Titre: Title:	A Runtime Environment for Micro and Macro UAV Swarms
Auteur: Author:	Ulrich Dah-Achinanon
Date:	2022
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Dah-Achinanon, U. (2022). A Runtime Environment for Micro and Macro UAV Swarms [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/10330/

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

A runtime environment for micro and macro UAV swarms

ULRICH DAH-ACHINANON
Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Mai 2022

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

A runtime environment for micro and macro UAV swarms

présenté par **Ulrich DAH-ACHINANON**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Michel DAGENAIS, président
Giovanni BELTRAME, membre et directeur de recherche
Tarek OULD BACHIR, membre

DEDICATION

"Education is the most powerful weapon which you can use to change the world."
– Nelson Mandela

ACKNOWLEDGEMENTS

The work presented in this thesis was made possible by the continuous support of many people. I would like to first thank Giovanni Beltrame, Seyed Ehsan Marjani Bajestani, Pierre Yves Lajoie, and all the members of the Polytechnique Montreal's MIST lab for their expertise, mentorship, help and collaboration throughout my master thesis.

I would also like to thank David St-Onge, and the members of the INIT Robots lab of ÉTS for their help, and for allowing me to learn and discover other facets of robotics during my time as a research assistant in their lab.

Finally, I would like to thank my family and friends for their unconditional emotional and, financial support during all these years.

RÉSUMÉ

Un essaim de véhicules aériens sans pilote (UAV) est un groupe de robots volants ou de drones exploitant les capacités de la robotique en essaim pour atteindre un objectif commun. Les drones attirent de plus en plus l'attention depuis quelques années grâce à leurs multiples applications dans des domaines tels que : le militaire, l'agriculture, la recherche et sauvetage, et plus encore. Lorsqu'elles sont couplées à la robotique en essaim, les applications des drones augmentent considérablement et ajoutent de la robustesse, de la flexibilité, de la tolérance aux pannes et de la rapidité d'exécution aux applications existantes. Bien qu'intéressantes, la plupart de ces applications potentielles sont uniquement théoriques, ce qui crée un manque de déploiements réels d'essaims de drones. Passer à des applications dans le monde réel nécessite plusieurs considérations, dont les réseaux ad hoc, l'interaction des utilisateurs, les outils de développement spécifiques au domaine et l'acquisition de télémétrie.

Ce mémoire propose de combler le fossé entre la théorie et la réalité pour les essaims de drones à travers deux réalisations concrètes. Tout d'abord, un outil de développement logiciel conçu pour les robots à ressources limitées, appelé BittyBuzz, a été développé. BittyBuzz est un environnement d'exécution du langage Buzz et a été conçu pour les microcontrôleurs. Il a été intégré à plusieurs plates-formes robotiques, dont Bitcraze Crazyflie, un nano quadricoptère utilisé dans la recherche et l'éducation. Deuxièmement, une application d'essaim de drones dans le monde réel a été conçue et déployée pour les opérations de recherche et de sauvetage. L'application utilise un modèle de recherche basé sur le point de rendez-vous et forme une arborescence de relais pour l'étape de sauvetage, permettant la connectivité entre les opérateurs au sol et les victimes. L'implémentation a été réalisée en utilisant ROSBuzz, un nœud ROS existant qui encapsule la machine virtuelle Buzz.

Afin de prouver l'efficacité de l'outil de développement logiciel pour les robots à ressources limitées, des expériences en laboratoire ont été réalisées. Ces dernières ont utilisé un essaim de micro drones, Crazyflie de Bitcraze, pour implémenter plusieurs algorithmes bien connus dans la robotique en essaim. Pour évaluer l'approche adoptée pour l'algorithme SAR et confirmer sa capacité à être mis à l'échelle, nous avons effectué des tests de simulation avec des tailles d'essaims variables. La simulation a ensuite été confirmée par des expériences en conditions réelles, utilisant un essaim de macro UAV composé de drones commerciaux. Ces expériences ont prouvé la facilité d'utilisation de l'algorithme dans une situation réelle. Dans les deux cas, les résultats ont confirmé que les systèmes développés peuvent être utilisés pour des applications d'essaim de drones, tout en offrant de bonnes performances.

ABSTRACT

A swarm of Unmanned Aerial Vehicles (UAVs) is a group of flying robots or drones that leverage the capabilities of swarm robotics to achieve a common goal. UAVs have been attracting an increasing level of attention for the past few years thanks to their multiple applications in fields such as: military, agriculture, search and rescue, and more. When coupled with swarm robotics, the applications of UAVs drastically increase while adding robustness, flexibility, fault tolerance and speed of execution to the existing applications. Although interesting, most of these potential applications are solely theoretical, creating a lack of real-world deployments of UAVs swarms. Moving toward real-world applications requires multiple considerations, including but not limited to ad-hoc networks, user interaction, widespread domain specific development tools and telemetry acquirement.

This thesis proposes to bridge the gap between theory and reality for UAV swarms through two concrete realizations. First, a software development tool designed for resource constrained robots, called BittyBuzz was developed. BittyBuzz is a runtime environment for the domain-specific language Buzz, and was designed for microcontrollers. It was integrated with multiple robotic platforms, including Bitcraze Crazyflie, a nano quadcopter used in research and education. Second, a real-world UAV swarm application was designed and deployed for search and rescue operations. The application uses a search pattern based on rendezvous point and forms a relay tree for the rescue stage, allowing connectivity between ground operators and victims. The implementation was realized by using ROSBuzz, an existing ROS node that encapsulates the Buzz Virtual Machine.

In order to prove the efficiency of the software development tool for resource constrained robots, some laboratory experiments were performed. The latter used a swarm of micro UAVs, Bitcraze's Crazyflie, to implement multiple well-known algorithms in swarm robotics. To evaluate the approach adopted for the SAR algorithm and confirm its ability to be scaled, we went through simulation tests with variable swarm sizes. The simulation was then confirmed by real-world experiments, using a macro UAVs swarm made up of commercial drones. These experiments proved the usability of the algorithm in a real-world situation. In both cases, the results confirmed that the developed systems can be used for UAVs swarm applications, providing good performance.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

BATMAN	Better Approach to Mobile Ad-hoc Networking
BF	Bayesian Factor
BVM	Buzz Virtual Machine
BBZVM	BittyBuzz Virtual Machine
DMR	Distributed mobile robotics
DSL	Domain specific language
FCU	Flight Control Unit
GNSS	Global Navigation Satellite systems
LSAR	Layered Search and Rescue
OSDK	Onboard Software development Kit
UAV	Unmanned Aerial Vehicle
UVS	Unmanned vehicle system
ROS	Robot Operating System
RPAS	Remotely piloted aircraft systems
RPV	Remotely piloted vehicle
SAR	Search and Rescue
VS	Virtual Stigmergy

CHAPTER 1 INTRODUCTION

1.1 Definitions and Basic Concepts

To have the necessary background to understand the work realized in this thesis, multiple concepts needs to be known. This section provides these basic concepts and some definitions to aid understanding.

1.1.1 Swarm Robotics

Swarm robotics is a field of robotics that applies swarm intelligence to groups of robots for their coordination. [1] defines it as an approach to collective robotics inspired by the interaction of social animals. It features larges groups of simple robots, aiming to accomplish a complex task through simple rules and local interactions. To be considered a swarm robotics system, a setup should satisfy multiple characteristics. In fact, the robots should [1]:

- be autonomous;
- be situated in the environment and able to act to modify their position;
- have local sensing and communication capabilities;
- cooperate to tackle a specific task;
- not rely on centralized control and/or global knowledge.

1.1.2 Unmanned Aerial Vehicles

Based on the unmanned vehicle system (UVS) international definition, an unmanned aerial vehicle (UAV) is a generic aircraft conceived to perform without the need for the presence of a human onboard. It is known under different terms, such as: remotely piloted vehicle (RPV), remotely piloted aircraft systems (RPAS), micro-aerial vehicles, to name a few. Until a few years ago, the development of UAVs was mainly motivated by its military applications, but nowadays, they became a valuable source of data for inspection, surveillance, mapping, and more. In fact, they are considered a low-cost alternative to the classical manned options. Therefore, UAVs have been used for multiple applications including: disaster management [2], precision agriculture [3], forestry, archeology and architecture. [4]

1.1.3 Real-world/Laboratory Experiments

In swarm robotics, most of the experiments carried out to test a newly developed algorithm are done in laboratory conditions rather than in real-world. In fact, deploying these applications in real-world can be time-consuming. Also, the requirements in robotics platforms can often be associated with a high cost. To overcome this problem, multiple existing projects developed small and low-cost robots to allow researchers to test their algorithm in a controlled, smaller, laboratory settings. Among these projects we have: Kilobots [5], Zooids [6], Bitcraze Crazyflie [7], to only name those.

Although useful and constituting an upgrade in comparison to simple tests in simulation, laboratory experiments introduce some simplifications that can make the experiments less relevant to the real-world. Laboratory environments are controlled and therefore reduce the unpredictability we have in real-world. For instance, connectivity is rarely an issue in the laboratory because the robots are relatively close to one another, allowing a connected swarm. However, when we leave the lab, as the environment to explore gets bigger, the connectivity can easily be lost in the swarm, creating a sparse swarm.

1.1.4 Domain Specific Language

A domain specific language is a programming language or executable specification language that provides a notation specific to an application domain and is based on the important concepts and features of that domain. To properly design a DSL, a good understanding of the application domain is required. In fact, that knowledge allows the designer to efficiently provide the necessary tools, or expressive power, in order to generate members of a family of programs in the domain. The advantages of DSLs include: reduced time to market, reduced maintenance costs, and higher portability. [8]

In swarm robotics, multiple projects attempted to create a DSL tailored to the needs in the domain. Buzz [9] for example, is a DSL for swarm robotics that intends to ease the development of swarm applications by providing high-level principles such as: discrete swarm, step wise execution, heterogeneous robots support, situated communication, swarm level abstraction, and information sharing.

1.1.5 Runtime Environment

A runtime environment is a software and hardware infrastructure that supports the execution of a specific codebase in real time. It is the environment supporting the execution of a program or application. The behavior of a program and the outcome of the execution highly depend

on the environment in which the program is executed. That observation adds importance to the role of the runtime environment, particularly for robotic platforms. In fact, resources have to be properly managed for an optimal operation of the application. [10]

Therefore, the goal of the runtime environment for resource constrained robots that will be presented in the following sections, is to support the execution of Buzz scripts on these platforms. The search and rescue application presented in the second part of the thesis used an existing runtime environment based on ROS (Robot operating system) and Buzz, and added the necessary adapter for the proper operation of the system.

1.1.6 Resource Constrained Microcontrollers

Typically, a microcontroller system is made of a processor core, an on-chip SRAM block and an on-chip embedded flash. When flashing a program into a microcontroller, the binary is usually preloaded into the flash and then transferred to the SRAM at startup. The SRAM is then used as primary data storage for the execution. Therefore, a program's memory footprint is an important factor when developing for microcontrollers. As indicated in [11], the tight resource constraints of embedded systems affect the way the developer programs. For microcontrollers, the performance is also a constraining factor, since they are designed for applications with low cost and high energy-efficiency. [12]

1.2 Problem Statement

This section presents the elements of the problematic resolved in this thesis. It states the observations that led to the work and the solutions developed in the following sections. These observations can be categorized in two groups: software development for microcontrollers in swarm robotics and real-world UAVs swarm applications.

1.2.1 Software Development for Resource Constrained Robots Swarm

Despite the enthusiasm surrounding swarm robotics, software development in this field is still a tedious task. That fact is partly due to the lack of dedicated solutions, in particular for resource constrained robots. However, multiple resource constrained systems are of use in swarm robotics. In fact, due to the high cost of macro robots, researchers often resort to low-cost, micro robots for their laboratory experiments. The reduced resources available on these robots make most of the few existing domain specific tools unusable for software development. To ease the developer's task, some tools are therefore necessary to provide

useful features for swarm application development. The work carried out in this thesis fits into that perspective.

1.2.2 Lack of UAVs Swarm Real-world Deployment

Research in the field of swarm robotics has been focusing on algorithmic innovations, somehow neglecting the real-world applications. However, to access the innumerable benefits that swarm robotics can offer, real-world applications should be the center of attention. Solving that problem requires a combination of research from different fields, to provide reusable, real-world deployments of swarms. This thesis focuses on UAVs swarm real-world deployment, but that statement is valid for all kinds of robots.

1.2.3 Network Issues in Real-world Applications

When we move from the laboratory to the real-world, multiple issues appears for UAVs swarm applications. To aid that transition, real-world deployments should consider factors such as: sporadic connectivity, telemetry acquisition, user interaction, and mission start/stop, to name a few. Even though all these factors are taken into account for the solution provided in this work, the sporadic connectivity is the one that was extensively analyzed and solved. In fact, a swarm of UAVs should be able to jointly accomplish a task without assuming a constant connection among the members. A real-world swarm application should therefore support a sparsely connected swarm, which is not the case for most of the existing applications.

1.3 Research Objectives

The global objective of this thesis is to bring swarm robotics one step closer to widespread real-world deployments of UAVs swarms. To do so, a runtime environment for micro robots should be developed and integrated with a commonly used micro UAV, to help laboratory experimentation. Real-world applications of UAVs swarm should then be designed and deployed to propose a reusable and functional system. The chosen application in this thesis is the search and rescue operation with a UAVs swarm. For the micro UAVs runtime environment, the specific objectives are the following:

- Develop a swarm oriented runtime environment capable of specifying the behavior of heterogeneous swarms of robots with as small as 32 KB of flash and 2 KB of RAM;
- Extend and optimize the capabilities of an existing domain specific language (DSL) to be useable on resource-constrained platforms;

- Integrate the developed environment with different robotic platforms used for research in swarm robotics;

Concerning the search and rescue application, the specific objectives aim to:

- Develop a search pattern based on rendezvous point, using ad-hoc networks with adaptive topology and routing, and accepting sparsely connected swarms;
- Develop and implement an algorithm to create an adaptive relay tree structure design to maintain the communication between the ground operators and the discovered targets;
- Implement a search method based on dynamic, distributed belief maps;
- Design an experimental robotic system for real-world deployment of swarms of drones;
- Realize simulation and real-world deployment of the proposed system.

1.4 Thesis Outline

The remainder of this thesis is divided into five sections. First, Chapter 2 presents recent state-of-the-art litterature, relevant to this thesis. Chapter 3 provides an overview of the research approach and an overall organization of the thesis, while explaining the consistency between the articles and the research objectives. In Chapter 4, BittyBuzz the runtime environment designed for resource constrained robots used in swarm robotics, is presented. Chapter 5 presents the article on search and rescue with sparsely connected swarms. Finally, Chapter 6 presents the conclusion, with a summary of the works presented in the thesis, a discussion of the limitations and a suggestion for future avenues of research.

CHAPTER 2 LITERATURE REVIEW

This chapter presents recent state-of-the-art literature, relevant to the field of this thesis. It includes a generic overview of the following topics: search and rescue (SAR) mission with UAVs swarms, rendezvous point based exploration, belief maps based exploration, sparse swarms, relay topology to aid communication, resource constrained oriented frameworks, and software development tools for swarm. All the articles presented in this section are not directly used in the developed systems, but they give useful insights into the field of the thesis and can help better understand the problem.

2.1 Search and Rescue with UAVs Swarms

As discussed in [13], UAVs swarms have clear and useful applications in search and rescue. In fact, SAR activities can be time-consuming and requires a lot of man-power. Using UAVs swarms for this task can highly optimize the search and rescue process. With aerial real-time images of the search area, the ground crew can optimize the search zones and possibly build a belief map for the search.

Recognizing the usefulness of drone swarms for this application, many articles discuss effective ways to perform it. For instance, [14] proposes a mission planning platform, that efficiently deployed a swarm of UAVs to cover complex-shaped areas in various applications. To demonstrate the applicability of the platform in real-life situation, a search and rescue mission was performed in the tests. The results showed a faster exploration and a good battery usage with a swarm of UAVs. The mission duration were directly proportional to the number of UAVs deployed. Therefore, a swarm performs better in SAR than a unique UAV.

The work presented in [15] reinforces that observation, with their simulation results indicating that a better success rate could be achieved by increasing the number of robots. Their paper proposes a layered search and rescue (LSAR) centralized "partitioning" algorithm, that needs reliable communication with a cloud server to operate properly. This algorithm need for a reliable communication raises one of the main concern in real-world deployment of UAVs swarm: the need for a robust communication network.

To address the communication issue, [16] introduced an optimized self-organized mesh network to cover large areas. The idea is to allow the UAVs to adjust their position, to maintain the connection within the swarm during an exploration. The approach uses genetic algorithms to achieve multi-node exploration, with emphasis on connectivity and swarm spread.

2.2 Rendezvous Point Exploration

In real-world applications of swarm robotics, proximity between the agents almost guarantees a reliable communication network. Therefore, having a meeting point during the mission in order to exchange information, have been used in multiple applications, some of which will be presented. [17] considered a periodic rendezvous strategy in order to overlap the communication ranges of the robots. It mainly presents an approach to mitigate the negative impact of these meetings on the time efficiency of the overall mission. Since the robots do not gain additional knowledge on their way to the meeting, the algorithm proposed to mitigate this negative impact.

The works presented in [18] and [19] describe an ant algorithm, whose objective is to allow multiple robots to autonomously explore an unknown environment. The approach also leverages a rendezvous strategy, but the robots only meet at the determined point, when the exploration is completed.

For its part, [20] uses independent distributed particle swarm Optimization algorithms to develop a distributed UAVs fleet control approach. The objective of the approach is to plan a 2D exploration in order to converge to a predefined spatial configuration around a rendezvous point. Such an approach is useful in applications such as SAR and large area coverage, to name a few. [21] also presents an exploration algorithm for unknown environments based on rendezvous point. The algorithm requires a limited communication between the robots, so they only communicate at the rendezvous.

2.3 Belief Maps

Belief maps have been studied for a long time as a tool for multi-robot exploration [22, 23]. They provide information on the parts of a map that are more likely to contain the target, reducing the search area. [24] for example, develop a novel exploration approach for UAVs operating in unknown environments. The approach leverages a probabilistic information gain map, called a belief map. The map is used as a prior to guide the exploration trajectory, while efficiently reducing false positive information in the process.

