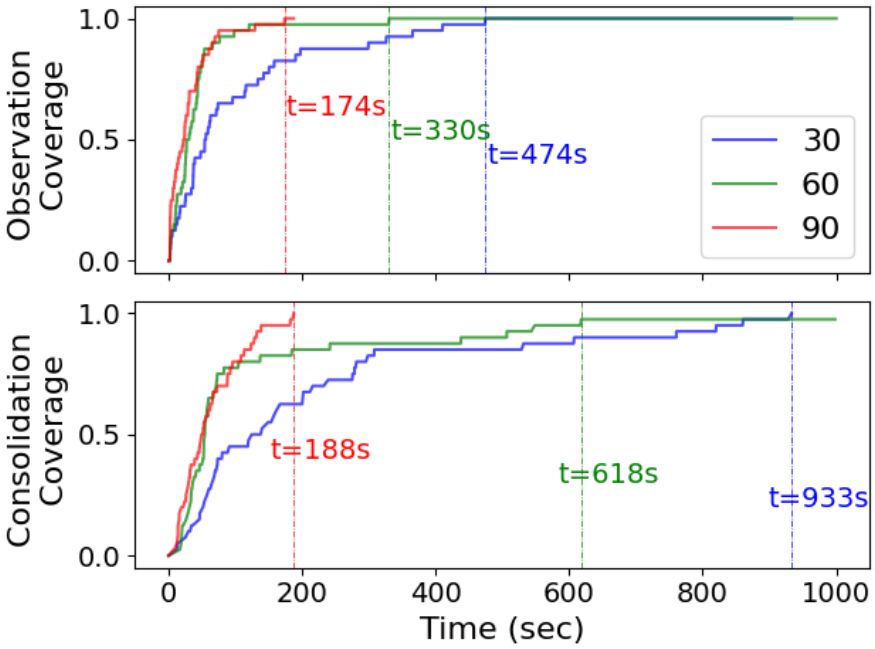


Figure 6.6: Map coverage over time for $V = 3$ and $N \in \{30, 60, 90\}$. Cursors show the time when the maximum coverage was first reached.

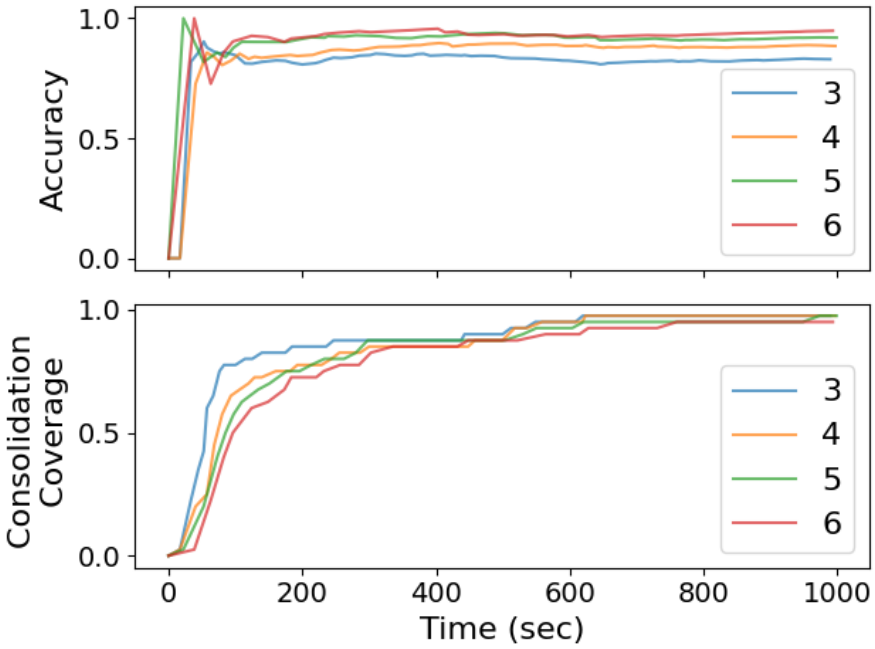


Figure 6.7: Map accuracy and coverage over time for $N = 60$ and $V \in \{3, 4, 5, 6\}$.

6.5.3 Memory-Related Performance

Distributed storage cost. Figure 6.8 shows the total storage cost (6.1a) of the tuples-to-robots assignment realized in practice over time in the simulation. At every time step, the tuple set $\mathcal{T}$ and the number of neighbors of each robot $|\mathcal{N}_i|$ are updated. In our evaluation, we consider tuples of unit volume ($v_\tau = 1, \forall \tau$) and an homogeneous memory capacity across robots ($M_i = M, \forall i$). We calculate the cost obtained in simulation using the heuristic tuple-to-robot assignment rule of *SwarmMesh*. In order to tell apart cases in which tuples are assigned to robots with null $|\mathcal{N}_i|$ and/or m_i, we cap each term in the sum to 1. These cases result in jumps of 1 of the cost in Figure 6.8.

The VSCBPP problem is NP-hard but our experimental settings ($v_\tau = 1, \forall \tau$) and ($M_i = M, \forall i$) allow for optimally solving instances of the simulation with $N = 30$, $V = 3$. We reduce the space of the exhaustive search by noting that permutations of the rows of $(a_{\tau i})$ lead to equivalent solutions in terms of cost. We enumerate integer partitions of $|\mathcal{T}|$ with fewer than $|\mathcal{I}|$ parts and such that all parts are smaller than M. For each such partition, we determine the minimal cost using the rearrangement inequality for the product of $1/|\mathcal{N}_i|$ and $1/m_i$:

$$\sum_{i \in \mathcal{I}} \frac{1}{|\mathcal{N}_i|} \cdot \frac{1}{m_i} \geq \frac{1}{|\mathcal{N}_{\sigma(1)}|} \cdot \frac{1}{|m_{\pi(1)}|} + \cdots + \frac{1}{|\mathcal{N}_{\sigma(|\mathcal{I}|)}|} \cdot \frac{1}{|m_{\pi(|\mathcal{I}|)}|}$$

where $\sigma(.)$ is an ascending ordering of $|\mathcal{N}_i|$, and $\pi(.)$ is a descending ordering of m_i.

Given $|\mathcal{N}_i|$ and $\mathcal{T}$, we compute the worst cost by assigning one tuple to each robot with $|\mathcal{N}_i| = 0$, filling the memory ($m_i = 0$) of as many robots as possible given the number of tuples $|\mathcal{T}|$ and placing the remainder of tuples in the robot of lowest non-zero $|\mathcal{N}_i|$.

SwarmMesh dimensioning. Figure 6.9 validates our selection of the *Swar-*

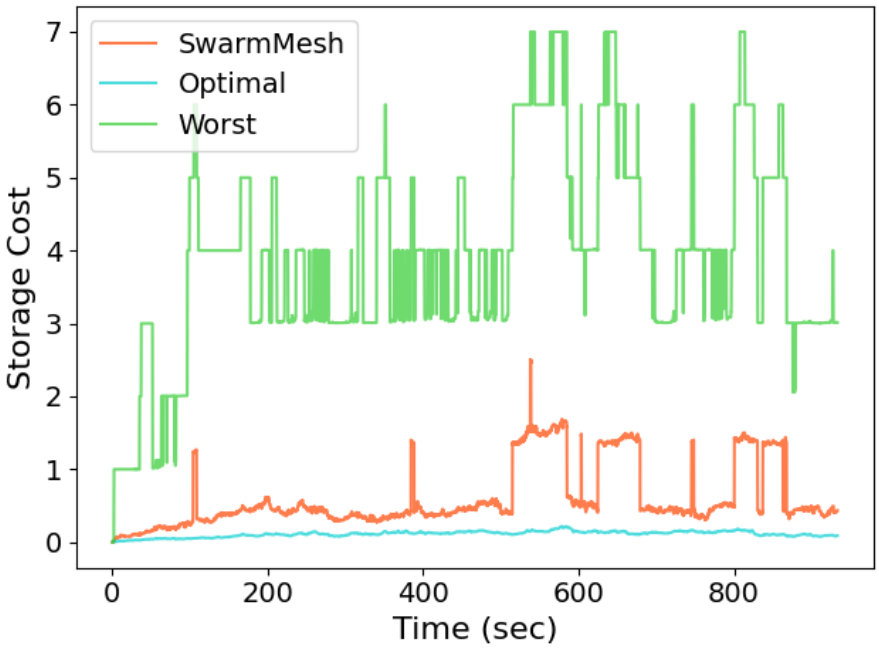


Figure 6.8: VSCBPP cost over simulation time for $V = 3$ and $N = 30$.

mMesh parameters. It shows that the bias of the NodeID distribution is greater than that of the distribution of data hashes. This indicates that the key partitioning of the distributed data structure is appropriate given the heuristic assignment rule (Equation 6.3), i.e it is likely to find a robot of greater NodeID than a given tuple hash. We keep the parameters constant across numbers of robots. This means that with higher N, the collective memory capacity $N \cdot M$ is over-sized and the key partitioning matches tuples to robots randomly as they almost all meet the condition in Equation 6.3. In practice, this means that the robot writing the tuple typically keeps it locally which leads to reduced communication overhead at greater N values.

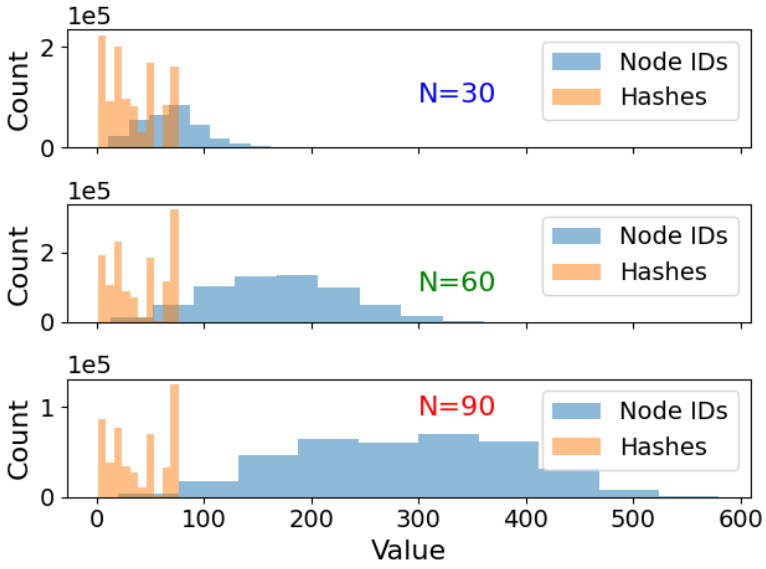


Figure 6.9: Histograms of data hashes and NodeIDs for $V = 6$ and $N \in \{30, 60, 90\}$.

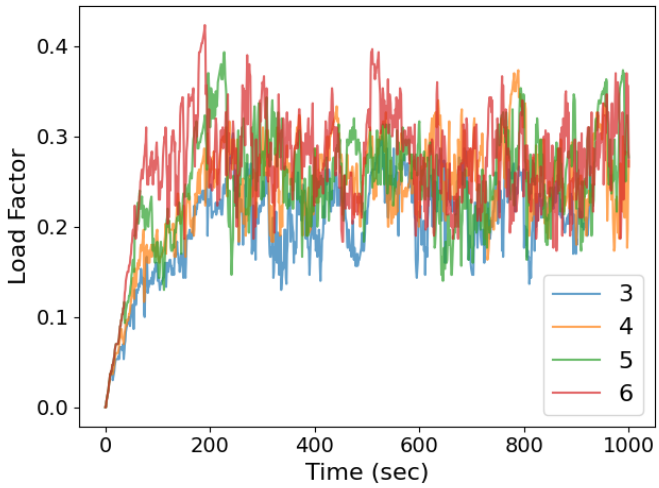


Figure 6.10: Load factor over time for $N = 60$.