Another approach proposes to determine the impact of cooperation and data exchange on the search time during multi-UAV cooperative search [25]. To do so, it presented some strategies for merging occupancy probabilities of target existence. The approach present, among others, a map merging strategy that updates the belief map with local observations and merges data from multiple UAVs. Recently, [26] presented an exploration system that leverages decentralized information sharing to update a common risk belief map. The shared

belief map is stored in a distributed database called virtual stigmergy [27]. Their approach allows a group of robots to explore unknown environments with a high coverage rate.

2.4 Sparse Swarms

In swarm robotics, most applications are developed with the assumption that the robots members of the swarm will operate in proximity of each other, forming a dense swarm [28]. Although, many of the real-world applications of swarms require the agents to jointly accomplish a mission while operating over large distances [28]. Among these applications we have: precision agriculture, SAR missions, and environmental monitoring, to name a few [28]. In fact, as mentioned by [29], a reliable communication structure is essential to share information among group of neighbors in swarm applications. Sharing the information becomes essential in applications, such as SAR, where a group of robots is trying to find a target in a large area. These situations make the use of dense swarms impractical and create the need for a new concept to formalize such a scenario: sparse swarms [28]. The work in [28] is dedicated to the theoretical formulation of the concept and the illustration of its necessity.

Also addressing the sparse swarms issue, [30] proposes an exploration and mapping algorithm for sparse swarms of robots, which completes a full exploration even in the extreme case of a single robot. The algorithm, called atlas, outperformed state-of-the-art algorithms such as Ramaithitima, that do not take into account the sparse swarms use case.

2.5 Communication with Relay Topology

Using a relay topology to broaden the communication range is a technique used in multiple applications and has been explored in this thesis for our SAR application. In fact, during the rescue phase of SAR missions, it is crucial to maintain the connection between the base station and the UAVs following the target. This section will present existing literature that use the relay technique for diverse purposes.

Using a heuristic optimization method, [31] increased the communication performance metric and determined the optimal positions for the communication relay robots in multi-node networks. The algorithm outperformed a recently-developed relay positioning algorithm in the simulations. To keep the connection between a heterogeneous group of robots (on-ground and flying robots), [32] introduced a fully decentralized algorithm to create and keep a chain of robots from the ground station to the target. [33] and [34] also used drones as relays and virtual potentials to create a stable link between a group of robots (or a survey drone) and a base station. Instead of using virtual potentials, [35] measured the communication quality

and expanded the drone relays when needed. While the above-mentioned works focused on the formation of relay chains during the search phase, this could make the search process very slow, which would be critical for SAR operations. Therefore, we propose to create the communication relay chains only in the rescue phase. Our communication relay approach is similar to the approach proposed by [36,37], in which the creation and expansion of tree/chain topologies between drones and target(s) have been evaluated. We use a tree topology for the relay connection when rescuing more than one target.

2.6 Resource Constrained Oriented Frameworks

Micropython [38] is a lightweight and efficient implementation of the programming language Python 3. That implementation is optimized to run on microcontrollers and in constrained environments. It is composed of a full Python compiler, a runtime and a subset of python standard library while able to fit a 256 KB code space and run with 16 KB RAM. Micropython has been proven to be a good option for rapid development of IoT devices with tested and verified libraries [39]. Currently, its main use targets are: development and testing of device, sensor designs, monitoring and configuration tool in design of complex applications, and education purposes [39]. For instance, [40] presents a prototype of an educational mobile robot based on MicroPython.

Another development tool for microcontrollers is Artoo [41], a micro-framework conceived for robotics. The framework provides a powerful DSL for robots control and physical computing. It works on Ruby and borrows some concepts and code from Sinatra [42]. Supporting 15 different platforms among which the Bitcraze Crazyflie, Artoo allows developers to create solutions that incorporate multiple, different hardware devices at the same time. However, Artoo is not designed for swarm application conception; it allows the user to connect to one or multiple robots and manipulate them with a centralized coordination.

Zephyr project [43] is also a project worth mentioning that units developers and users in building a small, scalable RTOS optimized for resource-constrained devices on multiple architectures. The goal of the project is to create an open, collaborative environment to deliver a RTOS that will answer the changing demand of the connected devices space. The system supports multiple boards and is easily portable across platforms. It has a different approach to the resource constraint problem when compared to the system proposed in this thesis, simply offering a configurable RTOS that can be adapted depending on the user's needs.

Another project is EmSBoT [44], that presents a component-based framework targeting resource-constrained devices (cost 13 KB of flash memory and 5 KB of RAM). It is built

upon µCOS-III with real-time support. Even though it is network-focused, this framework supports the development of swarm robotics applications.

OpenSwarm [45] is an OS designed for severely computationally constrained robots (costs 1 KB RAM and 12 KB ROM). It enables the developer to design platform independent solutions, that can easily be applied to swarm robotics, as showed in the experiments from the paper.

2.7 Software Development Tools for Swarms

With swarm robotics increasing in popularity, multiple projects aimed to provide software development tools to ease the implementation of new behaviours. For instance, Koord [46] is a swarm oriented language developed to make platform-independent code portable and modularly verifiable. It proposes various useful features for coordination in a swarm, such as shared variables between the robots and state recording. However, Koord does not focus on resource constraints, which is the focus in this thesis.

Another example is DRONA [47], a framework for building reliable distributed mobile robotics (DMR) applications. It provides a state-machine based language (P) for event-driven programming. P is a high level language that, once compiled, generate a C code that can be directly deployed in ROS. Protelis [48] is another language developed to provide a practical and universal platform for aggregate programming. Aggregate programming, unlike the traditional device-centric programming, is concerned in the behaviour of a collection of devices - that can be assimilated to a swarm of robots. The language is hosted in and integrated with Java. In [49] an actor-based programming framework is proposed for swarm robotic systems. It presents a bottom-up programming approach based on a new control unit, the 'Actor'. The latter is the virtualisation of the capabilities of a given robot, and it allows developers to design cooperative tasks without intricacies about details such as robotic algorithms and specific robot brands.

Buzz [9] is a swarm specific language that has been developed to address the lack of software development tools in swarm robotics. It offers a multitude of interesting features for swarm application development, including but not limited to: heterogeneous support, swarm-level abstraction, neighbor operations and virtual stigmergy. The system proposed in this thesis is a new implementation of that DSL.

CHAPTER 3 RESEARCH APPROACH AND THESIS ORGANIZARION

This section presents an overview of the research approach and an overall organization of the thesis. It explains the link between the articles presented in chapter 4 and 5 and their consistency with the research objectives mentioned earlier. The approach adopted in this research can be seen as following two distinct axes. The first is from a laboratory environment perspective to the real-world deployment. The second axis is from a micro UAV swarms point of view to the particularities of macro UAV swarms.

3.1 From Laboratory Experiments to Real-world Deployment

Getting to widespread real-world applications of UAV swarms, requires some thorough testing in simulation and laboratory settings in order to validate the implemented applications. Laboratory experiments are therefore essential to safe and functional real-world deployments. Indeed, laboratory tests can be done faster as they are usually easier to set up and access. Also, contrary to real-world experiments, they do not depend on natural phenomenons that can be unpredictable.

Aware of the importance of lab testing, this thesis first presents, in Article 1, a runtime platform for resource constrained robots aimed to laboratory experiments. After testing various algorithms in a controlled laboratory settings with the mentioned runtime environment, they were deployed in real-world applications. This is how Article 2 presents a real-world application that was deployed on commercial drones. The UAV swarm application that was deployed is a search and rescue application allowing sparsely connected swarms.

3.2 From Micro UAV Swarms to Macro UAV Swarms

UAVs come in different sizes from micro quadrotors to macro UAVs such as the commercial or military drones. The specifications of micro UAVs make them the perfect tools for research experiments. In fact, they take up less space and can easily be used to verify an algorithm indoor. However, most of these micro UAVs can't be used to carry actual swarm applications in real-world settings because of their size and their resource constraints, among other things. Macro UAVs are therefore needed to reach the goal of widespread UAV swarms applications, while the micro UAVs are necessary for verification and preliminary tests purposes.

This thesis addresses these needs by first proposing, in article 1, a runtime environment focused on micro swarms. The runtime platform allows a developer to easily implement

and test a swarm application in laboratory settings. Article 2 then proposes a framework that can be used to deploy in real-world conditions the lab-tested application using macro swarms. Both frameworks use the DSL Buzz for the behavior implementation. Thus, going from micro swarms to macro swarms is facilitated, as the core behavior implementation will need little to no change.

3.3 Document Structure

The document follows the recommended layout for a thesis by articles. The submitted articles are included in the body of the work as separate chapters. The document is structured as follow:

- Chapter 1 introduces the research, by giving some insights into some basic concepts and stating the elements of the problematic.
- Chapter 2 is an overview of the state-of-the-art literature relevant to this research.
- Chapter 3, the current one, presents the research approach and the overall thesis organization.
- In Chapter 4, BittyBuzz, a runtime platform with dynamic memory management that was designed for microcontrollers, is presented. This work has been submitted to IEEE Transactions on Computers in March 2022.
- Chapter 5, proposes a real-world deployment: a decentralized search system that only requires sporadic connectivity and allows information diffusion through a swarm of UAVs whenever possible. This work was submitted to Autonomous Robots in November 2021.
- Chapter 6, concludes this thesis by presenting a summary of works, the limitations of the solutions proposed and future research directions.

CHAPTER 4 ARTICLE 1 : BITTYBUZZ: A SWARM ROBOTICS RUNTIME FOR TINY SYSTEMS

Preface: Software development for swarm applications can be a time-consuming activity, in particular for resource-constrained systems. In order to ease that task, frameworks dedicated to swarm robotics applications are needed. This chapter presents BittyBuzz, a full virtual machine for a dynamically typed language with dynamic memory management. It works on microcontrollers with as little as 32 kB of flash and 2 kB of RAM. The runtime platform is designed to support the execution of scripts written with the DSL Buzz, on micro swarms.

The correct operation of BittyBuzz is confirmed through some experiments, mainly performed with micro swarms of Bitcraze Crazyflie: a nano quadcrotor used in fields such as: research and education.

Full Citation: Dah-Achinanon U, Belhaddad EK, Beltrame G, "BittyBuzz: a swarm robotics runtime for tiny systems", IEEE Transactions on Computers, [under review] 2022.

Abstract - Swarm robotics is an emerging field of research which is increasingly attracting attention thanks to the advances in robotics and its potential applications. However, despite the enthusiasm surrounding this area of research, software development for swarm robotics is still a tedious task. That fact is partly due to the lack of dedicated solutions, in particular for low-cost systems to be produced in large numbers and that can have important resource constraints. To address this issue, we introduce BittyBuzz, a novel runtime platform: it allows Buzz, a domain-specific language, to run on microcontrollers while maintaining dynamic memory management. BittyBuzz is designed to fit a flash memory as small as 32 KB (with usable space for scripts) and work with as little as 2 KB of RAM. In this work, we present the BittyBuzz implementation, its differences from the original Buzz virtual machine, and its advantages for swarm robotics systems. We show that BittyBuzz is successfully integrated with three robotic platforms, and conduct extensive experimentation with a swarm of nano-quadcopters (Crazyflie from Bitcraze) to show the computation and memory performance of BittyBuzz. The results show that BittyBuzz can be effectively used to implement common swarm behaviors on microcontroller-based systems.

4.1 Introduction

Swarm robotics is a research area that studies coordination in groups of relatively simple robots, leveraging local interactions to generate emergent global behaviors. It is inspired by societies of insects, where groups can achieve tasks beyond the capabilities of the individuals [50]. With properties such as scalability, robustness, autonomy, decentralization [51], swarm robotics allows a wide variety of applications, including nanomedicine, space exploration, search and rescue missions, and generally tasks in dangerous areas [51]. Despite all these possible applications, most of the field of swarm robotics is still firmly at the research stage. For reasons related to cost, time, space and complexity, swarm applications, requires small and low-cost robots. Multiple robotic platforms meeting these specifications are widely used for research (e.g. the Kilobot [52]). Due to their low-cost, such systems come with a limited amount of computing resources and sometimes make swarm-oriented software solutions developed in simulation unusable. If programming cooperative behaviours for swarm robotics has always been a challenging task [49], doing it for resource-constraint swarm systems only makes it harder.

In fact, the development process for swarm robotics applications is sometimes a slow and tedious task requiring a lot of effort from the developer. The lack of dedicated tools focusing on swarm interactions is to blame for this situation: it forces developers to "reinvent the wheel" when trying to integrate well-known algorithms to new applications. To solve the problem, some previous work proposed domain specific languages (DSLs) to raise the level of abstraction and provide common programming primitives, easing the development task. For instance, Koord [46] is an event-driven language which uses shared variables for coordination in distributed robotics applications, with capabilities applicable to swarm robotics. Protelis [48] is also an example of DSL used for aggregate programming. In the same spirit, the framework DRONA [47] proposes the P language for distributed mobile robots applications development. Pinciroli and al. created Buzz [9], a DSL for heterogeneous robot swarms. Buzz gives a configurable level of abstraction (swarm or individual robot) to the developer depending on their needs. It also lays down some important primitives that are necessary for swarm-oriented programming.

Unfortunately, these DSLs do not tackle the resource limitation problem in their implementations and may therefore be unusable in many practical applications. In fact, even the Buzz virtual machine, which in theory only takes 12 kB of memory, can quickly fill the flash when combined with the native firmware of certain robots. Furthermore, the RAM consumption during applications execution can easily exceed the one available on some robots (as little as 2 kB for the Kilobot). Faced with such an issue, we propose a system offering the same

capabilities as Buzz for swarm systems composed of resource constrained robots.

In particular, this system should offer, to name a few: swarm level abstractions, heterogeneous robots support, communication neighbor operations. The system should also be modular, allowing the user to reduce resource consumption by selecting the features to use depending on the implemented behaviour. It should be able to fit systems like the Kilobot with 32 kB of flash and 2 kB of RAM, and other resource constrained robotic platforms used for research in swarm robotics.

Answering these needs, this article introduces BittyBuzz an implementation of the Buzz Virtual Machine for microcontrollers. This implementation roughly uses the same structure as Buzz and supports 100% of its code, with very few limitations when compared to Buzz, to address resource constraints. The key contributions of this paper are:

- The development of a swarm-oriented runtime environment and language capable of defining the behaviour of heterogeneous swarms of robots with as small as 32 kB of flash and 2 kB of RAM;
- The adaptation and optimization of Buzz's capabilities (generality, mixed bottom-up/top-down logic, etc.) to resource-constrained platforms;
- The integration of BittyBuzz with three different robotic platforms used for research in swarm robotics.

BittyBuzz is released as open-source software and can be downloaded from our repository[1]. We evaluated BittyBuzz's performance through a series of experiments on the Bitcraze Crazyflie, the K-Team kilobot, and the Zooid robotic platforms. All these platforms are currently used in research on swarm robotics.

The rest of this paper is organized as follows. In section 2, we discuss some of the work related to BittyBuzz, while explaining the differences. Section 3 presents BittyBuzz's features and design principles while comparing them to Buzz's. Section 4 explains the design choices that were made to overcome the resource constraints. Section 5 presents an overview of the robotic platforms in which BittyBuzz was integrated. Sections 6 and 7 respectively present the performance evaluation and the experiments conducted. In section 8 we then draw the concluding remarks.

[1] `https://github.com/buzz-lang/BittyBuzz`

4.2 Related Work

A variety of previous works discussed the development of dedicated tools for swarm systems, as well as the creation of optimized frameworks for resource-constrained devices (see Table 4.1 for a summary). Micropython [38] is a lightweight and efficient implementation of the programming language Python 3. That implementation is optimized to run on microcontrollers and in constrained environments. It is composed of a full Python compiler, a runtime and a subset of the Python standard library, while able to fit a 256 KB code space and run with 16 kB RAM. Micropython has been proven to be a good option for rapid development of IoT devices with tested and verified libraries [39]. Currently, its main use targets are: development and testing of devices, sensor design, monitoring and configuration tools in design of complex applications, and education purposes [39]. For instance, [40] presents a prototype of an educational mobile robot based on MicroPython. Even by addressing the resource constraint issue, this language is very different from BittyBuzz. Apart from the difference in the memory usage, where BittyBuzz can be lighter, Micropython does not offer the numerous features (swarm primitives, neighbor management, etc.) that make BittyBuzz specific for swarm applications development.

Another development tool for microcontrollers is Artoo [41], a micro-framework designed for robotics. The framework provides a powerful DSL for robot control and physical computing. It works on Ruby and borrows some concepts and code from Sinatra [42]. Supporting 15 different platforms, among which the Bitcraze Crazyflie (one of our targets with BittyBuzz), Artoo allows developers to create solutions that incorporate multiple, different hardware devices at the same time. However, Artoo is not designed for swarm application design; it allows the user to connect to one or multiple robots to a centralized control computer, which is not suitable for the inherently distributed systems used in swarm robotics.

The Zephyr project [43] is also worth mentioning in that unites developers and users in building a small, scalable RTOS optimized for resource-constrained devices on multiple architectures. The goal of the project is to create an open, collaborative environment to deliver a RTOS that will answer the changing demand of the connected devices space. The system supports multiple boards and is easily portable across platforms. It has a different approach to the resource constraint problem when compared to BittyBuzz, simply offering a configurable RTOS that can be adapted depending on the user's needs.

Other existing works focus on the development of frameworks and/or DSLs for swarm applications. For instance, Koord [46] is a swarm-oriented language developed to make platform-independent code portable and verifiable. Koord proposes various useful features for coordi-

nation in a swarm, such as shared variables between the robots and state recording. However, Koord does not focus on resource constraints, which is the main contribution of this paper.

Another example is DRONA [47], a framework for building reliable distributed mobile robotics (DMR) applications. DRONA provides a state-machine based language (P) for event-driven programming. P is a high-level language that, once compiled, generates a C code that can be directly deployed in ROS. Protelis [48] is another language developed to provide a practical and universal platform for aggregate programming. Aggregate programming, unlike the traditional device-centric programming, is concerned with the behavior of a collection of devices - that can be assimilated to a swarm of robots. The language is hosted in and integrated with Java. Yi et al. [49] propose an actor-based programming framework for swarm robotic systems: they present a bottom-up programming approach based on a new control unit, the "actor". An actor is the virtualization of the capabilities of a given robot, and it allows developers to design cooperative tasks without dealing with the intricacies of robotic algorithms and specific robot brands. Similarly to Koord, these frameworks are not suitable to run on resource-constrained devices.

Peng et al. [44] present EmSBoT, a component-based framework targeting resource-constrained devices (using 13 kB of flash memory and 5 KB of RAM). It is built upon µCOS-III with real-time support. Even though it is network-focused, this framework supports the development of swarm robotics applications, although without the swarm-oriented features of BittyBuzz. OpenSwarm [45] is an OS designed for severely computationally constrained robots (using 1 kB RAM and 12 kB of flash). It enables the developer to design platform independent solutions, that can easily be applied to swarm robotics, as shown in the experiments. Both EmSBoT and OpenSwarm only provide low-level programming and work on a small subset of platforms, while Buzz and BittyBuzz are based on a virtual machine, meaning that the Buzz code is device-independent and can run on any system where the virtual machine is running without need of recompilation.

Overall, despite the fact that they are lightweight, these frameworks either do not focus on resource-constrained systems or do not meet the key requirements of a successful programming language for swarm robotics (decentralized control, spatial computing, neighbor communication, etc. [9]). As it has been mentioned earlier, Buzz [9] is a swarm-specific language that was developed to address the lack of software development tools in swarm robotics. Buzz offers a multitude of interesting features for swarm application development, including but not limited to: heterogeneous support, swarm-level abstractions, neighbor operations and a consensus system (the "virtual stigmergy"). The work presented in this paper is an adaptation of Buzz optimized to fit resource-constrained systems, such as those based on

microcontrollers. BittyBuzz supports the entirety of the Buzz language, with a runtime that can be adapted to different memory and computation constraints. We extended the support of BittyBuzz to three hardware platforms ([53], [5], and [6]) which are targeted towards resource-constrained robot swarms, but without a common framework or language. BittyBuzz provides a common language that allows the user to develop behaviours with minimal knowledge of the specifics of the platform it runs on.

Table 4.1 Comparison of existing frameworks for robots applications development.

System/Framework	Swarm support	Heterogeneous	DSL	Embedded	Support for resource constraints
MicroPython [38]		N/A	N/A	✓	✓
Artoo [41]		N/A	N/A		✓
Zephyr project [43]		N/A	N/A	✓	✓
DRONA [47]	✓		P	✓	
Protelis [48]	✓	✓	Protelis	✓	
Koord [46]	✓	✓	Koord	✓	
Actor-based framework [49]	✓	✓	(unnamed)	✓	
Emsbot [44]	✓	✓	N/A	✓	✓
OpenSwarm [45]	✓	✓	N/A	✓	✓
Buzz [9]	✓	✓	Buzz	✓	
BittyBuzz	✓	✓	Buzz	✓	✓

4.3 BittyBuzz Structure

In this section, we provide an overview of those features and design principles, while explaining the implementation difference in BittyBuzz, if any. We also give a reminder of Buzz's capabilities in swarm application development.

BittyBuzz virtual machine. BittyBuzz, just like Buzz, has a run-time platform based on a custom virtual machine written in C. The BBZVM has a variable size, and can be as small as 17.1 KB, and its structure is presented in Figure 4.1. The BittyBuzz virtual machine (BBZVM) structure and operation is the same as Buzz. Its structure is presented in Figure 4.1.

Swarm-level abstraction. A team of robots executing a common task is called a swarm. Buzz allows the developer to handle robot swarms as a first-class language object through the *swarm* primitive. This capability is conserved in BittyBuzz, and swarms can be easily created, tasks assigned, and robots can join and leave a swarm based on arbitrary runtime conditions:

```
# creation of a swarm with identifier 1
s = swarm.create(1)

# Join the swarm if the robot identifier (id) is even
s.select(id % 2 == 0)
# Join the swarm unconditionally
s.join()

# Assigning a task to a swarm
s.exec(function() { ... })

# Leave the swarm if the robot id is greater than 5
s.unselect(id > 5)
# Leave the swarm unconditionally
s.leave()
```

Swarms can also be manipulated via intersection, union or difference:

```
# a, b are swarms defined earlier in the script
i = swarm.intersection(100, a, b)
u = swarm.union(101, a, b)
d = swarm.difference(102, a, b)
```

Discrete swarm, step-wise execution. In Buzz and BittyBuzz, the swarm is considered as a discrete entity, with each robot running the BBZVM and executing the same script. The script execution on each device is done independently, the virtual machine operates in discrete time steps, each of which consists in a sequence of sub-steps: 1. the BBZVM collects

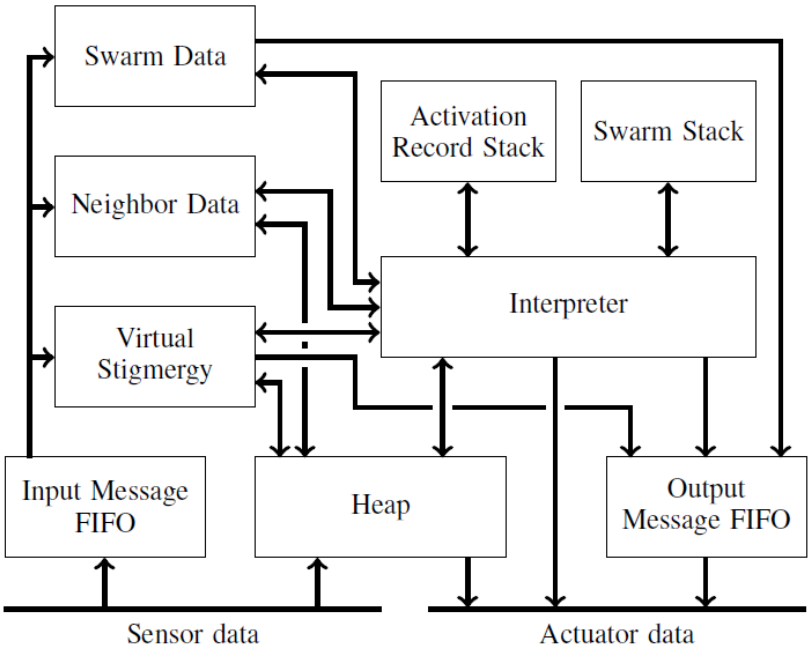

Figure 4.1 BittyBuzz virtual machine structure [9]

the robot sensors' readings, which would typically be stored in memory; 2. the BBZVM collects incoming messages and updates relevant data structures related to communications (e.g., the *neighbors* data structure) 3. the BittyBuzz interpreter is called to execute the Buzz script; 4. the BBZVM outputs outgoing messages; 5. and finally, the BBZVM outputs commands to the robot actuators.

Heterogeneous robots support. Buzz is an extension language and was explicitly designed to support heterogeneous robot swarms. Buzz and BittyBuzz alike allow the developer to extend the language by adding new robot-specific commands (see Section 4.3.1 for details).

Situated communication. All the robots running the BBZVM are assumed to have a device capable of broadcasting messages within a limited range and receiving them from robots in the vicinity. Situated communication is the ability to determine the relative position (i.e., the angle and distance) of the source of a received message, and it is one of the cornerstones of swarm robotics. While BittyBuzz does not require full situated communication, it supports its full or partial use (e.g., only estimating the distance of the source). The information pertaining to detected neighbors, including their distance and angle if available, is stored in a table inside the BBZVM and can be used for coordination, obstacle avoidance, etc. [9]

Virtual Stigmergy. Buzz implements the Virtual Stigmergy (VS), a conflict-free replicated

data structure that is used to provide consensus on a set of key-value pairs across the entire swarm [27]. The VS is essentially seen by the programmer as a shared table: each *put* operation by a member of the swarm triggers an automatic update in the others, with mechanisms to avoid conflicts. The VS is available in BittyBuzz with its complete features, and it is completely transparent to the user:

```
# Create a new virtual stigmergy
# A unique id (1 here) must be passed
v = stigmergy.create(1)
# Write a (key,value) entry into the structure
v.put("a", 6)
# Read a value from the structure
x = v.get("a")
```

Neighbor operations. Buzz and BittyBuzz provide a *neighbors* data structure that allows data collection and processing from neighboring robots. This type of neighbor operation is the cornerstone of spatial computing [54] and widely used in swarm robotics. The neighbors structure is a dictionary indexed by robot id and that contains the robot distance, the azimuth and the elevation angles of each neighboring robot, as detected by the BBZVM, and updated at each step through *situated communication*. It admits 3 spatial operations: iterate, map and reduce, with functions such as *map(), foreach(), reduce()* and *filter()*. The only difference with Buzz is that the functions *kin()* and *nonkin()* that allow the programmer to filter neighbors based on swarm membership are not implemented in BittyBuzz to reduce its memory footprint.

The *neighbors* data structure also allows direct communication with the robots in the neighborhood, by the means of a *broadcast* operation. The neighbors that want to receive messages on a given topic can subscribe to it with the *listen* function. A robot can also ignore the messages related to a topic by using the function *ignore*.

```
# Broadcast operation example
neighbors.broadcast("key", "value")
# Listen example
neighbors.listen("key",
   function(vid, value, rid) {
     print("Got (", vid, ",", value, ") from robot #", rid)
   })
# Stop listening to a key
neighbors.ignore("key")
```

These features of Buzz are fully implemented in the BBZVM, and allow a developer to focus on swarm behaviors, abstracting all the complexity of communication and neighbor

management. In fact, one can focus on the swarm behaviour as a whole with a top-down approach or use the robot-wise operations in a bottom-up fashion. The developer can even use both approaches if needed. Those are all useful and essential features for a swarm oriented programming language, and are all implemented in BittyBuzz.

Reducing the BBZVM's size to fit smaller microcontrollers needs some adaptations to the original Buzz virtual machine. Besides the two unimplemented functions mentioned above, we provide some parameters to modulate the BBZVM's size, and the memory consumption depending on the platform and application. The configurable attributes go from heap and stack size definition to feature deactivation/activation. The swarm structure as well as its operations can be disabled if needed. In this case, the swarm level abstraction will no longer be exploitable, and all the instructions will be robot-wise. The neighbor operations can also be disabled by the user, removing the situated communication. Similarly, the virtual stigmergy structure can be deactivated. BittyBuzz also has a memory usage reduction mode that allows the developer to drastically reduce RAM consumption at the cost of the flash. To reduce RAM consumption, that mode reimplements the *neighbors* structure, using a C structure based on a ring-buffer. In fact, creating a table for each neighboring robot would be memory-expensive, because a table segment would have to be allocated for each neighbor. Using a custom structure, allowed us to only allocate the 5 bytes of data necessary for each neighbor (robot ID, distance, azimuth, elevation). All these configurations are selected at compilation time by the user, depending on the robotic platform and the application.

4.3.1 BittyBuzz Extension

The ability to extend BittyBuzz (and Buzz) is a particularly useful feature when using the language for a new robotic platform. For instance, the *goto()* function's implementation and number of parameters is highly platform dependent and can be redefined for all the new robots by using external C closures. Defining such closures in BittyBuzz can be done in a very similar way to Buzz. In BittyBuzz, to use a new function in the Buzz script is as simple as writing a C function. The created function is then registered in the BBZVM with a call to *bbzvm_function_register(fnameid, funp)*. For instance:

```
### Usage in Buzz script ###
function step() {
    goTo(1.5, 2.0, 0.8)
}

### Implementation in C file ###
void bbz_goTo() {
```

```
    bbzvm_assert_lnum(3);
    float x = bbzheap_obj_at(
                bbzvm_locals_at(1))->f.value;
    float y = bbzheap_obj_at(
                bbzvm_locals_at(2))->f.value;
    float z = bbzheap_obj_at(
                bbzvm_locals_at(3))->f.value;
    # Robot specific code for movement
    ...
    bbzvm_ret0();
}

void setup() {
    bbzvm_function_register(
        BBZSTRING_ID(goTo), bbz_goTo);
}
```

4.4 BittyBuzz: Overcoming Resource Constraints

We used several "tricks" to make BittyBuzz efficient in terms of resource consumption and still be able to deal with dynamic memory management, which also represent one of the main contributions of this paper.

4.4.1 Dynamic Memory Management

BittyBuzz has a pre-allocated heap, the size of which is specified by the developer. The heap is represented by a static buffer with three sections: the object section (also containing the activation records of closure calls), the segment section and the unclaimed section. Two pointers are used to access the heap: the rightmost object pointer (ROP) and the leftmost table segment pointer (LSP) as shown in Figure 4.2. Each object has two bytes containing the payload information and an additional metadata byte.

Regarding the storage in the heap, non-structured types such as nil, int, float and string are written from left to right with ROP always pointing to the last added object. For structured types (tables), an object is stored in the objects' section, referring to a data segment stored from right to left in the segments section. Each data segment has a user-defined number of (key, value) pairs and two metadata bytes containing information about the segment validity and a pointer to the next segment, if any. Although the (key, value) pair is replaced by a single array with a user-defined size in the case where the keys are integers, in a way that

resembles the table management in Lua [55].

BittyBuzz has a simple and effective garbage collector algorithm which is frequently called during the script execution, specifically before the execution of instructions. Each entry in the BittyBuzz heap has an attribute containing its metadata, including some garbage collector bits. The algorithm starts by unmarking all the segments and objects in the heap, i.e., setting their garbage collector bits to zero. Then, it marks back the permanent objects that should never be garbage collected, and the valid variables on the BittyBuzz stack. Finally, all the remaining unmarked objects are invalidated, and the heap pointers are re-positioned.

4.4.2 Optimizations and Limitations

Ring-buffers

To reduce the time complexity for message queue management, BittyBuzz implements its own ring-buffer. Ring-buffers are used in several algorithms and are desirable in data stream use cases. They are buffers that are implemented to seamlessly loop on themselves, allowing constant time *push* and *pop* operations for queues. Previous works such as [56] propose a new implementation for multi-thread modifications of the ring-buffer: we use a similar but simpler implementation running in a single thread. The implementation uses a ring buffer, with an element size (the size of an individual element in the buffer), a buffer capacity, a start index and an end index. With this information, insert, pop, and index access are done in constant time. We use ring-buffers in BittyBuzz for storing the message queues, the payload buffer used for communications, and the *neighbors* data structure (this latter only when the BBZVM is in memory usage reduction mode).

Translated bytecode

The available flash memory is also an important factor in the resource consumption management. To reduce the storage space, the original Buzz object (.bo file) is converted into a smaller BittyBuzz object (.bbo file). The optimizations mainly include the conversion from Buzz integers on 32 bits to BittyBuzz integers, which are on 16 bits. Floats are also converted to *bbzfloat* that only use 16 bits. All the non-instruction related strings are also stripped from the new object file to minimize file size.

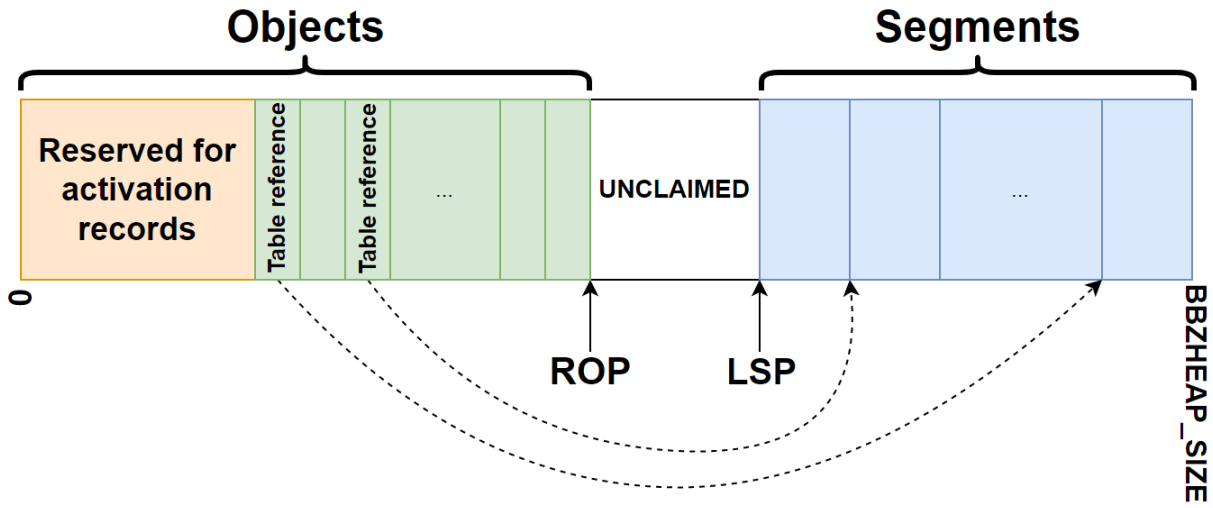

Figure 4.2 Illustration of BittyBuzz's heap: the object section containing the activation records section and the allocated objects (non-structured types and table references) in light green, the unclaimed section in white, the segment section in light blue, and the two pointers ROP and LSP.

Optimized loops

As discussed earlier, BittyBuzz uses a ring buffer to store neighbors' information when the BBZVM is in reduced memory usage mode. In fact, instead of using table segments on the heap, the reduced memory mode uses a static table of user-defined size for the neighbors' data. We optimize the looping through this data structure with a *foreach* operation: for each neighbor in the ring buffer, a data table is created to execute the needed instructions and then garbage collected at the end of the iteration. The loop therefore uses a constant space on the heap instead of a linearly increasing one.

Swarm lists

In BittyBuzz, a robot can be a member of a maximum of 8 swarms, numbered from 0 to 7. To keep track of which swarms a robot is a member of, we use a swarm list that is simply an 8-bit variable where the i-th bit represents whether a robot is a member of the i-th swarm (for example, if the bit 0 is 1 that means the robot is a member of the swarm 0). This list allows one to easily determine all the swarms of a given robot. When a robot is added to a swarm, BittyBuzz foresees to add the robot's swarm list to the outgoing message queue so that its neighbors can have the information and add it to their swarm table. This is a

significant departure from the original Buzz which used a stack of arbitrary size to determine swarm membership, but up to 8 swarms the difference is completely transparent to the user.

Virtual stigmergy

As presented earlier, BittyBuzz, just like Buzz, has a virtual stigmergy system, although BittyBuzz's has some limitations with respect to the original version. For instance, BittyBuzz only has a unique instance of the stimergy, whereas one could create an arbitrary number of stigmergies in Buzz. In addition, the current implementation only takes strings as topics (keys) for the virtual stigmergy.