6.5.4 Communication Load

Effect of the number of votes. We find that the number of bytes sent increases with the minimum number of votes V (Figure 6.11). This indicates that an increase in map accuracy comes at the cost of a higher communication overhead.

Effect of the number of robots. Figure 6.12 shows the bandwidth usage per

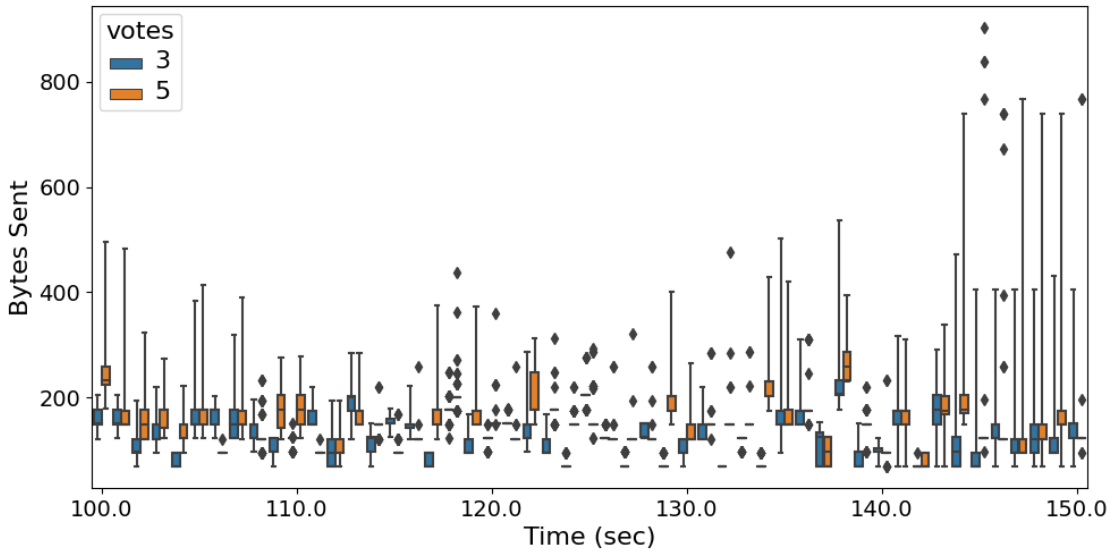


Figure 6.11: Bytes sent per second per robot over time for $N = 60$ and $V \in \{3, 5\}$.

robot across different values of N in the configuration with the largest amount of data to be managed ($V = 6$). Communication overhead decreases when N increases because we keep the same shared memory parameters across values of N. This is consistent with the effect described in the *SwarmMesh* dimensioning section.

6.6 Summary

We proposed a method for the distributed storage and fusion of semantic annotations across a swarm of robots. We consider robots with limited memory, each running a pre-trained classifier to annotate objects in the environment. We show the costs of storing and reducing the uncertainty of a semantic map in terms of mapping performance, memory usage and communication overhead for users of this framework to make informed decisions on the trade-offs between the key performance metrics.

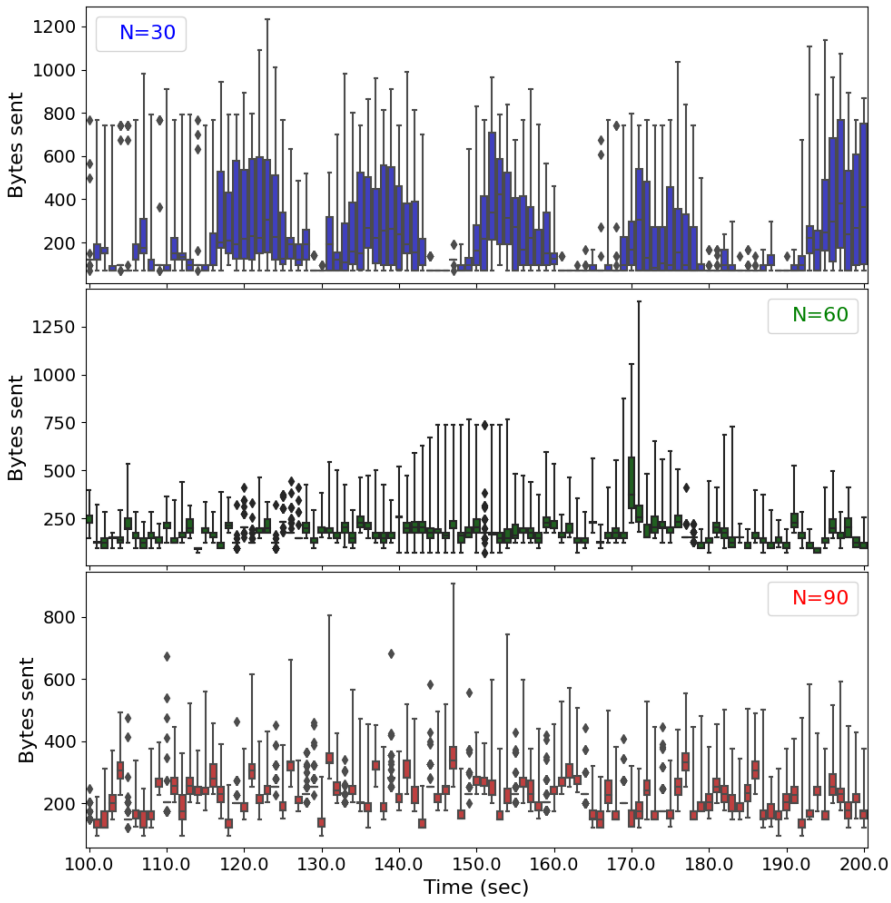


Figure 6.12: Bytes sent per second per robot over time for $V = 6$ and $N \in \{30, 60, 90\}$.

6.7 Open Problems

In future work, we will study the aggregation of outputs of different types of classifiers to produce richer semantic maps. We will also assess the performance of our approach on real robots.

Chapter 7

Conclusions

7.1 Summary

In the first part of this thesis, we dealt with creating and maintaining communication paths for information to be exchanged between robots.

In Chapter 3, we considered how to preserve a connected communication graph when certain robots have to reach distant targets. We contributed two decentralized algorithms to form and preserve a communication backbone by building a logical tree over network links between robots. We performed an extensive evaluation of our approach in simulation and with real robots.

Outcomes of this work are a scalable strategy tested with more than 100 robots in simulation and insights about the performance costs of enforcing connectivity.

The second part of the thesis is about organizing the ownership and flow of data throughout the network formed by robots.

In Chapter 4, we designed a shared data structure, SwarmMesh, that facilitates the collection and management of data in a distributed fashion in multi-robot systems. Our work targets low-memory, low-bandwidth, highly mobile teams of robots.

The main design idea in SwarmMesh is to rank data items and robots according to their defining features and subsequently, distribute data items to robots according to this hierarchy. We proposed two methods of distributing storage responsibility. We stress-tested the data structure to verify that we were able to retain large amounts of data appearing very quickly. Besides overall good routing performance, we recorded near-zero occurrences of robots having to discard data due to storage memory overflows.

The result of this work is the modular design of a distributed data structure suitable for various swarm applications. Our work can be reused and adapted easily as the logic that defines the data distribution can be changed by the user.

In the third part of this thesis, we demonstrate the usefulness of distributed data structures specifically suited to robot swarms. We tackle two multi-robot applications through novel communication paradigms.

In Chapter 5, we studied the design space of the Federated Learning framework in a robotics setting. This framework enables the training of a single ML model by multiple learners by merging their local models rather than aggregating their local datasets. We provided a practical realization of fully distributed Federated Learning through the use of a shared data structure. Our contribution includes a novel way of scheduling model updates based on the data flow. This was achieved through the use of two parameters: the *quota*, i.e., the minimum number of data samples for a robot to qualify for a model update, and the *quorum*, i.e., the minimum number of robots to start a learning round.

This work advances the field of multi-robot learning by showcasing and adapting a framework that has not been used in a similar setting before.

In Chapter 6, we developed an approach to the collective storage and fusion of semantic annotations by a team of memory-constrained robots. We assumed that

robots run a pre-trained classifier to annotate objects in the environment. Aside from adapting SwarmMesh to store these annotations, we also realized a voting mechanism to reduce the uncertainty of classifier outputs. We evaluated the data storage and fusion costs of this application. In particular, we considered the mapping performance, memory usage, and communication overhead.

The outcome of this work is a framework for data fusion in an ML-perception setting. Our evaluation intends to enable users to make informed trade-offs between the key performance metrics of the application.

7.2 Future Work

The goal of this thesis is to provide tools and algorithms for a team of robots to communicate in order to achieve cooperation. Our contributions abstract away some of the implementation details. Ultimately, to deploy the proposed applications, some technological choices will have to be made in terms of hardware and the lower layers of the communication protocol stack. Propagation effects, communication interference, and medium sharing will have to be accounted for.

We envision our data sharing approach to be useful for other applications that require merging world representations while coping with limited memory. A few such examples include multi-robot mapping, task allocation, and multi-view learning.

Finally, another direction for future work is securing inter-robot communications given a hardware implementation and a specific application. At the center of this work is the definition of a realistic threat model. There exists many security protocols for networked systems that ensure the confidentiality, authenticity, integrity of communications. However, multi-robot systems pose different challenges due to their high mobility. Therefore, an open problem in designing cryptographic

protocols is how to handle dynamic key management in such systems. Ensuring data privacy between robots is also a potential direction of study for settings where robots compete with each other. The Federated Learning framework is particularly well-suited in the data privacy aspect.

Bibliography

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

[2] Z. Yan, N. Jouandeau, and A. A. Cherif, "A survey and analysis of multi-robot coordination," *International Journal of Advanced Robotic Systems*, vol. 10, no. 12, p. 399, 2013.

[3] E. Tuci, M. H. Alkilabi, and O. Akanyeti, "Cooperative object transport in multi-robot systems: A review of the state-of-the-art," *Frontiers in Robotics and AI*, vol. 5, p. 59, 2018.

[4] Y. Liu and G. Nejat, "Robotic urban search and rescue: A survey from the control perspective," *Journal of Intelligent & Robotic Systems*, vol. 72, no. 2, pp. 147–165, 2013.

[5] W. Burgard, M. Moors, C. Stachniss, and F. E. Schneider, "Coordinated multi-robot exploration," *IEEE Transactions on robotics*, vol. 21, no. 3, pp. 376–386, 2005.

[6] D. Fox, W. Burgard, H. Kruppa, and S. Thrun, "A probabilistic approach to collaborative multi-robot localization," *Autonomous robots*, vol. 8, no. 3, pp. 325–344, 2000.

[7] D. Goldsmith, *Voyage to the Milky Way: The Future of Space Exploration*. TV Books, NY, 1999.