String manager

Strings are a data type used in many programming languages to represent a chain of characters. In BittyBuzz, in addition to variables used in the Buzz script, strings are used to identify global symbols and function names. In conventional implementations of strings, the data type is represented as an array of bytes (or words) [57]. However, such an implementation is memory consuming and needs an optimization to be effectively used in a resource constrained system. BittyBuzz represents strings as integers, by assigning a unique ID to each of the strings used in a given script, while the string content is placed in the flash RAM. This implementation reduces the memory footprint for strings as they always occupy 2 bytes instead of an array of unknown bytes. Some string IDs are necessary for BittyBuzz to operate correctly, and they are natively declared in the VM (for example the *stigmergy* keyword). As needed, the developer can create new string IDs for user-defined closures, using the macro *BBZSTRING_ID*.

Neighbors communication

As mentioned earlier, BittyBuzz provides a publish-subscribe like mechanism, to allow robots to communicate. To send a message, a program calls the function ***neighbors****.broadcast(topic, value)* . Because of the extremely limited payload size of the targeted robotic platforms, BittyBuzz does not support sending tables as value. Also, since strings in BittyBuzz are represented by unique IDs (see 4.4.2) created during the execution, the code on the robots must be the same if the developer is trying to implement a communication-based behaviour. More specifically, the order and the number of strings literals must be the same. This final constraint is not limiting, as using the same script for all robots is the standard way Buzz scripts are operating.

4.5 Robotic Platforms

We ported the BBZVM to three robotic platforms used in different swarm robotics projects, shown in Figure 4.3.

4.5.1 Kilobot

The Kilobot is an open-source low-cost small robot (33 mm diameter and 34 mm height) designed to ease the testing of algorithms for a large swarm (hundreds or thousands of robots), which can normally be time and money consuming. Each Kilobot can be made with $14 worth of parts and can be assembled in a relatively short time [5].

Kilobots have the basic capabilities of an autonomous swarm. Those include a programmable controller, basic locomotion and local communication [58]. The Kilobot uses an ATmega 328p processor (8 bit@8MHz), a 32 kB flash, 2 kB SRAM and a 1 kB EEPROM and a rechargeable battery, and they are programmed in C [59].

4.5.2 Bitcraze Crazyflie

The Crazyflie is an open-source experimental flying development platform used for research and education in robotics. This relatively low-cost, easily expandable and upgradeable quadrotor is 92x92x29mm and weighs 27g. It can easily be assembled without any soldering and supports several expansion decks. As presented in [7], the Crazyfile offers interesting features for research on UAVs and for swarms of UAVs. Crazyflies are equipped with a STM32F405 MCU Cortex-M4 @ 168MHz, a 192 kB SRAM and a 1 MB flash for the main application. They also have a nRF51822 radio and power management MCU Cortex-M0 @ 32MHz with 16 kB SRAM and 128 kB flash. Finally, they have an 8 kB EEPROM. [60].

4.5.3 Zooids

Zooids [6] are small open-source and open-hardware robots, designed for a new class of human-computer interfaces for tabletop swarm interfaces. They each weigh 12 g for 26 mm in diameter and 21 mm in height. They contain a 48 MHz ARM microcontroller (STM32F051C8) equipped with a 64 kB flash and 8 kB SRAM.

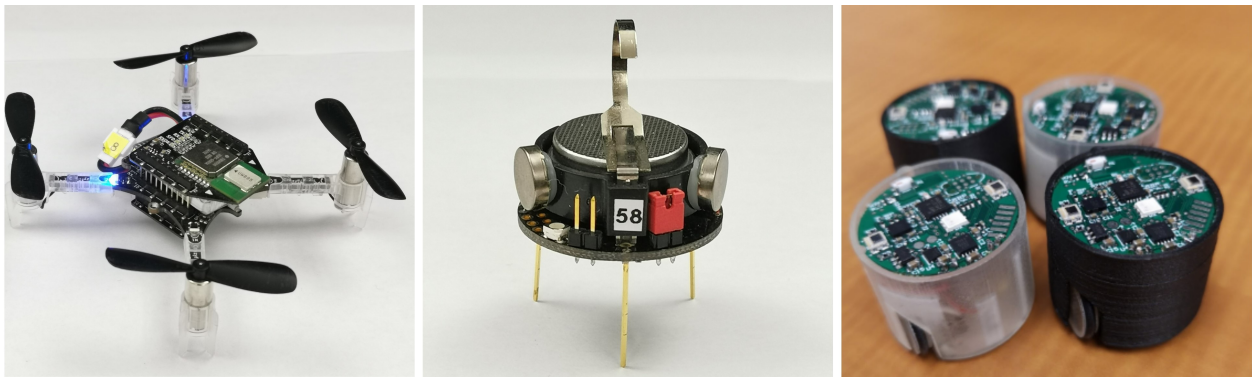

Figure 4.3 Robotic platforms integrated with BittyBuzz. From left to right: Bitcraze Crazyflie, K-Team Kilobot, and Zooids

4.6 Performance Evaluation

We analyze the performance of the BBZVM in terms of memory use, communication capability, and functionality.

Bittybuzz has a variable memory footprint depending on the activated features. Figure 4.4 summarizes that footprint for the three robotic platforms. For instance, when using the Crazyflie, the default configuration activates all the features, with a *128 bytes* stack size and a *3500 bytes* heap size. The Crazyflie's flash is occupied by the firmware, that takes 240.6 kB (24.1%), and the default configuration of the BBZVM occupying 19.3 kB (1.93%). That leaves 740.0 KB of flash available for the bytecode describing the robot behaviors, which is generally very compact. To give a better idea of the memory occupied by a user defined behavior, the memory used by the experiments presented in this paper is shown in Table 4.2. In particular, the memory usage of an exploration behavior combined with obstacle avoidance using most of the programming language features (swarm-level abstraction, situated communication, and virtual stigmergy) was just 2.48 kB (0.33 % of the remaining available space).

As mentioned, BittyBuzz uses a discrete time step, but we do not enforce a specific predefined time step duration. The duration of the time step can be set depending on the requirements of application and the characteristics of the hardware. To get an idea of the possible time step values, Table 4.3 presents the mean, minimal and maximal time step for each experiment (described in the following), when running BittyBuzz without timing constraints on the Crazyfile. The jitter shown is due to the internal behavior of the Crazyfile FreeRTOS system. Fundamentally, the results show that the BittyBuzz virtual machine can run at more than the required speed for implementing realistic swarm behaviors.

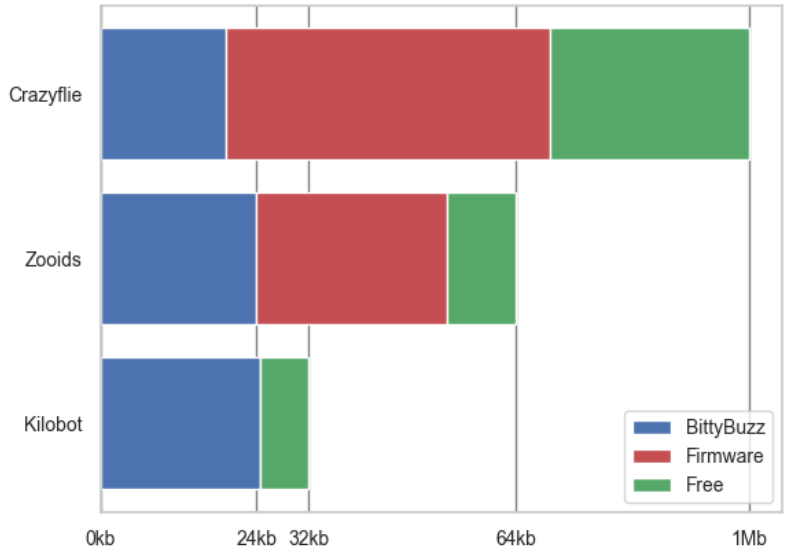

Figure 4.4 Sizes of the BBZVM, the robot native firmware, and the available space for a bytecode for the three integrated robotic platforms: Bitcraze Crazyflie, K-Team Kilobot and Zooids

Table 4.2 Memory footprint of three BittyBuzz scripts

Experiment	Memory (KB)
Exploration	2.48
VS	0.732
Neighbors broadcast	0.396

Table 4.3 Min, mean, and max time step in ms during the experiment execution with the confidence interval

Time step (ms)	Exploration	VS	Neighbor queries
Min	12	1	2
Mean	68.74 ± 9.66	3.83 ± 1.08	5.92 ± 0.56
Max	110	13	9

4.6.1 Neighbor Queries

One of the potential issues of gossip-based communication between neighbors is the flooding of the network, which is exacerbated by the limited capabilities of resource-constrained hardware. We show the validity of the BittyBuzz communication model by computing the packet loss during broadcast operations on the BitCraze Crazyflie, which uses Bluetooth Low Energy (BLE) as its communication protocol. We test BittyBuzz neighbor queries with 20 tests, each sending 200 broadcasts (meaning a precision of 0.5% for the packet loss measurement). We also vary the broadcast rate (i.e. the load): 500 ms (2 Hz), 100 ms (10 Hz) and 50ms (20 Hz).

The results show packet loss between 0 and 4.5% in all cases. The mean value is $1.9\% \pm 1.2\%$, $1.55\% \pm 0.97\%$, and no packet loss for the 50ms, 100ms, and 500ms periods, respectively. Figure 4.5 presents a summary view of these results, which show that, as can be expected, as the frequency increases, the packet loss increases as well, but remains well within acceptable levels for typical swarm robotics control cycles. It is worth noting that most swarm robotics applications have control cycles between 10 Hz and 50 Hz [61]. As a general rule, the user should set the step period as a compromise between the speed of execution and the reliability of the network.

4.6.2 Consensus

We evaluate the implementation of the Virtual Stigmergy (VS) [27], a conflict-free replicated data structure used to reach a consensus on key-value pairs in the whole swarm. We place a set of 10 Crazyflies running BittyBuzz on a flat surface, 0.5 meter away from each other. One Crazyfile of them acts as a writer and puts a value in the VS, while the others act as readers of the same value. The goal of this experiment is to determine the time necessary for all drones to share the same value, i.e. to reach consensus. We used two configurations, as shown in Figure 4.6: a chain structure where the writer is placed at the extremity, and a circle-like formation where all drones are at the same distance from the writer. The tests were performed 20 times while alternating between the 2 configurations. Varying the configuration allowed us to evaluate the performance of the VS based consensus in different use cases.

As shown in Figure 4.7, the mean consensus time is 56.8 ± 23.26 ms, with a global envelope between 17 and 87 ms. This means that each drone gets the value either from the first broadcast by the writer, or after a response from its own request for a value using a *get* operation, meaning that consensus is reached in either one or two control steps. The period of 50 ms used in this experiment explains the bimodal distribution shown in Figure 4.7, with

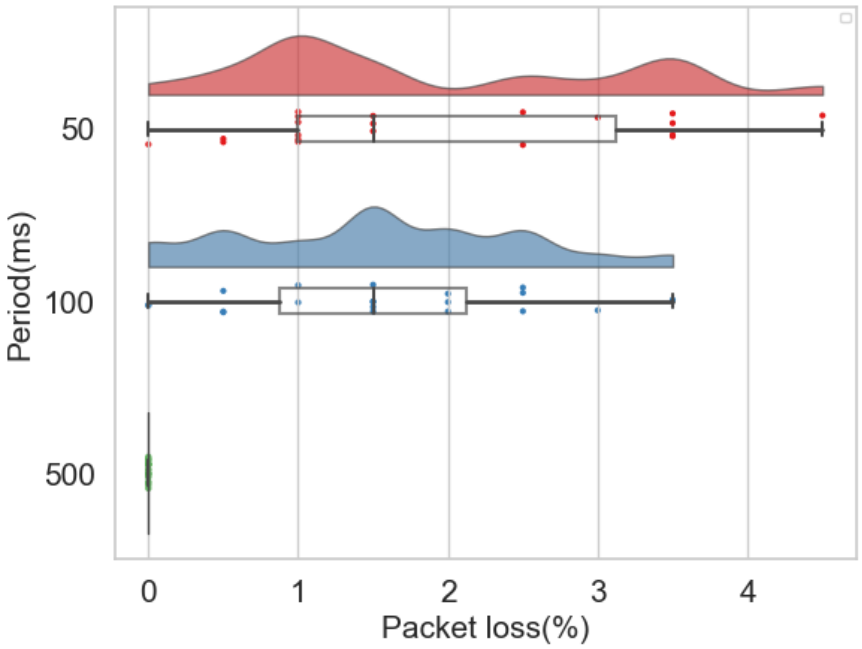

Figure 4.5 Packet loss during neighbor queries under different load. From top to bottom: 50 ms, 100 ms and 500ms

the large gap in the middle suggesting that the period could be further reduced without negative effects.

4.7 Behavior Experiments

To prove BittyBuzz's applicability for swarm application development, we implemented two well-known swarm behaviour, and evaluated their performance.

4.7.1 Bidding

We implemented a distributed bidding algorithm [62] commonly used for task allocation in swarm applications, and deployed it to a set of Crazyflies. Tasks are dynamically generated and provided to the swarm via Virtual Stigmergy. When a new task appears, each robot broadcasts a bid and listen for the bids of the other robots. Using a barrier mechanism [61], the robots wait until all the swarm has cast their bid. Given that all the bids are shared and known by the entire swarm, the winner of the bid can be determined locally without a centralized auctioneer. We run 20 experiments for each of 3 configurations, using an increasing number of robots: 3, 4 and 6 Crazyflies. As for the other experiments, the step period is 50 ms.

Figure 4.6 Configurations used for consensus tests. Left: chain like configuration; right: circle-like configuration

In addition to proving the usability of BittyBuzz, these experiments also validate the combined use of its main features: step-wise execution, Virtual Stigmergy and neighbor queries. The experiments show that there is sufficient slack in the step period to have decentralized task allocation, which is a generic, reusable behaviour for swarm applications.

The resulting average duration of the auction for one task is presented in Figure 4.8: 88.00 ± 29.25 ms, 99.05 ± 35.30 ms, 137.10 ± 21.70 ms for 3, 4, and 6 drones respectively. As expected, the auction duration increases with the number of drones, as more robots mean a higher number of messages exchanged for the auction. In particular, the robots need to wait for the value of their neighbors before they can compute the winner, adding to the overall consensus time. Globally, fully decentralized task allocation is relatively fast using BittyBuzz, and show that it can be used in practice.

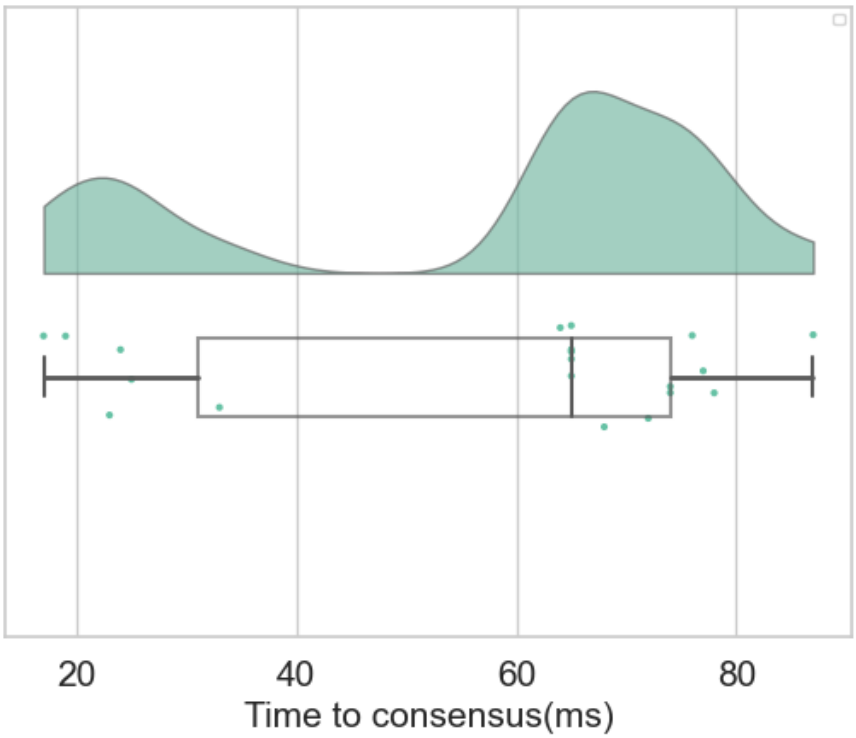

Figure 4.7 Distribution of the required time to obtain consensus when using virtual stigmergy (VS). 20 tests were run using different configurations for the robots

4.7.2 Exploration

We finally implemented an algorithm for the exploration of unknown environments using a random walk [63]. We use 2 Crazyflie quadrotors in a laboratory setting, each of which randomly samples a direction for the exploration and updates the direction periodically. The drones were also equipped with proximity sensors (multi-ranger and flow deck) to detect and avoid obstacles during the exploration. Figure 4.9 shows the trajectories of the two drones during one of the multiple runs for the exploration.

4.8 Conclusions

We presented BittyBuzz, a novel virtual machine for the Buzz language designed for resource-constrained microcontrollers.