[8] R. F. Rubio, "Mining: The challenge knocks on our door," *Mine Water and the Environment*, vol. 31, no. 1, pp. 69–73, 2012.

[9] H. M. A. Fahmy, *Protocol Stack of WSNs*. Cham: Springer International Publishing, 2021, pp. 53–66.

[10] P. Ghosh, A. Gasparri, J. Jin, and B. Krishnamachari, *Robotic Wireless Sensor Networks*. Cham: Springer International Publishing, 2019, pp. 545–595.

[11] D. B. Johnson and D. A. Maltz, "Dynamic source routing in ad hoc wireless networks," in *Mobile computing*. Springer, 1996, pp. 153–181.

[12] W. Ren and R. W. Beard, "Consensus seeking in multiagent systems under dynamically changing interaction topologies," *IEEE Transactions on automatic control*, vol. 50, no. 5, pp. 655–661, 2005.

[13] A. Cornejo, "Local Distributed Algorithms for Multi-Robot Systems," *PhD Thesis*, 2012.

[14] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128610001568

[15] C. Gomez, J. Oller, and J. Paradells, "Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology," *Sensors*, vol. 12, no. 9, pp. 11 734–11 753, 2012.

[16] S. Safaric and K. Malaric, "Zigbee wireless standard," in *Proceedings ELMAR 2006*. IEEE, 2006, pp. 259–262.

[17] K. Mekki, E. Bajic, F. Chaxel, and F. Meyer, "Overview of cellular LPWAN technologies for IoT deployment: Sigfox, LoRaWAN, and NB-IoT," in *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 2018, pp. 197–202.

[18] "Sigfox world coverage," http://www.sigfox.com/en/coverage.

[19] S.-M. Oh and J. Shin, "An efficient small data transmission scheme in the 3GPP NB-IoT system," *IEEE Communications Letters*, vol. 21, no. 3, pp. 660–663, 2016.

[20] "LoRa world coverage," http://www.lora-alliance.org.

[21] E. M. Belding-Royer, S. Basagni, M. Conti, S. Giordano, and I. Stojmenovic, "Routing approaches in mobile ad hoc networks," *Mobile ad hoc networking*, vol. 1, no. 1, pp. 275–300, 2004.

[22] Y. Fang and W. Ma, "Mobility management for wireless networks: modeling and analysis," in *Wireless communications systems and networks*. Springer, 2004, pp. 473–512.

[23] O. S. Oubbati, M. Atiquzzaman, P. Lorenz, M. H. Tareque, and M. S. Hossain, "Routing in flying ad hoc networks: Survey, constraints, and future challenge perspectives," *IEEE Access*, vol. 7, pp. 81 057–81 105, 2019.

[24] V. D. Viswacheda, A. Chekima, F. Wong, and J. A. Dargham, "A study on vehicular ad hoc networks," in *2015 3rd International Conference on Artificial Intelligence, Modelling and Simulation (AIMS)*, Dec 2015, pp. 422–426.

[25] S. Yousefi, M. S. Mousavi, and M. Fathy, "Vehicular ad hoc networks (vanets):

Challenges and perspectives," in *2006 6th International Conference on ITS Telecommunications*, June 2006, pp. 761–766.

[26] J. Sánchez-García, J. García-Campos, M. Arzamendia, D. G. Reina, S. Toral, and D. Gregor, "A survey on unmanned aerial and aquatic vehicle multi-hop networks: Wireless communications, evaluation tools and applications," *Computer Communications*, vol. 119, pp. 43–65, 2018.

[27] D. Tarapore, R. Gross, and K.-P. Zauner, "Sparse robot swarms: Moving swarms to real world applications," *Frontiers in Robotics and AI*, 2020.

[28] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, "Advances and open problems in federated learning," *arXiv preprint arXiv:1912.04977*, 2019.

[29] N. Majcherczyk, A. Jayabalan, G. Beltrame, and C. Pinciroli, "Decentralized connectivity-preserving deployment of large-scale robot swarms," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 4295–4302.

[30] N. Majcherczyk and C. Pinciroli, "SwarmMesh: A distributed data structure for cooperative multi-robot applications," *IEEE International Conference on Robotics and Automation*, 2020.

[31] N. Majcherczyk, N. Srishankar, and C. Pinciroli, "Flow-fl: Data-driven federated learning for spatio-temporal predictions in multi-robot systems," *arXiv preprint arXiv:2010.08595*, 2020.

[32] N. Majcherczyk, D. J. Nallathambi, T. Antonelli, and C. Pinciroli, "Distributed data storage and fusion for collective perception in resource-limited mobile robot swarms," *arXiv preprint arXiv:2012.08061*, 2020.

[33] J. Dede, A. Förster, E. Hernández-Orallo, J. Herrera-Tapia, K. Kuladinithi, V. Kuppusamy, P. Manzoni, A. bin Muslim, A. Udugama, and Z. Vatandas, "Simulating opportunistic networks: Survey and future directions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 2, pp. 1547–1573, 2017.

[34] A. Howard, M. Mataric, and G. Sukhatme, "Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem," in *DARS*, 2002.

[35] W. M. Spears, R. Heil, and D. Zarzhitsky, "Artificial physics for mobile robot formations," in *2005 IEEE International Conference on Systems, Man and Cybernetics*, vol. 3, 2005, pp. 2287–2292 Vol. 3.

[36] W. Liu, A. F. T. Winfield, J. Sa, J. Chen, and L. Dou, "Towards energy optimization: Emergent task allocation in a swarm of foraging robots," *Adaptive Behavior*, vol. 15, no. 3, pp. 289–305, 2007. [Online]. Available: https://doi.org/10.1177/1059712307082088

[37] M. Nakagami, "The m-distribution—a general formula of intensity distribution of rapid fading," in *Statistical methods in radio wave propagation*. Elsevier, 1960, pp. 3–36.