The contributions of this work include: 1. the development of a virtual machine for a dynamically typed language with dynamic memory management with as little as 32 kB of flash and 2 kB of RAM; 2. the full inclusion of Buzz's programming model (including neighbor

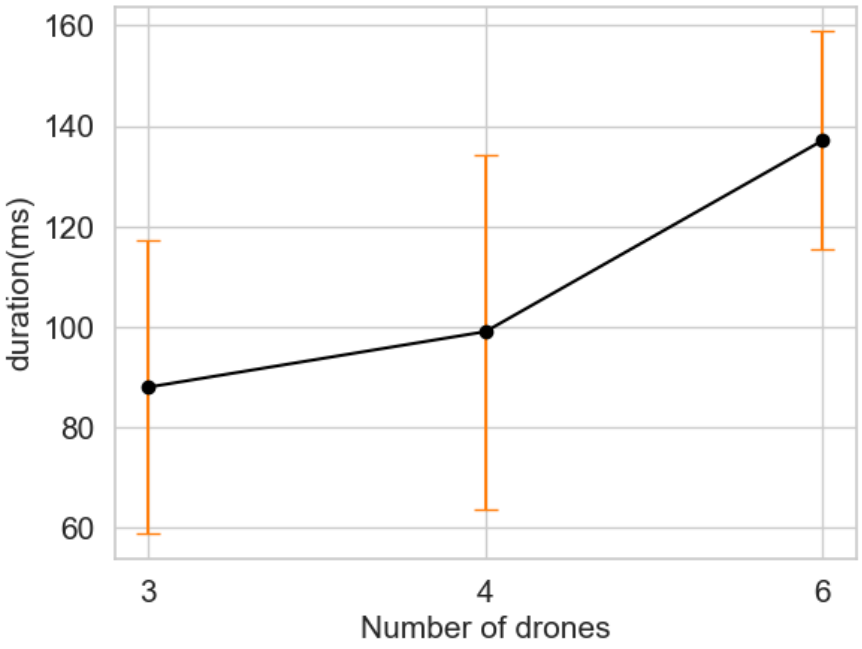

Figure 4.8 Auction duration with different number of drones

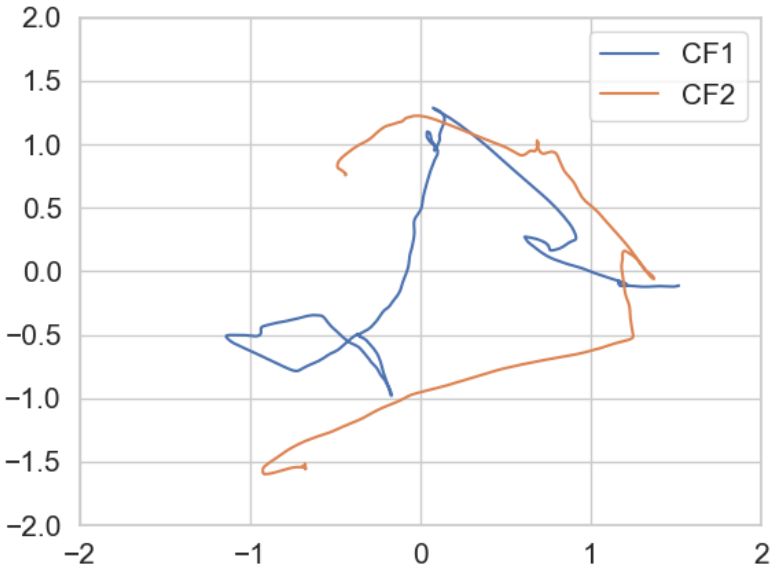

Figure 4.9 Trajectories of the two drones used for the random walk based exploration

queries, consensus, etc.) to a resource-constrained platform; 3. the integration of BittyBuzz with three different robotic platforms used for research in swarm robotics. In addition to these contributions, we believe that resource-constrained robotic platforms have an impor-

tant part to play in the future of the internet of everything. A domain-specific dynamic language such as BittyBuzz will ease the development and prototyping of swarm applications. Our experiments show that BittyBuzz can be used for established swarm behaviors and that the communication model used by the virtual machine has sufficient performance even on severely constrained devices.

Acknowledgments

This work was supported by the Fonds de recherche du Québec – Nature et technologies (FRQNT) under grant 296737 and by the National Research Council Canada (NRC).

CHAPTER 5 ARTICLE 2 : SEARCH AND RESCUE WITH SPARSELY CONNECTED SWARMS

Preface: Research in the field of swarm robotics has been focusing on algorithmic innovations, somehow, neglecting the real-world applications. However, to access the innumerable benefits that swarm robotics can offer, real-world applications should be the center of attention. Aware of that need for real-world deployments, this chapter proposes a solution to connectivity issues in sparse swarms during search and rescue applications. It also implements a search method based on a distributed belief map for a fast detection of the targets.

In addition to the proposed search pattern and algorithm, this chapter provides a reusable experimental robotic system for real-world deployment of swarms of UAVs.

Full Citation: Dah-Achinanon U, Bajestani SEM, Lajoie PY, Beltrame G, "Search and rescue with sparsely connected swarms", Autonomous Robots, [under review] 2021.

Abstract - Designing and deploying autonomous swarms capable of performing collective tasks in real-world is extremely challenging. One drawback of getting out of the lab is that realistic tasks involve long distances with limited numbers of robots, leading to sparse and intermittent connectivity. As an example, search and rescue (SAR) requires robots to coordinate in their search, and relay the information of found targets. The search's effectiveness is greatly reduced if robots must stay close to maintain connectivity. This paper proposes a decentralized search system that only requires sporadic connectivity and allows information diffusion through the swarm whenever possible. Our robots share and update a distributed belief map, to coordinate the search. Once a target is detected, the robots form a communication relay between a base station and the target's position. We show the applicability of our system both in simulation and with real-world experiments with a small swarm of drones.

5.1 Introduction

Drones or unmanned aerial vehicules (UAVs) have been experiencing steady growth for the past few years in terms of their popularity, availability, and potential. This enables a wide range of applications in several fields going from surveillance to search and rescue missions [64]. The emergence of swarms systems, made of multiple UAVs collaborating towards a common goal, further improves the efficiency of those solutions. Swarm robotics has been

Figure 5.1 The real-world experiments setup with three DJI M300 quad-copters

defined by [1], as an approach to collective robotics inspired by the self-organized behaviors of social animals, where a large group of simple robots aims to accomplish a complex task through simple rules and local interactions.

Using teams of robots, instead of a single one, brings robustness, scalability, survivability, and it increases the speed of execution [29]. Despite those practical benefits, many unresolved challenges prevent swarm robotics from being used in commercial products. In particular, swarms of UAVs are confronted with multiple challenges such as: in-flight coordination, swarm layout reconfiguration, handling losses of swarms elements, and data relaying optimization among others [65]. [28] outline one of those challenges and formalizes the notion of sparse swarms in which it can be prohibitively expensive for the robots to maintain close proximity. For example, during swarm-based search and rescue (SAR) operations, preserving close proximity among the robots would certainly restrain the area that can be covered. Therefore, a robust ad-hoc communication system, resilient to disconnections between the swarm agents, is essential to deploy such systems in realistic scenarios.

In this article, we attempt to bridge the gap between theoretical approaches and practical applications by proposing a SAR algorithm based on ad-hoc networks accepting sporadic connectivity. This algorithm leverages a search pattern accepting sparsely connected swarms by scheduling a rendezvous point for periodic meetings and target discovery report. It then allows the swarm to adopt a relay tree formation connecting the meeting point (or base)

to all the detected targets. Thus, the connectivity between the ground operators and the robots is restored only when targets are found. The search method is based on belief space exploration in order to incorporate crucial priors from the authorities, such as the last known locations of the targets.

This paper argues that ad-hoc networks accepting sporadic connectivity are the key to real-world deployments of swarms of UAVs in large areas. Such networks should handle adaptive topology and routing while assuring reliable data exchange within the swarm. Existing works exploit either a rendezvous point or a relay chain formation to improve the communication links, but, to the best of our knowledge, none combine these for SAR applications. For instance, [66], [17], and [20] leverage a rendezvous point for exploration and SAR, but they do not use communication relays. Other works present relay chain architectures for communication enhancement [32,33], without considering the search pattern. This article aims to bridge these two principles with the mentioned algorithm. To summarize, the main contributions presented in this paper are:

- A search pattern based on rendezvous point, using ad-hoc networks with adaptive topology and routing, and accepting sparsely connected swarms;
- An algorithm to create an adaptive relay tree structure design to maintain the communication between the ground operators and the discovered targets;
- An implementation of a search method based on dynamic, distributed belief maps;
- An experimental robotic system for real-world deployment of swarms of drones;
- Simulation and real-world deployments of the proposed system.

We validate our approach through tests in simulation and real-world experiments. In simulation, we test our dynamic belief map search algorithm on different area sizes and number of drones. We also performed real-world field tests with three drones to confirm our findings. Figure 5.1 shows the real-world experiments setup with three DJI M300 quad-copters.

The rest of the paper is organized as follows: in section 5.2 we present related works outlining similarities and differences with our approach. In Section 5.3 we describe our algorithm and its components. Sections 5.4 and 5.5 present our simulations and experiments setup and results. Finally, Section 5.6 draws concluding remarks while presenting possible future works.

5.2 Related Work

As mentioned by [29], a reliable communication structure is essential to share information among a group of neighbors in swarm applications. Sharing the information becomes essential in applications, such as SAR, where a group of robots is trying to find a target in a large area. As indicated by [15], finding the target could be faster using more robots. However, the lack of reliable communication could separate a group of robots or make the whole mission fail, especially in centralized applications. For instance, [15] proposed a layered search and rescue (LSAR) centralized "partitioning" algorithm, that needs reliable communication with a cloud server. Although their simulation results indicate that a better success rate could be achieved by increasing the number of robots, it is inherently limited by the central communication bottleneck to the server. Recent work [16] introduced an optimized self-organized mesh network to cover large areas, but do not consider disconnections.

To solve these communication issues in a team of robots during exploration missions, [17] considered a periodic rendezvous strategy in order to overlap the communication ranges of the robots. It also presented an approach to mitigate the negative impact of these meetings on the time efficiency of the overall mission. Our approach is similar, but we drop the connectivity maintenance requirement during the searching phase. Another benefit of our technique is that the drones continue to search for the targets while going to the periodic meetings. [18] and [19] also used a meeting point, but the robots only meet at the rendezvous when the exploration is completed. Adopting a different approach, [20] plans the exploration in order to converge to a predefined spatial configuration around the rendezvous point.

Belief maps have been studied for a long time as a tool for multi-robot exploration [22, 23]. Similar to our work, [25] updates the belief map with local observations and merges data from multiple UAVs. Our distributed belief map implementation is an adaptation of the work proposed by [26], in which the authors stored the shared belief map in a distributed database called virtual stigmergy [27].

During the rescue phase of SAR missions, it is crucial to maintain the connection between the base station and the UAVs following the target. To this end, we propose to maintain a relay chain from the rendezvous position to the targets once these latter are found. Using a heuristic optimization method, [31] increased the communication performance metric and determined the optimal positions for the communication relay robots. To keep the connection between a heterogeneous group of robots (on-ground and flying robots), [32] introduced a fully decentralized algorithm to create and keep a chain of robots from the ground station to the target. [33] and [34] also used drones as relays and virtual potentials to create a stable

link between a group of robots (or a survey drone) and a base station. Instead of using virtual potentials, [35] measured the communication quality and expanded the drone relays when needed. While the above-mentioned works focused on the formation of relay chains during the search phase, this could make the search process very slow, which would be critical for SAR operations. Therefore, we propose to create the communication relay chains only in the rescue phase. Our communication relay approach is similar to the approach proposed by [36, 37], in which the creation and expansion of tree/chain topologies between drones and target(s) have been evaluated. We use a tree topology for the relay connection when rescuing more than one target.

5.3 Search and Rescue with Sparsely Connected Swarms

In this paper, we consider the scenario in which a swarm of drones needs to be deployed in an unknown environment to search for one or more targets, and track them as rescuers are dispatched to the target locations. The drones explore the area autonomously and in a decentralized manner, searching for targets. Communication links are needed between the swarm members either to inform the others when a target is found and to share the target positions, propagating this information to a base station so that the targets can be rescued.

In realistic scenarios, the search area is likely to be larger than the combined communication coverage of the robots in the swarm. Therefore, the searching robots need to disconnect from their neighbors to explore enough space to find the desired targets, creating a sparse swarm.

Let us consider a swarm $\boldsymbol{S}$ of n robots $\boldsymbol{S} = \{1, 2, ..., n\}$. At a given time step $t_s \geq 0$ during the mission, $\boldsymbol{S}$ is considered a sparse swarm if a robot $r \in \boldsymbol{S}$ satisfies:

$$\begin{aligned} cost_r(\text{"}move \quad to \quad nearest \quad neighbor\text{"}, t_s) \gg \\ cost(\text{"}perform \quad typical \quad operation\text{"}, t_s) \end{aligned} \tag{5.1}$$

where $\gg$ is defined as "at least one order of magnitude greater than" and $cost_r$ is a function defining the cost for robot r to perform a given task at a given time [28]. In order to ensure coordination and efficient searching in such a swarm, we designed an algorithm inspired from typical search parties in rescue operations, shown in Figure 5.2.

The overall idea is that robots perform their search for a target, regularly reporting at a fixed rendezvous location or meeting point. If a robot finds a target, it immediately goes to the meeting point to share the location of the target with the operator and the rest of the swarm. Assuming a robot is the first to go to the meeting point after finding a target, it becomes a *root* robot, and it coordinates the formation of a relay chain towards the target. The chain

construction starts as the root robot broadcasts a call for *networkers* (i.e., relay robots) that other members of the swarm respond to with bids based on their distance from the required position of the relay, and the root robot assigns roles based on the received bids [67]. As robots find more targets, the root adds branches to the relay formation, and should robots find targets simultaneously, they elect a root through a basic consensus mechanism.

5.3.1 Agent Roles

The overall strategy is based on a state machine that assigns roles to the robots in the swarm, with each role is associated with a task executed by the robot.

Searcher: when a robot is a searcher, it looks for targets using a predefined search method (a belief-based search in our case). During its operation, a searcher listens and responds to *calls for bids* from other agents. These calls offer networker (i.e., relay) roles, and a searcher bids based on its distance to the requested relay position.

Root candidate: upon finding a new target, an agent will go to the rendezvous point to become a root robot. In case of a root being already present, the agent shares its target information with the existing root and go back to being a searcher. If multiple robots are heading to the rendezvous point, the first robot to arrive proclaims itself the winner and shares that information with every incoming robots. Should multiple agents arrive to the rendezvous at the same time, we use a conflict management method based on robot ID [27].

Root: winning the bid for the root node, an agent becomes the root: it listens for new target information from the swarm, it computes the number of networkers needed per target and their positions, and calls for robots to fill the networker roles. Note that this strategy does not make the system centralized: the root is easily replaced in case of failure with a new election.

Auctioneer: when the root needs networkers to cover a target, it switches to the auctioneer state, for a typical market-based task allocation strategy [67]. The root/auctioneer broadcasts the relay position, opens the auction, listens for bids, and closes the auction after a predefined period of time, remaining in the same state until a winner is found. The auctioneer then broadcasts the winning bid and goes back to the root state.

Networker bidder: when a searcher receives a call for bids, it stops moving and bids for a networker role. Its bid value is inversely proportional to its distance to the assigned relay position.

Networker: the networker bidder that is the closest to a relay position wins the bid and becomes a networker. A network relays information between a target location and a base station at the meeting point, connecting the target with the operator and providing constant communication coverage in the area of the target.

Rendezvous: robots regularly switch to the rendezvous state and go back to the meeting point to check for any new information (new root node, request for a networker, or updated found targets list). Note that robots keep searching for targets on their way back to the meeting point. If there are no networker calls for bids happening, the robots in rendezvous state go directly back to being searchers.

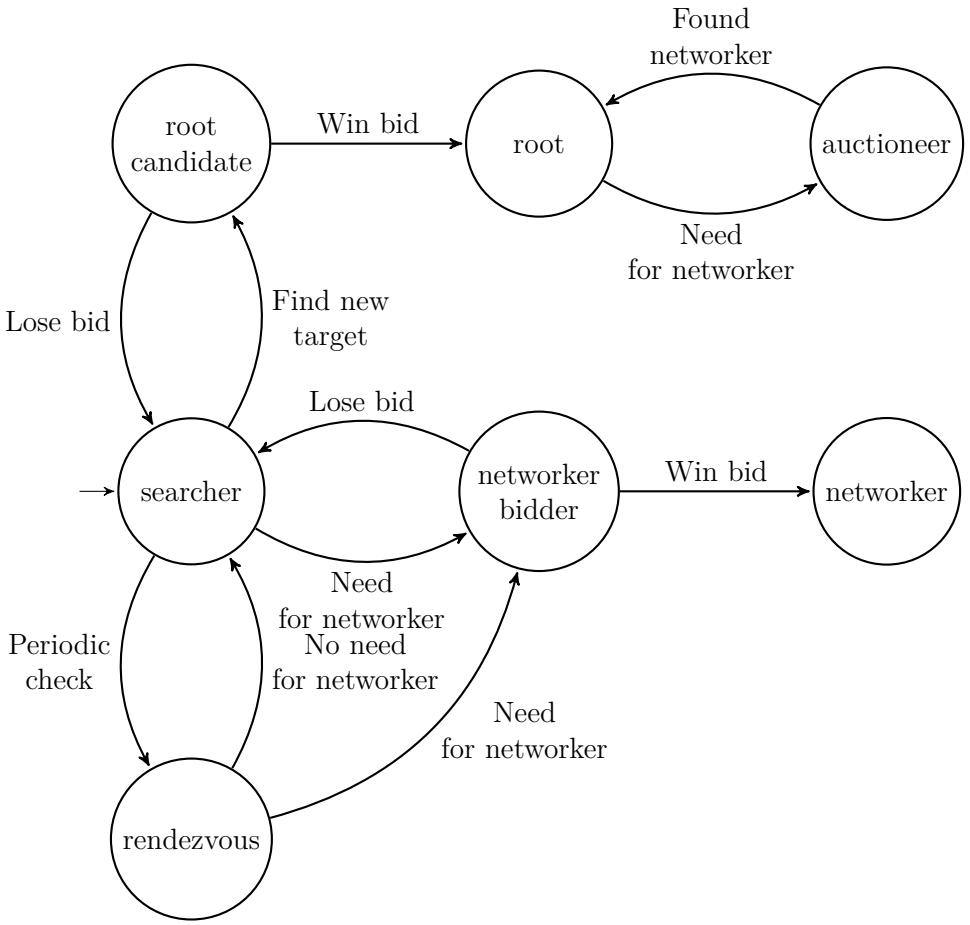

Figure 5.2 Coordination algorithm for target searching

5.3.2 Algorithm

All the robots execute their search for the targets based on the available belief information. The belief information, represented as a map, is updated and distributed to the neighbors

during the search to avoid searching the same area multiple times. This distributed belief map based search is inspired from the work in [26]. To distribute a belief map, we use the virtual stigmergy (VS), a system that allows a swarm of robots to agree on a set of (key,value) pairs through gossip communication [27]. Thus, at each step, a searcher will decrease the belief value for its current position if no target is detected there. The new value is then put into the VS, sharing the information with all neighbors in communication range. A searcher sampling a new position to navigate to, will then opportunistically get the most recent belief for the position (from the initial map if no updated version is available in the VS). It is worth noting that the propagation of the VS is strictly best-effort, and therefore tolerant to disconnections and communication delays. Based on the value of the belief map, the robots decide to either move to the sampled position or sample a new one (if the belief is below a certain threshold).

The idea of the search pattern is to realize multiple runs of a user-defined duration and come back to a rendezvous point between each of them. During the search, if a robot finds a target, it goes back to the initially fixed rally point and checks for the existence of a root node. The first drone to come back to the rally point after finding a target will be the root. When multiple drones arrive to the rendezvous at the same time, a conflict management routine selects one of the drones as the root. This robot becomes the first link of the communication and tracking relay between the meeting point and the targets. The root stays at the rendezvous point and broadcasts relevant information (updated found targets list, root id, networking positions, etc.) to all the robots in its communication range. As explained previously, this node computes the networkers' positions and manages an auction every time it needs a new networker by switching to the auctioneer state. The networkers's positions depend on the *communication range* of the robots. Let N be the set of networkers positions in the system and $\boldsymbol{r}$ the root's position. To find the networkers positions for a target located at $\boldsymbol{t}$, we choose a branching node at a position $\boldsymbol{b}$ such that:

$$\|\boldsymbol{t} - \boldsymbol{b}\| = min(\|\boldsymbol{t} - \boldsymbol{n}\|) \tag{5.2}$$

for all $\boldsymbol{n} \in N \bigcup r$.

The networkers are then placed one after the other on the line connecting $\boldsymbol{b}$ and $\boldsymbol{t}$. They are spaced at a maximum distance of *communication range* to ensure connectivity in the relay.

Other robots that did not find any target, go back periodically to the meeting point to check if another robot found a target or if a networker is needed. Since the battery life of flying robots is quite limited, this periodic check is an opportunity for recharging or battery swapping. When reaching the meeting point, if the robot receives a message from the root

for a networking position, it immediately sends its bid for the auction and waits for the results announcement. When a robot wins the bid it acknowledges the auctioneer that it received the message and goes to its assigned position, becoming part of the communication and tracking relay. The robots in relay positions form a tree from the meeting point, allowing the operator at a base station to constantly and simultaneously to "see" and monitor the state of all the detected targets.

With a sufficient number of searcher robots, the relay should allow connectivity maintenance and a live camera stream of the target to the ground operators. In the case of a moving target, the closest robot to target has the responsibility of tracking its motion and sharing the updated position along the relay. This way, the relay can adapt itself and follow the target up until its rescue.

5.3.3 Real-world Deployment

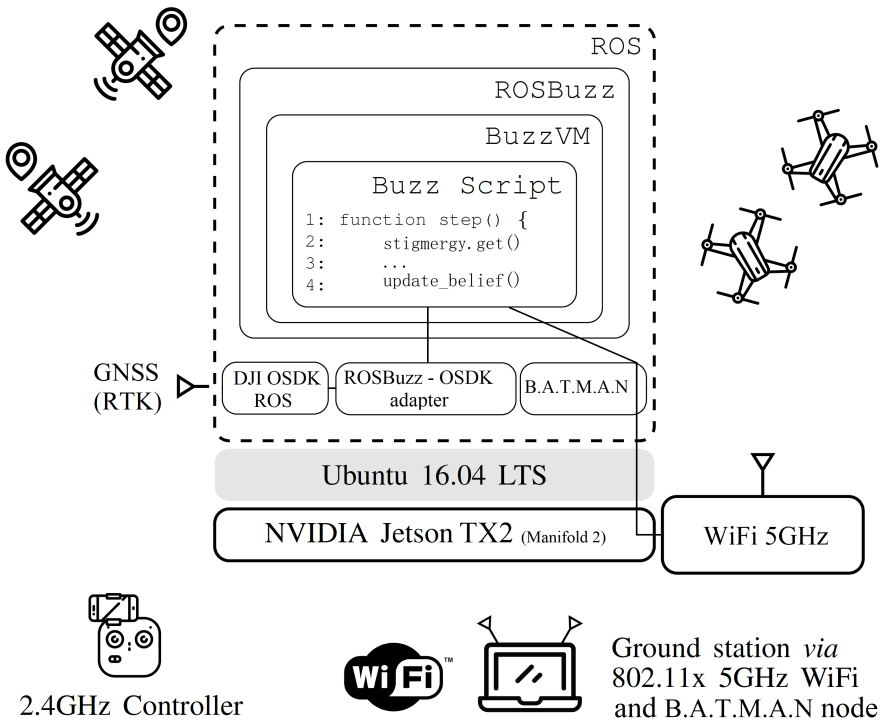

Figure 5.3 The control architecture. The global positioning system and (back-up) remote controllers join with the DJIs OSDK and flight controller to perform the decentralized behavioural Buzz script. The entire fleet runs the same script, interfacing with the Flight Control Unit (FCU) through DJI OSDK ROS and communication device (WiFi) through the Robot Operating System (ROS). The communication between swarm has been achieved by creating a B.A.T.M.A.N ad-hoc mesh network and using the DJI-Manifold 2 WiFi as the network hardware [68]

For the real-world deployment of our solution, we used DJI Matrice 300 RTK (M300 RTK) drones equipped with a Manifold 2 onboard computer which was connected to the drone through a serial connection. The M300 RTK is a powerful UAV platform offering an adaptive onboard software development kit (OSDK) for autonomous control of the aircraft. It uses an advanced flight controller system, a 6 directional sensing and positioning system and FPV camera. Thanks to these features, the drones were able to perform a basic collision avoidance routine during the flights.

The decentralized control of the drones is achieved using Buzz, a domain-specific language designed for programming multi-robot teams and swarms behaviors [9]. The software consists of three main layers: the Buzz control layer taking care of the algorithm logic, the ROSBuzz [69] layer responsible for the integration of the swarm-oriented programming language and its virtual machine (BVM) into the ROS environment, and the DJI OSDK layer that manages the flight controller and other UAV related features. The Buzz control layer is responsible for the system's behavior, using a Buzz script to implement the proposed algorithm, while sending hardware specific commands to the lower layers.

Our whole experimental system is based on ROS (Robot Operating System), an open-source and now standard software system for robotic development [70]. To link the Buzz control layer to ROS, we use ROSBuzz, an existing implementation of the BVM as a ROS node. ROSBuzz encapsulates all the BVM logic, publishes the Buzz script commands, and subscribes to external data such as sensor readings. A main feature of Buzz is the implementation of gossip-based situated communication among neighbors in a swarm. This feature is implemented in our system with batman-adv (Better Approach to Mobile Ad-hoc Networking), using a ROS node that manages the neighbors of each robot and broadcasts messages as needed by the Buzz script. Batman-adv is a layer 2-based protocol leveraging adaptive topology and routing to offer a robust ad-hoc networking solution [71]. As our algorithm assumes sporadic connectivity among the robots in the swarm, batman-adv can easily handle such a pattern, ensuring a reliable link between the robots when in communication range. The physical device supporting batman-adv is the 5 GHz WiFi antenna on the Manifold 2, set to communicate on a common IBSS ad-hoc wireless network.

The DJI OSDK layer is a DJI proprietary API that allows a developer to control the aircraft with a program. DJI proposes a version of the OSDK integrate with ROS that we used in our setup, and the UAVs are actuated via service calls. The interaction between the ROSBuzz ecosystem and the OSDK layer is taken care of by a custom adapter node. This node receives actuation commands from the Buzz script (in the form of topics) and sends flight controller-specific commands to the OSDK, as well as publishing the required data

for ROSBuzz's operation (e.g., GNSS readings). Such an architecture, allows for an easily portable code base. In fact, since the adapter is the only M300 dependent node, using the same algorithm with a different robotic platform only requires to write a similar adapter. Figure 5.3 synthesizes the described software architecture, which is completely open source (see github.com/mistlab).

5.4 Experimental Results: Simulations

To validate the presented algorithm and evaluate the necessary time required to find targets and obtain the final relay chain, we performed a series of tests in a simulated environment. We used 15, 20, and 25 robots while randomly varying their starting positions and the position of the targets. We also compare the performance of a random walk search to our dynamic belief map search, while using in both cases the presented search pattern.

5.4.1 Simulation Setup

We use ARGoS, a multi-physics robot simulator [72]. The experiments were performed on an ARGoS model of the Spiri Mu quadrotor from Spiri Robotics. Three foot-bot mobile robots [73] were randomly spawned and used as targets to be detected by the Spiris. To perform the detection, we simulate a basic sensing mechanism on the drones with a downward facing camera using blob detection. Figure 5.4 presents an example of a starting state of the simulation.

We first used a random walk exploration pattern [63] where each drone sequentially samples a 2D position and autonomously navigates to it(keeping the height constant). The second search method was the proposed dynamic belief space exploration. In both cases, we simulate the system until all the targets are detected and the relay network is fully formed. The total time needed to find the targets is the metric used for our evaluation.

5.4.2 Results

We perform six test cases with three arena sizes: *(i)* 15 drones with random search, *(ii)* 20 drones with random search, *(iii)* 25 drones with random search, *(iv)* 15 drones with belief space search, *(v)* 20 drones with belief space search, and *(vi)* 25 drones with belief space search. We used three arena sizes: **(a)** 20m x 20m, **(b)** 30m x 30m, and **(c)** 40m x 40m. To obtain statistically relevant results, each test case was executed 30 times, randomly assigning the initial positions for the drones, the targets' positions and the belief map if applicable. The other parameters, including the meeting point, were maintained constant through the

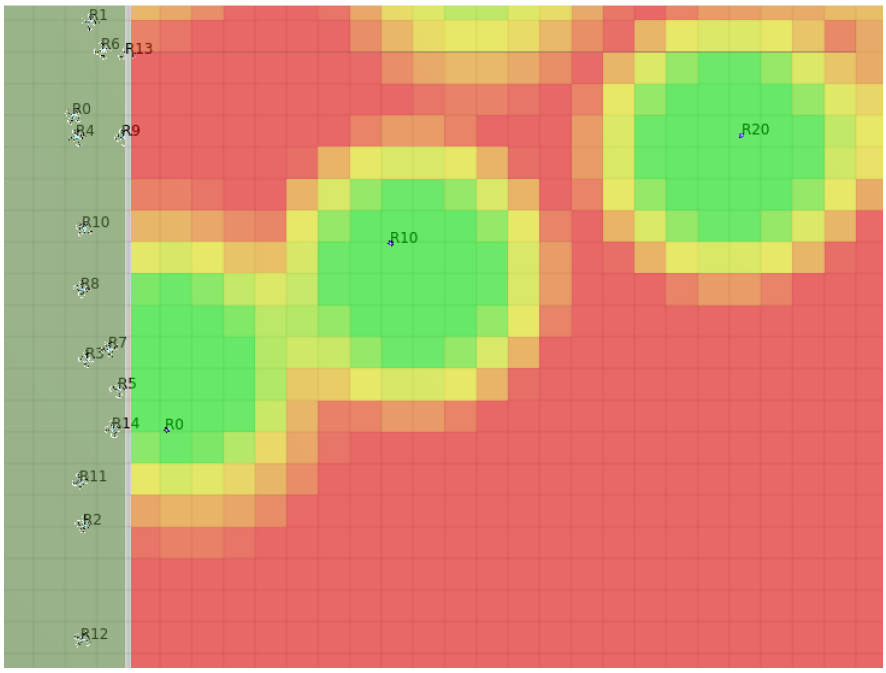

Figure 5.4 Initial simulation setup sample using 15 drones (R0 to R19 on the left side) with 3 targets (R0, R10, and R20 in the searching area). The searching area is represented as a belief 2D map where red represents a probability close to 0 to find the target and green represents a probability close to 1

experiments: 20 search steps, three targets and a communication range of 10 meters (not realistic, but distances can be simply scaled to realistic values). An output sample for a test with 15 drones and 3 targets is presented in Figure 5.5 where we can see the relay tree connecting the meeting point to the three targets. The figure also shows the remaining drones (not used in the relay) searching for any additional target.

For every experiment, we report the number of Buzz timesteps necessary to find the three targets. For reference, a Buzz timestep is 0.1s by default but it is fully configurable depending on the capabilities of the robots and communication system. Figures 5.6, 5.7 and 5.8 present the number of timesteps necessary to find the targets for different arena sizes and different number of drones. They show the results obtained with both the random search and the belief space search while increasing the number of drones and the arena size. From those results, we can see as expected for a swarm based SAR algorithm that the time to find the target decreases when the number of drones increases. This observation is due to the fact that a bigger search area can be covered with more drones, allowing them to find the targets faster.

Also, the time decreases in all cases when we use a belief space search in comparison to a

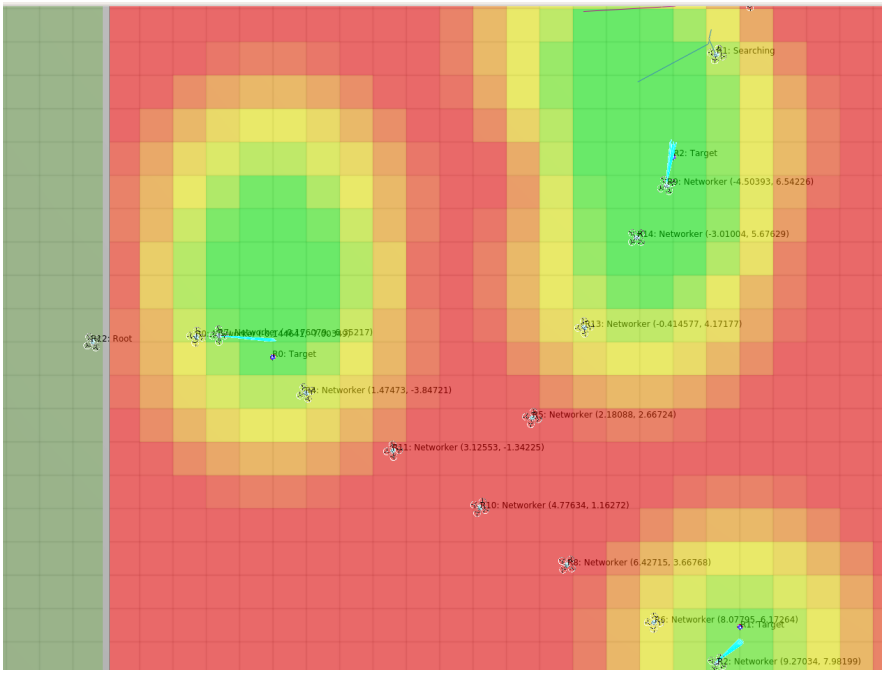

Figure 5.5 Possible final state for a test with 15 drones and 3 targets and a communication range of 4 meters. Each drone logging its state: root, networker or serching. The searching area is represented as a belief 2D map where red represents a probability close to 0 to find the target and green represents a probability close to 1

random walk search. In fact, by reducing step after step the searching area (thanks to the distributed belief information), our search method allows drones to converge faster towards the targets.

Finally, when the arena size increases, we can see that the time needed to find the targets increases as well (the sub graphs have different vertical scales). This can be explained by the sampling space which gets bigger with the arena. Thus, there are more options to explore before we can find the target wherever it might be.

To confirm the hypothesis that the belief space search would be faster than the random walk, in any case, we did a Bayesian paired samples t-test on the results by using [74] software. The results show that all data accept the hypothesis with the Bayesian Factor (BF) greater than 1 (BF min = 2.886, BF max = 66550.284).

Another hypothesis we postulated was that the search pattern that was designed should create the relay formation in nearly constant time. To verify that assertion, we considered for each experiment the total time spent by the root node in the auctioneer state. Figures 5.9, 5.10, and 5.11 summarize the results obtained from that metric. It shows in most cases a slightly higher formation time for the belief based search, with a higher standard deviation.

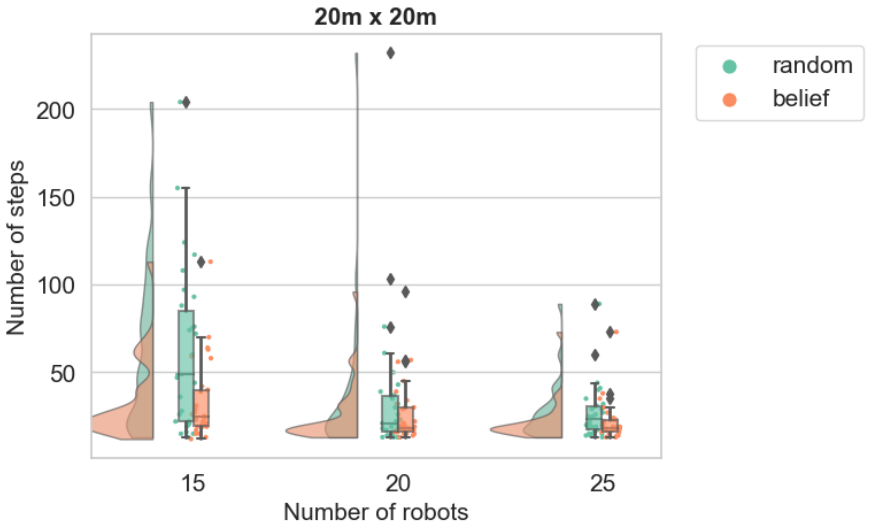

Figure 5.6 Number of timesteps to find 3 targets in a 20 m x 20 m arena. The results for the random search are in green and the results for the belief space search are in orange. The BF and the error of the Bayesian paired samples t-test for each setup are respectively 92.404, and 8.273e-5 for 15 robots, 2.886 and 0.001 for 20 robots, and 70.606, and 1.043e-4 for 25 robots.

Those results hints that the relay formation might take more time for the belief based search, refuting our hypothesis. To get a clearer answer, we once again did a Bayesian paired samples t-test on the statistical results considering the null hypothesis. The results show that 89% of the data reject the null hypothesis with the Bayesian Factor lower than 1 (BF min = 2.539e-6, BF max = 3.862).

We can also notice a constant, but slight increase of the formation time when the arena gets bigger. That fact is expected as the searching area increases. In fact, we have less chances of finding a drone in communication range of the root since they will now search further from the root. Another interesting result is that the formation time does not change much when we compare for the same arena size, different numbers of drones. Here, the number of agents does not impact the results as the auction time is based on whether a winner is found or not. Thus, more agents do not change the ability for the auctioneer to find a winner, assuming they are in range.