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

[39] R. Aragues, C. Sagues, and Y. Mezouar, "Triggered minimum spanning tree for distributed coverage with connectivity maintenance," in *2014 European Control Conference (ECC)*, 2014, pp. 1881–1887.

[40] T. Nestmeyer, P. R. Giordano, H. H. Bülthoff, and A. Franchi, "Decentralized simultaneous multi-target exploration using a connected network of multiple robots," *Autonomous Robots*, vol. 41, no. 4, pp. 989–1011, 2017.

[41] P. Yang, R. A. Freeman, G. J. Gordon, K. M. Lynch, S. S. Srinivasa, and R. Sukthankar, "Decentralized estimation and control of graph connectivity for mobile sensor networks," *Automatica*, vol. 46, no. 2, pp. 390–396, 2010.

[42] L. Sabattini, N. Chopra, and C. Secchi, "On decentralized connectivity maintenance for mobile robotic systems," *Proceedings of the IEEE Conference on Decision and Control*, pp. 988–993, 2011.

[43] "Locally constrained connectivity control in mobile robot networks," in *2013 IEEE International Conference on Robotics and Automation.* IEEE, may 2013, pp. 901–906. [Online]. Available: http://ieeexplore.ieee.org/document/6630680/

[44] P. Robuffo Giordano, A. Franchi, C. Secchi, and B. HH, "A passivity-based decentralized strategy for generalized connectivity maintenance," *The International Journal of Robotics Research*, vol. 32, no. 3, pp. 299–323, 2013.

[45] P. Di Lorenzo and S. Barbarossa, "Distributed Estimation and Control of Algebraic Connectivity over Random Graphs," pp. 1–13, sep 2013. [Online]. Available: http://arxiv.org/abs/1309.3200http://dx.doi.org/10.1109/TSP.2014.2355778

[46] A. Bertrand and M. Moonen, "Distributed computation of the Fiedler vector with application to topology inference in ad hoc networks," in *Signal Processing*, vol. 93, no. 5, 2013, pp. 1106–1117.

[47] T. Sahai, A. Speranzon, and A. Banaszuk, "Hearing the clusters of a graph: A distributed algorithm," *Automatica*, vol. 48, no. 1, pp. 15–24, 2012.

[48] M. A. Hsieh, A. Cowley, V. Kumar, and C. J. Taylor, "Maintaining network connectivity and performance in robot teams," *Journal of Field Robotics*, vol. 25, no. 1-2, pp. 111–131, 2008.

[49] N. Michael, M. M. Zavlanos, V. Kumar, and G. J. Pappas, "Maintaining Connectivity in Mobile Robot Networks," *Springer Tracts in Advanced Robotics*, vol. 54, pp. 117–126, 2009.

[50] A. Cornejo, F. Kuhn, R. Ley-Wild, and N. Lynch, "Keeping mobile robot swarms connected," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5805 LNCS, pp. 496–511, 2009.

[51] D. Krupke, M. Ernestus, M. Hemmer, and S. P. Fekete, "Distributed cohesive control for robot swarms: Maintaining good connectivity in the presence of exterior forces," in *IEEE International Conference on Intelligent Robots and Systems*, vol. 2015-Decem, 2015, pp. 413–420.

[52] T. Soleymani, E. Garone, and M. Dorigo, "Distributed Predictive Connectivity Control for Double Integrator Agents based on a Receding Horizon Scheme," in *American Control Conference, ACC 2015*, 2015, pp. 1369–1374.

[53] M. Schuresko and J. Cortés, "Distributed tree rearrangements for reachability and robust connectivity," *SIAM Journal of Control Optimization*, vol. 50, no. 5, pp. 2588—-2620, 2012.

[54] M. Fiedler, "Algebraic connectivity of graphs," *Czechoslovak Mathematical Journal*, vol. 23, no. 98, pp. 298—-305, 1973.

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

[56] M. Kudelski, L. M. Gambardella, and G. A. Di Caro, "RoboNetSim: An integrated framework for multi-robot and network simulation," *Robotics and Autonomous Systems*, vol. 61, no. 5, pp. 483–496, 2013.

[57] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Transactions on automation science and engineering*, vol. 12, no. 2, pp. 398–409, 2015.

[58] G. Urdaneta, G. Pierre, and M. V. Steen, "A survey of DHT security techniques," *ACM Computing Surveys*, vol. 43, no. 2, pp. 1–49, 2011. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1883612.1883615

[59] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Communications Surveys & Tutorials*, vol. 7, no. 2, pp. 72–93, 2005.

[60] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, *A scalable content-addressable network*. ACM, 2001, vol. 31, no. 4.

[61] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 149–160, 2001.

[62] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *IFIP/ACM International*

Conference on Distributed Systems Platforms and Open Distributed Processing. Springer, 2001, pp. 329–350.

[63] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," *IEEE Journal on selected areas in communications*, vol. 22, no. 1, pp. 41–53, 2004.

[64] B. Cohen, "Incentives build robustness in bittorrent," in *Workshop on Economics of Peer-to-Peer systems*, vol. 6, 2003, pp. 68–72.

[65] J. P. Ahulló, P. G. López, M. S. Artigas, and A. F. Gómez Skarmeta, "Supporting geographical queries onto DHTs," *Proceedings - Conference on Local Computer Networks, LCN*, pp. 435–442, 2008.

[66] W. Wu, Y. Chen, X. Zhang, X. Shi, L. Cong, B. Deng, and X. Li, "Ldht: Locality-aware distributed hash tables," in *2008 International Conference on Information Networking*. IEEE, 2008, pp. 1–5.