To explain the relay formation time difference between the two search methods, we made the hypothesis that the network usage is at the root of this observation. In fact, because of the dynamic update of the belief map, the network is heavily used in the belief based search. The update messages could delay the handling of the auction-related ones and therefore extend the formation time. To confirm this hypothesis, we compiled the bandwidth usage for the

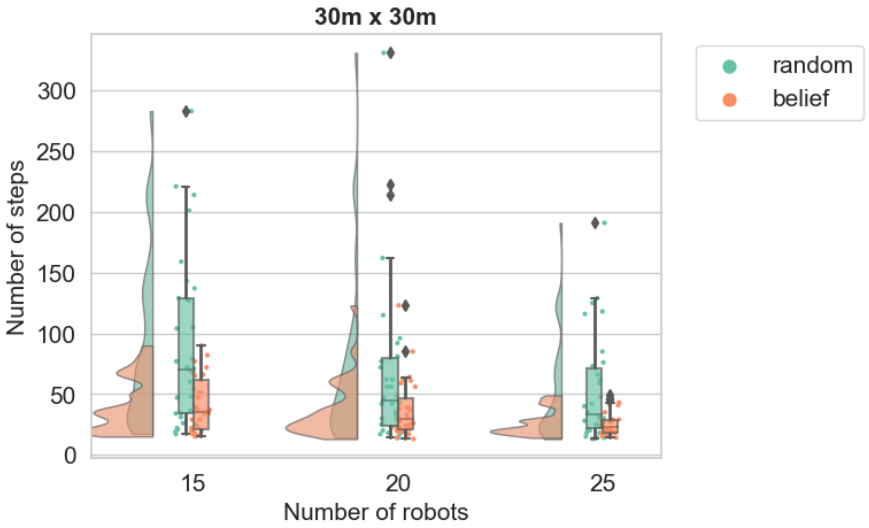

Figure 5.7 Number of timesteps to find 3 targets in a 30 m x 30 m arena. The results for the random search are in green and the results for the belief space search are in orange. The BF of the Bayesian paired samples t-test for each setup are respectively 66550.284, for 15 robots, 914.282 for 20 robots, and 25.268 for 25 robots. No error estimate can be given for 15 robots, however the error for 20 and 25 are respectively 4.609e-8 and 2.954e-5.

belief based search. For the sake of brevity, we present in figure 5.12 the result of a randomly selected experiment's configuration. We can see on the graph that for the belief-based search, the bandwidth is constantly increasing and we observe an homogeneous network usage by all the agents. In some rare occasions, when a drone would be out of range, we can observe a decrease of its network usage which increases drastically when he is back in range. Those observations confirmed our hypothesis by showing a fairly high network usage in the belief search.

It should also be noted that the number of search steps (set to 20 for our tests) could impact the time needed to have the relay chain, showing the importance for parameters tuning. For example, if the target is found by a drone at step 1, the first relays will not come to the meeting point before step 20 (unless they also find the target or are in *communication range* of the root). That fact can make the system stabilization unnecessarily long.

5.5 Experiments

The real-world experiments were performed in an outdoor football field with three DJI M300 quadcopters. The search area considered for the tests is a 36 x 36 meters field. The experimental platform was mainly presented in section 5.3.3. The decentralized control of the

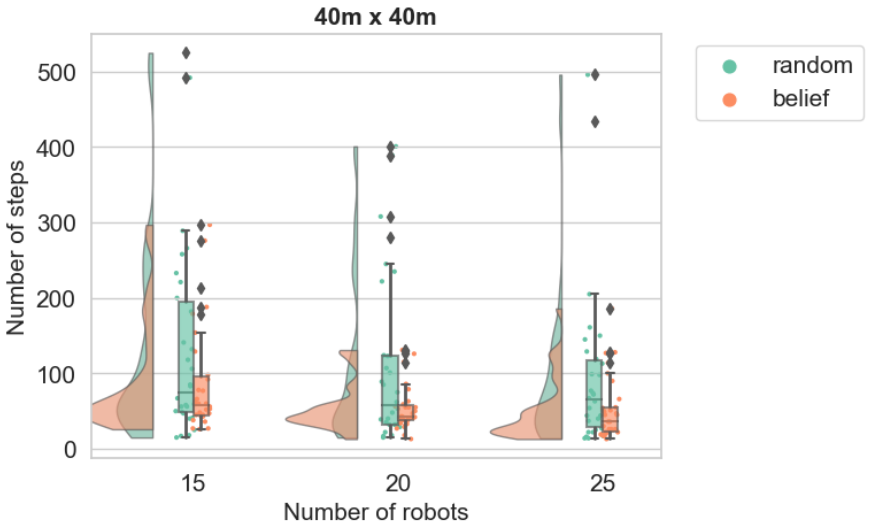

Figure 5.8 Number of timesteps to find 3 targets in a 40 m x 40 m arena. The results for the random search are in green and the results for the belief space search are in orange. The BF and the error of the Bayesian paired samples t-test for each setup are respectively 210.765 and 1.358e-6, for 15 robots, 203.165 and 3.369e-7 for 20 robots, 3577.307 for 25 robots. No error estimate can be given for 25 robots.

drones is achieved using the same Buzz scripts used in the simulation, with some minor changes related to the auction duration, and the M300 RTK's flight controller.

The tests were performed with both the random walk search and the distributed belief map search. Each of the experiments were performed three times to confirm the proper functioning of the algorithm. Figure 5.14 confirms the feasibility of our method. After performing the previously explained search pattern, the drones adopt, as seen in the picture, a relay formation. Given that we only have one target and three drones the relay was a line from the predefined rendezvous point to the target position. Sending a message from the agent at the target position, the information can now be relayed back to the root for analysis.

To give a better idea of the searching behavior during the experiments, we present in Figure 5.13 the path taken by the drones during two randomly selected runs (Right: random, Left: belief). That figure shows for the random search sparse lines, scattered randomly over the field, indicating a lack of pattern during the search. That lack of pattern can give fast discovery of the target under certain circumstances and a very long time in other cases. On the other hand, for a belief search, the lines are concentrated in areas with high belief.

Table 5.1 presents, for information, the recorded metrics for the belief map search. It shows the number of timesteps needed to find the target for each of the experiments. We obtained an average 1105 timesteps with the belief search, a value that is bigger than the ones obtained in

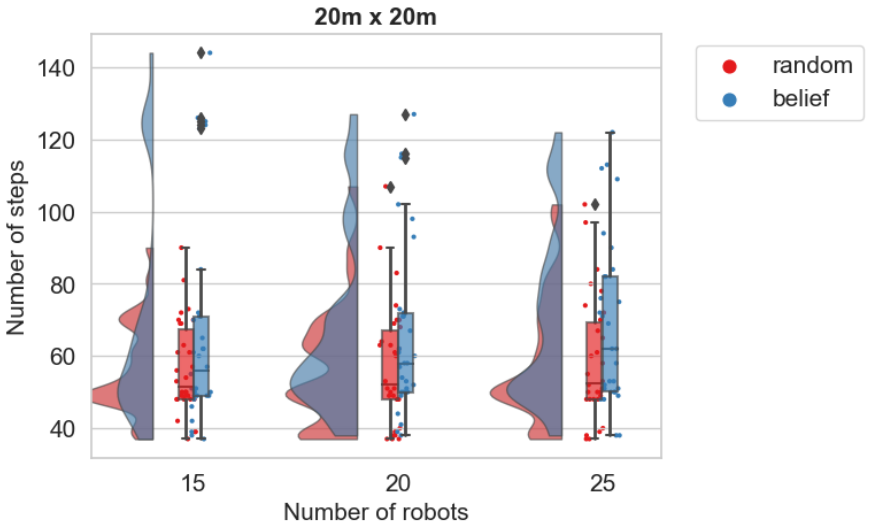

Figure 5.9 Number of timesteps needed per experiment to obtain the whole relay structure in a 20 m x 20 m arena. The results for the random search are in red and the results for the belief space search are in blue. The BF and the error of the Bayesian paired samples t-test for each setup are respectively 0.115 and 2.281e-6, for 15 robots, 8.070e-4 and 1.277e-6 for 20 robots, 2.539e-6 and 2.511e-8 for 25 robots.

Table 5.1 Real-world tests results

Experiment #	Target discovery (timesteps)	Relay formation (timesteps)
1	1159	301
2	863	324
3	1295	249

simulation. In fact, the difference with field experiments is that the number of steps required to get from one position to another is higher because of the velocity of the drones set to a relatively low value. The table also contains the necessary time to form a relay chain (as the one shown in figure 5.14). The mean value here is 295 timesteps.

Unlike the simulations, these experiments were only performed 3 times with a fixed target and a fixed belief map. That configuration can easily produce some biased results. Therefore, more real-world tests would be necessary to have statistically relevant data.

5.6 Conclusion and Future Works

In this paper, we presented a novel swarm robotic system for search and rescue operations in realistic scenarios. The proposed system is fully decentralized and robust to sporadic

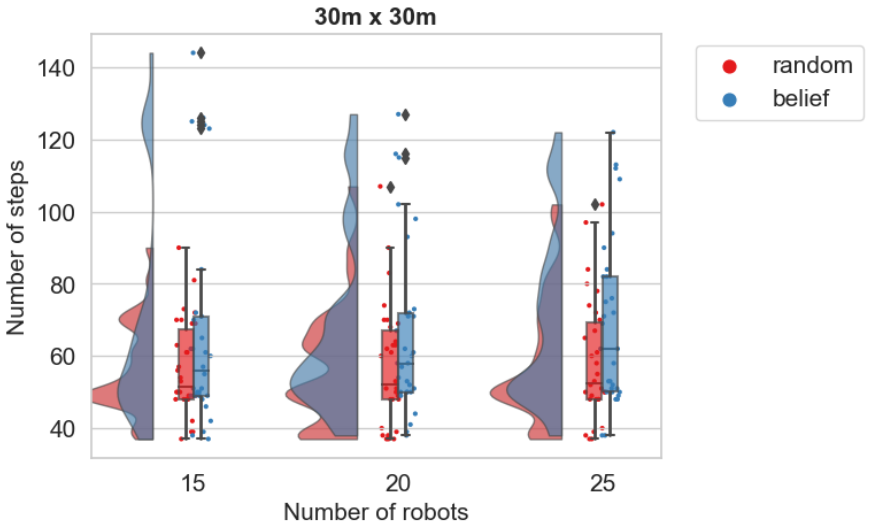

Figure 5.10 Number of timesteps needed per experiment to obtain the whole relay structure in a 30 m x 30 m arena. The results for the random search are in red and the results for the belief space search are in blue. The BF and the error of the Bayesian paired samples t-test for each setup are respectively 0.264 and 3.022e-6, for 15 robots, 3.327e-6 and 2.817e-8 for 20 robots, 0.022 and 1.896e-4 for 25 robots.

disconnections. It was also coupled with an implementation of a distributed belief map search algorithm leveraging prior knowledge on the area to realize a faster search. Based on the results obtained in simulation, we were able to confirm that our search method performs better than a random walk. The results obtained during our simulations and the real-world experiments, confirmed the feasibility of our approach. The deployed architecture also provide a modular, easily portable and scalable system, that could be used in other swarm deployments. In future work, we will leverage the distributed belief map in the search algorithm to perform dynamic updates on the target location belief. Indeed, we could model the target motion (e.g. due to the flow of a river) to update the prior over time.

Declarations

Funding This work was supported by the Fonds de recherche du Quebec – Nature et technologies (FRQNT) under grant 296737 and by the National Research Council Canada (NRC).

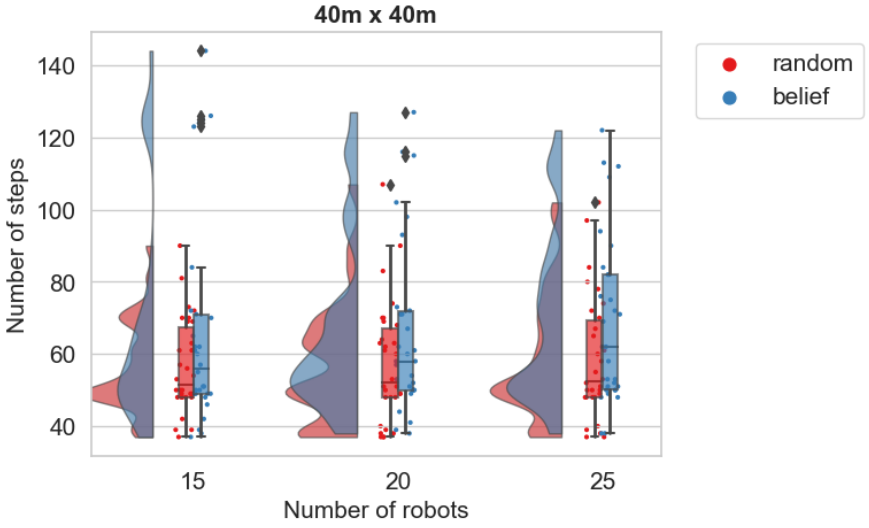

Figure 5.11 Number of timesteps needed per experiment to obtain the whole relay structure in a 40 m x 40 m arena. The results for the random search are in red and the results for the belief space search are in blue. The BF and the error of the Bayesian paired samples t-test for each setup are respectively 3.862 and 0.001, for 15 robots, 0.017 and 1.133e-4 for 20 robots, 0.469 and 3.806e-6 for 25 robots.

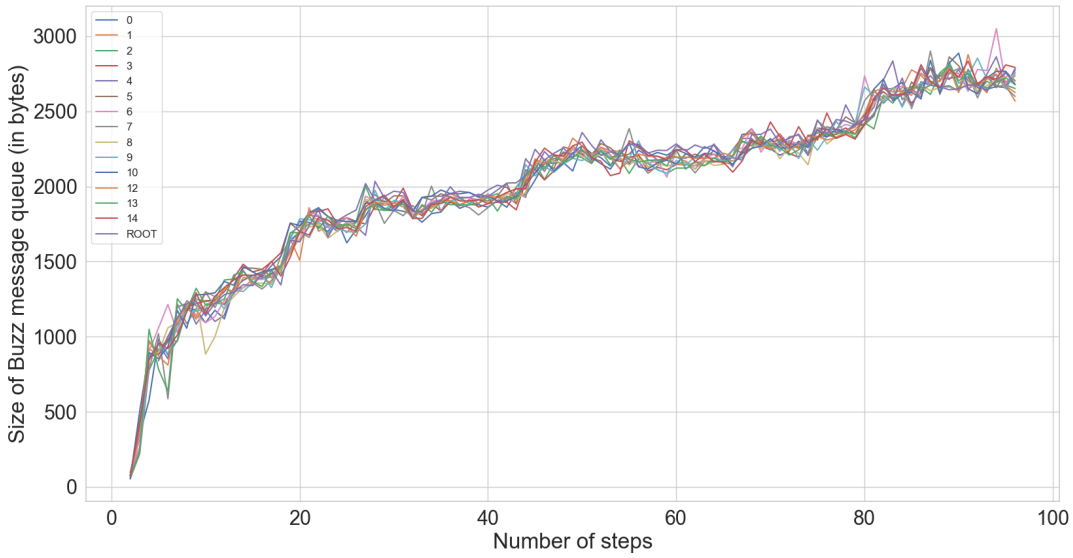

Figure 5.12 Bandwidth usage for 15 drones in a 20 x 20 arena (single experiment)

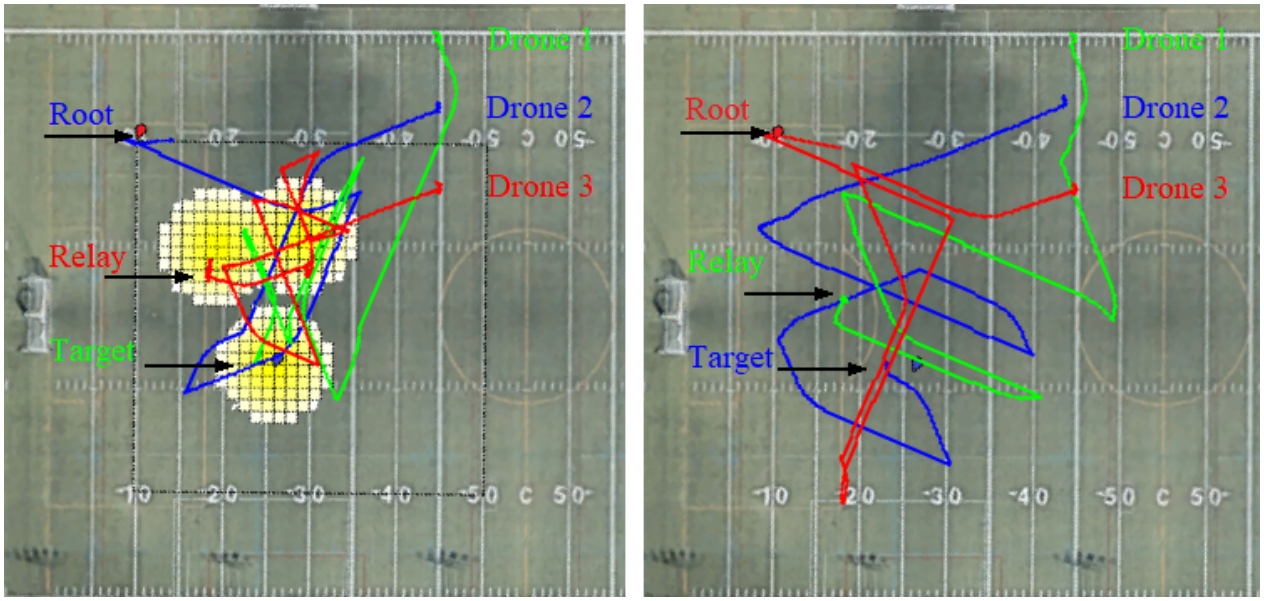

Figure 5.13 GPS trajectory during tests on a football filed. Right: random, Left: belief map, the yellow cells indicate the probability of having a target

Figure 5.14 Relay chain formation during real-world experiment

CHAPTER 6 GENERAL DISCUSSION

This chapter presents a discussion of the results obtained in the previous sections. It aims to recapitulate the main findings and to discuss the impact of those contributions to the research community.