[67] A. Pethalakshmi and C. Jeyabharathi, "Geo-chord: Geographical location based chord protocol in grid computing," *International Journal of Computer Applications*, vol. 94, no. 3, 2014.

[68] S. Matsuura, K. Fujikawa, and H. Sunahara, "Mill: A geographical location oriented overlay network managing data of ubiquitous sensors," *IEICE transactions on communications*, vol. 90, no. 10, pp. 2720–2728, 2007.

[69] F. Araujo, L. Rodrigues, J. Kaiser, C. Liu, and C. Mitidieri, "Chr: a distributed hash table for wireless ad hoc networks," in *25th IEEE international conference on distributed computing systems workshops*. IEEE, 2005, pp. 407–413.

[70] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron, "Exploiting network proximity in distributed hash tables," in *International Workshop on Future Directions in Distributed Computing (FuDiCo)*, 2002, pp. 52–55.

[71] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, Y. Li, and Y. U. Fang, "Data-Centric Storage in Sensornets with GHT, a Geographic Hash Table," Tech. Rep., 2003. [Online]. Available: https://link.springer.com/content/pdf/10.1023/A:1024591915518.pdf

[72] S. Zeadally, R. Hunt, Y.-S. Chen, A. Irwin, and A. Hassan, "Vehicular ad hoc networks (vanets): status, results, and challenges," *Telecommunication Systems*, vol. 50, no. 4, pp. 217–241, 2012.

[73] R. Ravichandran, E. Prassler, N. Huebel, and S. Blumenthal, "A workbench for quantitative comparison of databases in multi-robot applications," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 3744–3750.

[74] D. Fourie, S. Claassens, S. Pillai, R. Mata, and J. Leonard, "Slamindb: Centralized graph databases for mobile robotics," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 6331–6337.

[75] A. J. Fiannaca and J. Huang, "Benchmarking of relational and nosql databases to determine constraints for querying robot execution logs," *Computer Science & Engineering, University of Washington, USA*, pp. 1–8, 2015.

[76] D. Sun, A. Kleiner, and C. Schindelhauer, "Decentralized Hash Tables For Mobile Robot Teams Solving Intra-Logistics Tasks," in *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems*. IFAAMAS, 2010, p. 10.

[77] C. Pinciroli, A. Lee-Brown, and G. Beltrame, "A tuple space for data sharing in robot swarms," in *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*. ICST (Institute for Computer Sciences, Social-Informatics and . . . , 2016, pp. 287–294.

[78] V. S. Varadharajan, D. St-Onge, B. Adams, and G. Beltrame, "Soul: data sharing for robot swarms," *Autonomous Robots*, pp. 1–18, 2019.

[79] P. S. Almeida, C. Baquero, and V. Fonte, "Interval tree clocks," in *International Conference On Principles Of Distributed Systems*. Springer, 2008, pp. 259–274.

[80] K. Yao, R. E. Hudson, C. W. Reed, D. Chen, and F. Lorenzelli, "Blind beamforming on a randomly distributed sensor array system," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 8, pp. 1555–1567, 1998.

[81] G. Frazier, Q. Duong, M. P. Wellman, and E. Petersen, "Incentivizing responsible networking via introduction-based routing," in *International Conference on Trust and Trustworthy Computing*. Springer, 2011, pp. 277–293.

[82] B. McMahan, E. Moore, D. Ramage, and S. Hampson, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*.

[83] M. Everett, Y. F. Chen, and J. P. How, "Collision avoidance in pedestrian-rich environments with deep reinforcement learning," 2020.

[84] D. Peteiro-Barral and B. Guijarro-Berdiñas, "A survey of methods for dis-

tributed machine learning," *Progress in Artificial Intelligence*, vol. 2, no. 1, pp. 1–11, 2013.

[85] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A Survey on Distributed Machine Learning," *ACM Computing Surveys*, vol. 53, pp. 1–33.

[86] S. Savazzi, M. Nicoli, and V. Rampa, "Federated Learning with Cooperating Devices: A Consensus Approach for Massive IoT Networks," *IEEE Internet Things J.*, pp. 1–1, 2020, arXiv: 1912.13163. [Online]. Available: http://arxiv.org/abs/1912.13163

[87] A. Lalitha, S. Shekhar, T. Javidi, and F. Koushanfar, "Fully decentralized federated learning," in *Third workshop on Bayesian Deep Learning (NeurIPS)*, 2018.

[88] A. Lalitha, O. C. Kilinc, T. Javidi, and F. Koushanfar, "Peer-to-peer federated learning on graphs," *arXiv preprint arXiv:1901.11173*, 2019.

[89] J. George and P. Gurram, "Distributed deep learning with event-triggered communication," *arXiv preprint arXiv:1909.05020*, 2019.

[90] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "Conflict-free replicated data types," in *Symposium on Self-Stabilizing Systems*. Springer, 2011, pp. 386–400.

[91] P. Ramanan, K. Nakayama, and R. Sharma, "Baffle: Blockchain based aggregator free federated learning," *arXiv preprint arXiv:1909.07452*, 2019.

[92] A. Sadeghian, V. Kosaraju, A. Gupta, S. Savarese, and A. Alahi, "Trajnet: Towards a benchmark for human trajectory prediction," *arXiv preprint*, 2018.

[93] S. Becker, R. Hug, W. Hübner, and M. Arens, "An evaluation of trajectory prediction approaches and notes on the trajnet benchmark," 2018.

[94] I. Gilitschenski, G. Rosman, A. Gupta, S. Karaman, and D. Rus, "Deep context maps: Agent trajectory prediction using location-specific latent maps," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, p. 5097–5104, Oct 2020. [Online]. Available: http://dx.doi.org/10.1109/LRA.2020.3004800

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

[96] J. van den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics Research*, C. Pradalier, R. Siegwart, and G. Hirzinger, Eds., 2011, pp. 3–19.