In Chapter 4, BittyBuzz, the runtime environment for micro swarm application development, was presented. It extended the features of the DSL Buzz to resource constrained robotic systems and was integrated with three robots used for research in swarm robotics: Bitcraze Crazyflie, K-Team Kilobot, and Zooids. The results of the experiments proved that the system is functional, allowing developers to easily conceive well-known swarm applications. Providing domain specific features, BittyBuzz is expected to ease software development for micro swarms. Since the code base is publicly available, researchers can easily use it in their project as is or integrate it with new platforms as needed. As a functional runtime environment, BittyBuzz will be the building block of swarm applications in laboratory experiments.

Chapter 5 for its part, provides a real-world application of UAVs swarm: a search and rescue with sparse swarms. The proposed approach provides a solution to the sporadic connectivity problem that appears during UAVs search and rescue operations. It designs and implements an algorithm for SAR using ad-hoc networks and accepting sparsely connected swarms to ensure an efficient search operation, and a connected swarm during the rescue phase. The simulation and real-world experiments confirmed the correct operation of the algorithm. It also proposes a modular, easily portable and scalable software architecture controlled by the DSL Buzz, that can be use for the deployment of any robotic system. Such a system enables a wide range of real-world deployments for UAVs swarm. Researchers and specialists in the domain can therefore take inspiration from the open sourced architecture or directly use the code in their applications deployments.

In conclusion, BittyBuzz and the proposed system for real-world deployments, provides concrete systems that, respectively, bring the capacities of Buzz to micro swarms, and to macro swarms in the real-world.

CHAPTER 7 CONCLUSION

The main goal of the research work presented in this thesis was to bring swarm robotics closer to widespread real-world deployments of UAV swarms. It introduced two concrete tools to help reach that goal. BittyBuzz, a novel runtime platform, was developed and integrated with three robotic platforms including a nano quadrotor used in research. That environment is used to test algorithms in laboratory settings before they can be deployed in the real-world. A UAV swarm search and rescue application was also designed and deployed, while proposing a reusable framework for UAV swarm deployment. This section aim to recapitulate the contributions of the work in this thesis and provide a short discussion about their impact on the research community.

7.1 Summary of Works

The work presented in this thesis aimed to accomplish the research objectives stated in Section 1.3. First, BittyBuzz, the runtime platform presented in Chapter 4 was developed to be swarm oriented and is able to fit robots with as small as 32 KB of flash and 2 KB of RAM. It extended the features of the DSL Buzz to micro swarms and was integrated with three different robotic platforms used for research in swarm robotics. The results of the experiments proved that the main features of Buzz are implemented and operational in BittyBuzz, allowing developers to easily conceive well-known swarm applications. In fact, in addition to the performance evaluation of the communication means, two applications (exploration and bidding) were implemented as a proof of concept and showed good performance (in terms of space explored and bidding time). Providing domain specific features, BittyBuzz is expected to ease software development for micro swarms. Since the code base is publicly available, researchers can easily use it in their project as is or integrate it with new platforms as needed. It is already being used by a team of students from the University of Sherbrooke.

The second part of this thesis, presented in Chapter 5, focused on a UAV swarm application: search and rescue with sparse swarms. It mainly proposed an experimental robotic system for real-world deployment of UAV swarms, while developing and implementing an algorithm for SAR using ad-hoc networks and accepting sparsely connected swarms. The algorithm was tested in simulation and through real-world experiments and confirmed the scalability and correct operation of the solution in real life settings. A distributed belief map search was also implemented and proved to outperform a random walk search in simulation. The code base of this project is also available to the public. It provides a modular, easily portable

and scalable software architecture that can be use for the deployment of any robotic system, enabling a wide range of real-world deployments. Finally, the proposed approach provides a solution to sporadic connectivity during UAVs search and rescue operations and can be use for these applications to ensure connectivity.

7.2 Limitations

Even though all the objectives set have been achieved, the proposed solutions have some undeniable limitations. For instance, BittyBuzz only allows 8 swarms at the same time and one valid VS at the time. Those limitations are fine for most applications, but still limits the capabilities of the developer in some applications. Also, the stack sizes on the robotic platforms are an important limitation that reduce the scope of the applications that can be implemented. Concerning the proposed SAR applications, the real-world experiments require more runs and use cases. In fact, limited by the time and the number of drones, the experiments were only performed with 3 commercial drones with 3 runs per algorithm. Although these tests are enough to prove the operation of the system, widely adopting such a solution requires more testing. The search algorithms compared for the SAR application are also limited (only two). To give a better idea of the belief map exploration performance, we could have compared it to more algorithms (for example, frontier based exploration [75]). Finally, the proposed solution is based on a distributed belief map and assumes that the map is static (do not change in time), which is not necessarily the case in real life.

7.3 Future Research

Future research perspectives in the field of this thesis should first aim to mitigate, if possible, the mentioned limitations. For instance, more real-world experiments should be done and the algorithm should take into account dynamic targets. In addition to these, an interesting add-on to BittyBuzz would be the implementation of some important robot-specific function such as *goTo* natively so that a developer would have less custom closures to implement. As it is a new platform, the next steps for BittyBuzz will certainly include some bugs' resolution as its use increases in the research community.

REFERENCES

[1] M. Brambilla *et al.*, "Swarm robotics: a review from the swarm engineering perspective," *Swarm Intelligence*, vol. 7, no. 1, pp. 1–41, 2013.

[2] M. Erdelj and E. Natalizio, "Uav-assisted disaster management: Applications and open issues," in *2016 international conference on computing, networking and communications (ICNC)*. IEEE, 2016, pp. 1–5.

[3] P. Radoglou-Grammatikis *et al.*, "A compilation of uav applications for precision agriculture," *Computer Networks*, vol. 172, p. 107148, 2020.

[4] F. Nex and F. Remondino, "Uav for 3d mapping applications: a review," *Applied geomatics*, vol. 6, no. 1, pp. 1–15, 2014.

[5] M. Rubenstein *et al.*, "Kilobot: A low cost robot with scalable operations designed for collective behaviors," *Robotics and Autonomous Systems*, vol. 62, no. 7, pp. 966–975, 2014.

[6] M. Le Goc *et al.*, "Zooids: Building blocks for swarm user interfaces," in *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, 2016, pp. 97–109.

[7] W. Giernacki *et al.*, "Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering," in *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*. IEEE, 2017, pp. 37–42.

[8] A. Van Deursen and P. Klint, "Domain-specific language design requires feature descriptions," *Journal of computing and information technology*, vol. 10, no. 1, pp. 1–17, 2002.

[9] C. Pinciroli and G. Beltrame, "Buzz: An extensible programming language for heterogeneous swarm robotics," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 3794–3800.

[10] "What is a runtime environment (rte)? - definition from techopedia," https://www.techopedia.com/definition/5466/runtime-environment-rte, (Accessed on 03/07/2022).

[11] G. Gridling and B. Weiss, "Introduction to microcontrollers," *Vienna University of Technology Institute of Computer Engineering Embedded Computing Systems Group*, 2007.

[12] Y. Zhang *et al.*, "Hello edge: Keyword spotting on microcontrollers," *arXiv preprint arXiv:1711.07128*, 2017.

[13] A. Tahir *et al.*, "Swarms of unmanned aerial vehicles—a survey," *Journal of Industrial Information Integration*, vol. 16, p. 100106, 2019.

[14] S. D. Apostolidis *et al.*, "Cooperative multi-uav coverage mission planning platform for remote sensing applications," *Autonomous Robots*, pp. 1–28, 2022.

[15] E. T. Alotaibi, S. S. Alqefari, and A. Koubaa, "Lsar: Multi-uav collaboration for search and rescue missions," *IEEE Access*, vol. 7, pp. 55 817–55 832, 2019.

[16] L. Ruetten *et al.*, "Area-optimized uav swarm network for search and rescue operations," in *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 2020, pp. 0613–0618. [Online]. Available: http://dx.doi.org/10.1109/CCWC47524.2020.9031197

[17] H. Hourani, E. Hauck, and S. Jeschke, "Serendipity rendezvous as a mitigation of exploration's interruptibility for a team of robots," in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 2984–2991.

[18] M. Andries and F. Charpillet, "Multi-robot exploration of unknown environments with identification of exploration completion and post-exploration rendezvous using ant algorithms," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 5571–5578.

[19] ——, "Multi-robot taboo-list exploration of unknown structured environments," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 5195–5201.

[20] A. Belkadi, L. Ciarletta, and D. Theilliol, "Uavs fleet control design using distributed particle swarm optimization: A leaderless approach," in *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2016, pp. 364–371.

[21] G. Aguilar *et al.*, "A distributed algorithm for exploration of unknown environments with multiple robots," *Journal of Intelligent & Robotic Systems*, vol. 95, no. 3, pp. 1021–1040, 2019.

[22] F. Kobayashi, S. Sakai, and F. Kojima, "Sharing of exploring information using belief measure for multi robot exploration," in *2002 IEEE World Congress on Computational*

Intelligence. 2002 IEEE International Conference on Fuzzy Systems. FUZZ-IEEE'02. Proceedings (Cat. No. 02CH37291), vol. 2. IEEE, 2002, pp. 1544–1549.

[23] ——, "Determination of exploration target based on belief measure in multi-robot exploration," in *Proceedings 2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation. Computational Intelligence in Robotics and Automation for the New Millennium (Cat. No. 03EX694)*, vol. 3. IEEE, 2003, pp. 1545–1550.

[24] Y. Zhang *et al.*, "A symbolic-ai approach for uav exploration tasks," in *2021 7th International Conference on Automation, Robotics and Applications (ICARA)*. IEEE, 2021, pp. 101–105.

[25] A. Khan, E. Yanmaz, and B. Rinner, "Information merging in multi-uav cooperative search," in *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2014, pp. 3122–3129.

[26] D. Vielfaure *et al.*, "Dora: Distributed online risk-aware explorer," *arXiv preprint arXiv:2109.14551*, 2021.

[27] 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)*, 2016, pp. 287–294.

[28] D. Tarapore, R. Groß, and K.-P. Zauner, "Sparse robot swarms: moving swarms to real-world applications," *Frontiers in Robotics and AI*, vol. 7, p. 83, 2020.

[29] A. I. Hentati and L. C. Fourati, "Comprehensive survey of uavs communication networks," *Computer Standards & Interfaces*, vol. 72, p. 103451, 2020.

[30] R. Abu-Aisheh *et al.*, "Atlas: Exploration and mapping with a sparse swarm of networked iot robots," in *2020 16th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, 2020, pp. 338–342.

[31] J. Kim, P. Ladosz, and H. Oh, "Optimal communication relay positioning in mobile multi-node networks," *Robotics and Autonomous Systems*, vol. 129, p. 103517, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889019309145

[32] V. S. Varadharajan *et al.*, "Swarm relays: Distributed self-healing ground-and-air connectivity chains," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5347–5354, 2020.

[33] J. Li, "Throughput-aware flying communication relay network for disaster area search and rescue," in *Proceedings of the 2019 8th International Conference on Networks, Communication and Computing*, 2019, pp. 138–141. [Online]. Available: http://dx.doi.org/10.1145/3375998.3376038

[34] H. G. Zhang *et al.*, "Servo relays as distributed controllable-mobility network to maintain long-term stable links for mobile robot swarms," *Ad Hoc Networks*, vol. 117, p. 102497, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1570870521000597

[35] S. P. Yamaguchi *et al.*, "Autonomous position control of multi-unmanned aerial vehicle network designed for long range wireless data transmission," in *2017 IEEE/SICE International Symposium on System Integration (SII)*. IEEE, 2017, pp. 127–132. [Online]. Available: http://dx.doi.org/10.1109/SII.2017.8279200

[36] N. Majcherczyk *et al.*, "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.

[37] S. A. Çeltek, A. Durdu, and E. Kurnaz, "Design and simulation of the hierarchical tree topology based wireless drone networks," in *2018 International Conference on Artificial Intelligence and Data Processing (IDAP)*. IEEE, 2018, pp. 1–5. [Online]. Available: http://dx.doi.org/10.1109/IDAP.2018.8620755

[38] "Micropython - python for microcontrollers," https://micropython.org/, (Accessed on 07/01/2021).

[39] G. Gaspar *et al.*, "Development of iot applications based on the micropython platform for industry 4.0 implementation," in *2020 19th International Conference on Mechatronics-Mechatronika (ME)*. IEEE, 2020, pp. 1–7.

[40] M. Khamphroo *et al.*, "Integrating micropython-based educational mobile robot with wireless network," in *2017 9th International Conference on Information Technology and Electrical Engineering (ICITEE)*. IEEE, 2017, pp. 1–6.

[41] "Artoo platforms," http://artoo.io/documentation/guides/what_is_artoo/, (Accessed on 07/01/2021).

[42] "Github - sinatra/sinatra: Classy web-development dressed in a dsl (official / canonical repo)," https://github.com/sinatra/sinatra, (Accessed on 07/01/2021).

[43] "Zephyr Project | Applications," Sep 2020, [Online; accessed 2. Jul. 2021]. [Online]. Available: https://www.zephyrproject.org/learn-about/applications

[44] L. Peng *et al.*, "Emsbot: A modular framework supporting the development of swarm robotics applications," *International Journal of Advanced Robotic Systems*, vol. 13, no. 6, p. 1729881416663662, 2016.

[45] S. M. Trenkwalder *et al.*, "Openswarm: An event-driven embedded operating system for miniature robots," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 4483–4490.

[46] R. Ghosh *et al.*, "Koord: a language for programming and verifying distributed robotics application," *Proceedings of the ACM on Programming Languages*, vol. 4, no. OOPSLA, pp. 1–30, 2020.

[47] A. Desai *et al.*, "Drona: A framework for safe distributed mobile robotics," in *Proceedings of the 8th International Conference on Cyber-Physical Systems*, 2017, pp. 239–248.

[48] D. Pianini, M. Viroli, and J. Beal, "Protelis: Practical aggregate programming," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, 2015, pp. 1846–1853.

[49] W. Yi *et al.*, "An actor-based programming framework for swarm robotic systems," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 8012–8019.

[50] I. Navarro and F. Matía, "An introduction to swarm robotics," *International Scholarly Research Notices*, vol. 2013, 2013.

[51] A. R. Cheraghi, S. Shahzad, and K. Graffi, "Past, present, and future of swarm robotics," *arXiv preprint arXiv:2101.00671*, 2021.

[52] M. Rubenstein, C. Ahler, and R. Nagpal, "Kilobot: A low cost scalable robot system for collective behaviors," in *2012 IEEE international conference on robotics and automation*. IEEE, 2012, pp. 3293–3298.

[53] J. A. Preiss *et al.*, "Crazyswarm: A large nano-quadcopter swarm," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 3299–3304.

[54] F. Zambonelli and M. Mamei, "Spatial computing: An emerging paradigm for autonomic computing and communication," in *Workshop on Autonomic Communication*. Springer, 2004, pp. 44–57.

[55] "jucs05.pdf," https://www.lua.org/doc/jucs05.pdf, (Accessed on 07/03/2021).

[56] S. Feldman and D. Dechev, "A wait-free multi-producer multi-consumer ring buffer," *ACM SIGAPP Applied Computing Review*, vol. 15, no. 3, pp. 59–71, 2015.

[57] K. L. Busbee and D. Braunschweig, "String data type," *Programming Fundamentals*, 2018.

[58] Kilobots | self-organizing systems research group. [Online]. Available: https://ssr.seas.harvard.edu/kilobots

[59] Kilobot – k-team corporation. [Online]. Available: https://www.k-team.com/mobile-robotics-products/kilobot

[60] Crazyflie 2.1 – bitcraze store. [Online]. Available: https://store.bitcraze.io/products/crazyflie-2-1

[61] D. St-Onge *et al.*, "From design to deployment: decentralized coordination of heterogeneous robotic teams," *Frontiers in Robotics and AI*, vol. 7, p. 51, 2020.

[62] C. Tovey *et al.*, "The generation of bidding rules for auction-based robot coordination," in *Multi-Robot Systems. From Swarms to Intelligent Automata Volume III*. Springer, 2005, pp. 3–14.

[63] C. Dimidov, G. Oriolo, and V. Trianni, "Random walks in swarm robotics: an experiment with kilobots," in *International Conference on Swarm Intelligence*. Springer, 2016, pp. 185–196.

[64] L. Apvrille, T. Tanzi, and J.-L. Dugelay, "Autonomous drones for assisting rescue services within the context of natural disasters," in *2014 XXXIth URSI General Assembly and Scientific Symposium (URSI GASS)*. IEEE, 2014, pp. 1–4.

[65] J. Wubben *et al.*, "Toward secure, efficient, and seamless reconfiguration of uav swarm formations," in *2020 IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. IEEE, 2020, pp. 1–7.

[66] J. V. Nickerson, "Robots and humans reconvening," in *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No. 04CH37583)*, vol. 3. IEEE, 2004, pp. 2803–2808.

[67] B. P. Gerkey and M. J. Mataric, "Sold!: Auction methods for multirobot coordination," *IEEE transactions on robotics and automation*, vol. 18, no. 5, pp. 758–768, 2002.

[68] D. St-Onge *et al.*, "Planetary exploration with robot teams: Implementing higher autonomy with swarm intelligence," *IEEE Robotics & Automation Magazine*, vol. 27, no. 2, pp. 159–168, 2019. [Online]. Available: http://dx.doi.org/10.1109/MRA.2019.2940413

[69] ——, "Ros and buzz: consensus-based behaviors for heterogeneous teams," *arXiv preprint arXiv:1710.08843*, 2017.

[70] M. Quigley *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[71] K. Kiran *et al.*, "Experimental evaluation of batman and batman-adv routing protocols in a mobile testbed," in *TENCON 2018-2018 IEEE Region 10 Conference*. IEEE, 2018, pp. 1538–1543.

[72] C. Pinciroli *et al.*, "Argos: a modular, parallel, multi-engine simulator for multi-robot systems," *Swarm intelligence*, vol. 6, no. 4, pp. 271–295, 2012.

[73] M. Dorigo *et al.*, "Swarmanoid: a novel concept for the study of heterogeneous robotic swarms," *IEEE Robotics & Automation Magazine*, vol. 20, no. 4, pp. 60–71, 2013.

[74] JASP Team, "JASP (Version)[Computer software]," 2021. [Online]. Available: https://jasp-stats.org/

[75] A. Topiwala, P. Inani, and A. Kathpal, "Frontier based exploration for autonomous robot," *arXiv preprint arXiv:1806.03581*, 2018. [Online]. Available: https://arxiv.org/abs/1806.03581