[97] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, p. 1735–1780, Nov. 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[98] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

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

[100] "Argos large-scale robot simulations: examples," https://www.argos-sim.info/examples.php, accessed: 2020-04-15.

[101] G. Valentini, D. Brambilla, H. Hamann, and M. Dorigo, "Collective Perception of Environmental Features in a Robot Swarm," in *Swarm Intelligence*. Cham: Springer International Publishing, 2016, pp. 65–76.

[102] J. T. Ebert, M. Gauci, F. Mallmann-Trenn, and R. Nagpal, "Bayes bots: Collective bayesian decision-making in decentralized robot swarms," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 7186–7192.

[103] L. Riazuelo, M. Tenorth, D. Di Marco, M. Salas, D. Gálvez-López, L. Mösenlechner, L. Kunze, M. Beetz, J. D. Tardós, L. Montano *et al.*, "Roboearth semantic mapping: A cloud enabled knowledge-based approach," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 432–443, 2015.

[104] R. Khodayi-mehr, Y. Kantaros, and M. M. Zavlanos, "Distributed state estimation using intermittently connected robot networks," *IEEE Transactions on Robotics*, vol. 35, no. 3, pp. 709–724, 2019.

[105] N. Atanasov, J. Le Ny, K. Daniilidis, and G. J. Pappas, "Decentralized active information acquisition: Theory and application to multi-robot SLAM," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 4775–4782.

[106] K. Y. Leung, T. D. Barfoot, and H. H. Liu, "Decentralized cooperative slam for sparsely-communicating robot networks: A centralized-equivalent approach," *Journal of Intelligent & Robotic Systems*, vol. 66, no. 3, pp. 321–342, 2012.

[107] R. Polikar, *Ensemble Learning*, C. Zhang and Y. Ma, Eds. Boston, MA: Springer US, 2012.

[108] N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford *et al.*, "The limits and potentials of deep learning for robotics," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 405–420, 2018.

[109] L. I. Kuncheva, *Combining pattern classifiers: methods and algorithms*. John Wiley & Sons, 2014.

[110] M. A. Uy, Q.-H. Pham, B.-S. Hua, T. Nguyen, and S.-K. Yeung, "Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 1588–1597.

[111] G. Valentini, E. Ferrante, and M. Dorigo, "The best-of-n problem in robot swarms: Formalization, state of the art, and novel perspectives," *Frontiers in Robotics and AI*, vol. 4, p. 9, 2017. [Online]. Available: http://journal.frontiersin.org/article/10.3389/frobt.2017.00009

[112] M. Crosscombe, J. Lawry, S. Hauert, and M. Homer, "Robust distributed decision-making in robot swarms: Exploiting a third truth state," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 4326–4332.

[113] A. Mitra, J. A. Richards, S. Bagchi, and S. Sundaram, "Resilient distributed state estimation with mobile agents: overcoming byzantine adversaries, communication losses, and intermittent measurements," *Autonomous Robots*, vol. 43, no. 3, pp. 743–768, 2019.

[114] A. Tahbaz-Salehi and A. Jadbabaie, "On consensus over random networks," in *44th Annual Allerton Conference*. Citeseer, 2006.

[115] L. Xiao, S. Boyd, and S.-J. Kim, "Distributed average consensus with least-mean-square deviation," *Journal of Parallel and Distributed Computing*, vol. 67, no. 1, pp. 33–46, Jan. 2007.

[116] H. J. Leblanc, H. Zhang, X. Koutsoukos, and S. Sundaram, "Resilient asymptotic consensus in robust networks," *Ieee Journal on Selected Areas in Communications*, vol. 31, no. 4, 2013.

[117] D. Albani, D. Nardi, and V. Trianni, "Field coverage and weed mapping by uav swarms," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ieee, 2017, pp. 4319–4325.

[118] P.-Y. Lajoie, B. Ramtoula, Y. Chang, L. Carlone, and G. Beltrame, "DOOR-SLAM: Distributed, online, and outlier resilient slam for robotic teams," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1656–1663, 2020.

[119] J. P. Queralta, J. Taipalmaa, B. C. Pullinen, V. K. Sarker, T. N. Gia, H. Tenhunen, M. Gabbouj, J. Raitoharju, and T. Westerlund, "Collaborative multi-robot systems for search and rescue: Coordination and perception," *arXiv preprint arXiv:2008.12610*, 2020.

[120] J.-R. Ruiz-Sarmiento, C. Galindo, and J. Gonzalez-Jimenez, "Building multiversal semantic maps for mobile robot operation," *Knowledge-Based Systems*, vol. 119, pp. 257–272, 2017.

[121] A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecka, "3d bounding box estimation using deep learning and geometry," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7074–7082.

[122] H. Carrillo, K. H. Brodersen, and J. A. Castellanos, "Probabilistic performance evaluation for multiclass classification using the posterior balanced accuracy," in *ROBOT2013: First Iberian Robotics Conference*. Springer, 2014, pp. 347–361.

[123] B.-S. Hua, Q.-H. Pham, D. T. Nguyen, M.-K. Tran, L.-F. Yu, and S.-K. Yeung, "Scenenn: A scene meshes dataset with annotations," in *International Conference on 3D Vision (3DV)*, 2016.

[124] T. G. Crainic, G. Perboli, W. Rei, and R. Tadei, "Efficient lower bounds and heuristics for the variable cost and size bin packing problem," *Computers & Operations Research*, vol. 38, no. 11, pp. 1474–1482, 2011.

[125] G. E. Andrews, *The theory of partitions*. Cambridge university press, 1998, no. 2.